NEA computer Science

Clocking System

By: William Puchy

Centre Number: 31210

Candidate Number: 7356

Contents

Analysis **3**

Description of the problem 3

Description of the current solution 5

Proposed Solution 7

Interview with employer: 9

Objectives 10

Initial Modelling 12

Initial ideas for client readers 12

Initial ideas for server 13

Initial ideas for the administrator role 13

Initial ideas for data modelling 14

Documented Design **15**

System Overview 15

Main Communication Overview 17

Database Design 18

Data Dictionary 18

Administration of Server 24

Client Design 32

Testing Section **42**

Overview 42

Video link 42

List of tests 42

Evaluation **45**

Evaluation of whole project 45

Evaluation against objectives 45

End-user Feedback 46

Response to end-user feedback 46

Improvements & Extensions 46

Technical Solution **47**

PROGRAM NAME: will_db.py 47

PROGRAM NAME: admin.py 53

PROGRAM NAME: client_gui.py 59

PROGRAM NAME: server_code.py 64

Analysis

Description of the problem

The client for my project is my parents' company. My parents are in the manufacturing business for kitchen and bathroom fittings. They manufacture components in the UK in a factory located in Rugby. They employ about 100 people to come into work and manufacture the fittings. The employees come into work at nine O'clock in the morning and leave at five O'clock in the evening. On a Friday, everyone from the factory leaves at four thirty giving them a shortened day. The business opens at seven O'clock in the morning allowing for IT to make sure there are no issues with the computer and network equipment, maintenance to make sure all of the manufacturing equipment is in working order and is not damaged and the cleaners to clean the office space and factory before employees get to work. The factory and office block is officially closed and locked up at eight in the evening.

Employees can start work from seven thirty if they want to however most employees come in at nine O'clock in the morning and leave at five. Overtime is added to the wages of the employees only if they do over 38 hours of work in the week. If an employee works from seven thirty in the morning to eight in the evening for two days, which amounts to 25 hours, this is not considered overtime as it is not yet more than 38 hours in the week worked.

My parents have a manufacturing business that makes kitchen fittings. Recently they have noticed that many people are getting late into work because of the single clocking in and out machine that is in the front entrance of the building. This means that people who come into the building from the other entrances such as the carpark or the street behind must wait in a queue of people all trying to clock in for nine o'clock in the morning. Some people are having to aim to be in the building 15 minutes before nine o'clock as the queue of people waiting to clock in can exceed 20 minutes. This has made some people be late to clocking in in the morning which has resulted in people being misinterpreted as coming into work 20 minutes late.

Access to the building is already controlled by RFID fobs with electromagnetic door locks - this could be something that I integrate with my system as it would save the employees having to have two methods of getting into work and would cost more money. There is no central server to unlock these doors as they are standalone units. To enter the building to clock in, the fob is placed in close proximity to the reader and the reader detects the fob and checks it against the pre-programmed list on the flash memory. Once it has done that, it allows access to the building by releasing the electromagnetic door lock. The door locks itself after 7 seconds of being unlocked. This gives the user enough time to go through the door and it lock before someone else can get through the door without a fob.

There is about 100 employees in the manufacturing building. Most of them use the entrance from the carpark which is about a 200m walk from the clocking in machine. The back-street door is about 300m away from the clocking in machine.

For payroll, the cards are all checked to see how many hours the individual has worked and to see if their hours have matched up to where they should be. If the person has worked more hours than they should have, that gets added to their pay as overtime. Overtime does not apply unless the person has done about 38 hours during the week. This must be manually checked by the database administrator every day to make sure records are where they should be with pay. The database structure that the administrators use that works with the payroll software is SQL. This works well as the administrators know how to use it and the other software that the company uses it too.

What I will be trying to achieve in this project is to make a better solution for clocking in and out at my parents' workplace. This would save many people time and could mean that people are not penalized for clocking into work 20 minutes late as there was a queue of many people all trying to use the same clocking in and out machine. This will allow people to get to work not 20 minutes early and have to wait in a queue but means that they will not have to all clock in and out in the same place causing congestion and a possible safety hazard. There would be multiple clocking in stations around the building meaning that people can clock in as soon as they get into the building.

The project will be able to accommodate the needs of the company with management of the employees which will be easier than reading off punch cards from the old machine.

With a new electronic system, there would be no need for employees to doubt their hours worked as with an electronic system, they can see their total hours for the week worked easier.

Description of the current solution

The building that people clock into is in the manufacturing warehouse which is separated from the admin offices of the building. The clocking in station is in the corridor to enter the manufacturing building. The corridor is perhaps about three people wide, so it is not very big. People in the morning line up to clock in with the machine. It is a paper stamping machine that clocks the exact time that the user has clocked in. The method of checking who is in the building is laborious and takes time to read all the cards. In the event of a fire, the lead safety officer checks through the cards to make sure everyone is present at the fire assembly point and that no one is left unaccounted for. This takes a lot of time to do and there is a possibility of losing some of the cards if not enough care is taken. This can difficult as it means that the only way that people are clocked in is on small bits of printed card.

This is the system that is currently used in the company. The way that it works, is that the employees pick up their card from the card holder above the machine and then push the card through the slot on the machine. The machine puts a hole for when they clocked in and when they clock out subsequently. This system works OK, however there is still the subsequent issue of how long it takes the employee to find their card , get it stamped by the machine and then put it back whilst the next person is trying to find theirs. There is also the risk that an employee could fake their time in or out by asking another coworker to clock them in in the morning and out in the evening. This can easily be done by finding their colleagues and stamping that along with theirs. This can lead to employees believing that they do not need to go into work and that they will still be getting payed.

The method of using a physical punch card is good however as it allows the employee to question the amount of money, they have been payed as they can directly compare it to the amount of hours that they have worked as it is stamped onto a piece of card that cannot be altered. Before the punch card machine, employees were meant to write down the number of hours however, this proved to be a mistake as this allowed employees to lie about the number of hours that they had actually worked leading to

over-paying the employees when they had actually worked less hours than what was recorded.

The advantages of this card punch system is that it is an initial low cost to the company that buys it and it is quite easy to maintain. The original cost of the unit was £225. There are no buttons that employees can press to change or modify the settings on the machine, the enclosure is built out of thick cast steel and it is easy to set up, all it needs inputted to the system is the correct time and date. The downsides to having this type of machine is that there is no option for getting the timesheet data as a digital format, the machine is very heavy and the system for keeping cards of every employee is impractical after a certain point as the admin for putting the cards into an analogue machine would take a long time to do every day.

The other methods out there include the ‘Safescan Time Moto’ brand. These clocking in and out stations cost from £309 (image on the left) to £539 (image on the right). The features range from only being able to enter a pin and use an RFID fob to being able to have an eye, face, or biometric scanner to clock in. The advantages of the Time Moto is that you can use multiple of these units around your building with them auto updating the status of users being either clocked in or not, employees can clock themselves in with an app or by using a web interface if that is required and enabled (this also sends the geolocation of the device clocking in so that if the employee was off site when they clocked in, the system would be notified that they were not currently in the building). The data on what employees are currently in is accessible via the web interface. This can allow administrators to know who is in the building if there was a fire alarm etc. The timesheet data of employees is stored on the cloud allowing people to access it anywhere however the data is encrypted meaning that in the case that someone did copy the database files, there would be no way of them getting into the files without decrypting the data.

The disadvantages of this system are that on the cheaper models, there is a maximum of 200 users. In a company like my parents, there are about 100 factory workers and about 35 office and admin staff. With a user constraint of only 65 extra users, there is not enough capacity in the system to allow scalability of the business. A system for clocking in should be scalable to many times more than the number of employees the company now has. Another disadvantage of the unit is that there are only two display sizes – a 2.8 inch screen or a 3.5 inch screen. This might be difficult to read by some of the older members of the factory or office staff who have a hard time reading small text on computer monitors. The means of connecting the machine to the network to allow cloud access is done by either wifi or ethernet. Unfortunately, there is not an option for having power over ethernet to power the unit – all of the ethernet ports in the building are power over ethernet giving anywhere from 3 watts all the way up to 30 watts of power over an ethernet cable whilst still operating at 1.0Gbps.

The storage on these units is only 128mb of flash memory up to 256mb of flash memory on the 800 series units. This is not enough memory when talking about hundreds or thousands of users that could be on multiple sites. The ram capacity on the units is less than the amount of storage at only 64mb of ram on the 600 series and 128mb of ram on the 800 series. Since ram is so cheap nowadays, I don't personally see why they didn't go with higher capacities of storage and ram. The onboard processors on the units are plenty enough for the task at hand which is basically sending data off to the cloud and showing a welcome and goodbye message.

Proposed Solution

My proposed solution would be to use a raspberry pi 4[1] as the clocking in and out station, have a USB Radio Frequency Identity (RFID) card reader attached to it along with a display large enough for people to see clearly with their information, the time and whether they are clocking in or out of work.

[1] raspberrypi.org

My solution would be better as it is much cheaper than the base model of the Timemoto whilst being able to be scaled up to hundreds of times as it would use a database and would not have ram and storage limitations (£34 for the raspberry pi, £5 for the RFID USB reader, £30 for the display and £10 for the POE adapter and a 3D printed custom case for about £6 which comes up to £85).

There would be no limitations to the number of users on the system as a; there is no software limitation, and b; there is enough storage and ram to support this. There would be the option of using WiFi and ethernet to connect the device to the network however there would also be the option to power the devices by Power Over Ethernet (POE) alleviating the hassle of having power and ethernet cables connected to the machine. The included processors on the raspberry pi 4 is a Broadcom BCM2711 which is an ARM based processor that is a quad core 64 bit SOC and can run up speeds up to 1.5 GHz. This is needed as the units will need to process the data for storage on the database and not just send the data off to the cloud where servers can process it all. There is 1GB of ram on the cheapest raspberry pi 4 which should be plenty enough of ram for sorting data. There is also two display outs meaning that you can if needed use a computer monitor instead of the smaller 4 inch display. This could be useful for external visitors or contractors coming in to the building who will need to sign in and out. The way that the clients would store the information would be on another raspberry pi that would act as a server. It would have more ram than the client computers and would be connected to full gigabit ethernet. The clients would send over the fob digits and the server would clock them in or out and save that on a database and send a message back to the client which would display to the user that they are clocked in. The

timesheets would be able to be exported from the server as a .csv file which will be able to be ingested by timesheet applications etc.

Interview with employer:

I explained the idea of the clocking in and out system to the employer and he said to me that it would work in his office complex however he said that he would want it as simple as possible for the employees to use – no buttons for them to press just put their badge up to the reader and it says that they are signed in. He said that he would also want it to integrate with door locking and unlocking via electromagnetic door locks. He said that a screen on the inside and outside of the entrance door that ask the user to put their badge up to the reader would be ideal as it lets the employees just clock in and out and not hold up others by going through menus – also the lack of buttons will lead to the system to be less likely to get broken.

I then talked to the client about the server that would be communicating with the nodes. I explained that the server could be integrated into one of the company's existing high performance servers by just running the server side program off it and giving it access to a shared folder – or the use of another raspberry pi with higher performance could be used. The client liked the idea that using an existing server could be used but thought that it would be more convenient to run the clocking in and out system off a separate system.

I then discussed with the client about the physical hardware of the system. I pitched the idea that the client-side nodes would be Raspberry Pi's with a 2.8" to 3.5" screen on top of them. I then asked about connectivity at the entrances to the buildings – there is power and ethernet there. I asked the client if a POE HAT on the raspberry pi would be useful as it will eliminate the power cable going into the Pi just needing an ethernet cable that is injected with power. The client said that that would be good as there is no immediate power nearby as the last system was hard wired into the circuit breaker. I asked the client how he would want the system to look – he said that he would want it to look minimalistic and not obtrusive. We settled on the case of the Raspberry Pi and RFID reader to be mounted inside a custom-made 3D printed case that would allow for mounting options on the wall of the building, space for an ethernet connection for power, clear visibility of the screen and not taking up too much room.

Some other requirements that the client had was that the unit could be easily updated by swapping a memory stick plugged into the machine using a protocol called runusb and that the unit would turn its self off at night one to two hours after the business day has ended and turn its self on in the morning. I explained to the client that the server

would be able to do this as Raspberry Pi's can respond to WOL (Wake On Lan). This can be configured on the server to wake up the clients. However, since the server and client could both be Raspberry Pi's, they do not draw much power so it may be just as easy to leave them on but turn the displays off after a period of inactivity. One of the programmable buttons on the side of the display could be programmed to turn on the display after it has gone to sleep (perhaps during the day where everyone has signed in during the morning and there is no one signing out until the evening. The client also wants the server to be able to host an internal website where the HR department will be able to access time sheet records of employees.

Objectives

1. The system will comprise of up to 10 Raspberry Pi client machines and servers
2. The system must hold data on client machine
    a. Mac Address
    b. IP Address
    c. Client ID
    d. Description
    e. Last Seen
3. The system must allow for new data to be added for new employee
4. The system must allow data on employees to be amended and deleted
5. Stations must turn on at 06:00
6. Stations must check server presence on network by pinging IP address
7. Stations must turn off at 21:00
8. When RFID badge is held within 2cm, the reader should register the correct RFID code
9. Client should send RFID code to server

10.The server should correctly receive RFID code

11.The server should check the RFID code against the database of existing users

12.If the server cannot find the RFID code in the list of existing users, it will send back a message to the stations saying ‘Card Invalid’

13.If the server finds the scanned RFID code in the database, the server will compare with last records and the time to determine if the person is clocking in or out

14.The server will update the records to show that the user is clocking in or out

15.The server will update the records to show the hours worked.

16.If found, server will update client by sending a message to acknowledge that the user is successful in clocking in or out

17.On clocking in or out, display on the station will show current total hours worked in the week for that employee

18.Administrators can view, amend or delete records

19.Administrators can print reports on hours worked by different employees

a. Reports can be exported as .CSV files to payroll software such as SAGE or XERO

20.At 20:30 all employees that have not clocked out on their own will be clocked out automatically however the time that will be put down for their clocking out will be 17:00 as that is the end of the standard working day

a. This will stop employees getting payed for extra hours if they forget to clock out

b. This will also put a warning record on their hours table indicating that they did not clock out that day

21.Output from the stations will be via 2-3.5” LED touchscreen displays

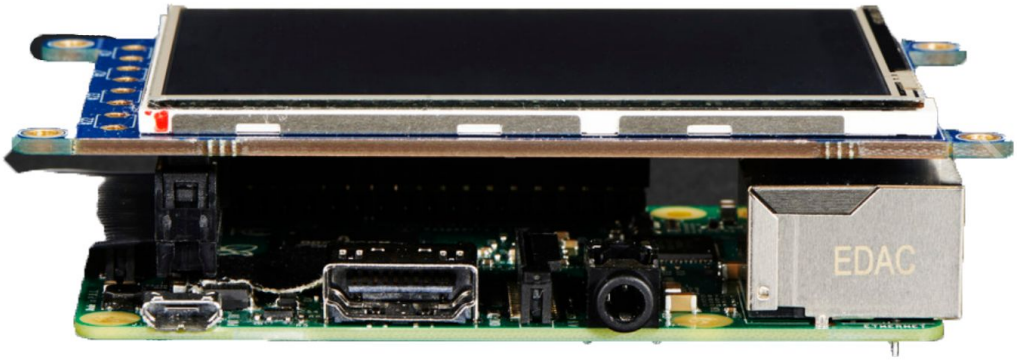
22.This is a full colour GUI capable screen however this system will keep things simple with only text and the company’s logo

23. The client stations will communicate with the central server via WiFi or over ethernet.

    a. If the system is using WiFi, power must be provided to the unit via a 5v 3A USB C cable

Initial Modelling

Initial ideas for client readers



The client readers will be Raspberry Pi's

When turned on they will try to contact the server for a 'client ID'

Once connected to the server they will be live for clocking in / out and this will be displayed on the screen by showing a suitable message

When the client receives the RFID fob tag number, it will send that information to the server along with the time and date.

When the server gets this information it will check through the database for the fob number and it will send back the name of the employee if it finds the fob number.

It will also send back the employee ID along with whether they are clocking in or out.

The raspberry pi will display their name, the time and if they are clocking in or out. This will be on a screen on the raspberry Pi.

Once it is verified that they are clocking in, the raspberry pi will send a pulse out on one of the data digital pins that will unlock the electromagnetic door lock system which requires a button to be pushed to unlock it.

An hour or two after the working day has finished, the clients will automatically shut down for the evening. In the morning an hour or two before work starts, the server will send a wake on lan request to wake the clients up

The clients can be powered by a power over ethernet switch to alleviate some cabling needed at the client end. This means that all the power and data can be delivered by an ethernet cable.

The clients can update their software by a .py file being changed on a memory stick plugged into the unit.

Initial ideas for server

The server will be another Raspberry Pi

The server will be listening out for data from clients

The server will have a database for storing data about the employees and their clocking in / out times

The server will be able to send Wake On Lan requests

The server will host a website

The server and clients will be able to be updated by changing the .py file on a memory stick plugged into it.

Initial ideas for the administrator role

The administrator will be someone in the company (most likely HR manager)

They will be able to log in to a web interface that shows them the statistics

The web interface will be secured with a username and password so unauthorized people cannot get onto it

The administrators can download the file of time sheets on the web interface.

At any time the administrator can generate a list of all employees currently checked in.

The administrator can search for an employee and bring up a list of their last check in / out times and a chart showing times for the last week / month.

Initial ideas for data modelling

The data that would have to be computed on the server would be:

The RFID badge number

Employee 1st name

Employee 2nd name

Employee ID

Clocking In or Out

Currently in building

Date

Time

Current hours

Hours for today

Hours for this week

These are the features that would have to be logged on the server as means of being able to tell who is in the building at any time. The employee ID would be assigned to an

employee and that data will be transmitted across the network allowing the clocking terminals to be able to greet the user as they clock in or out. The way that it would work is that the node will send across the badge number

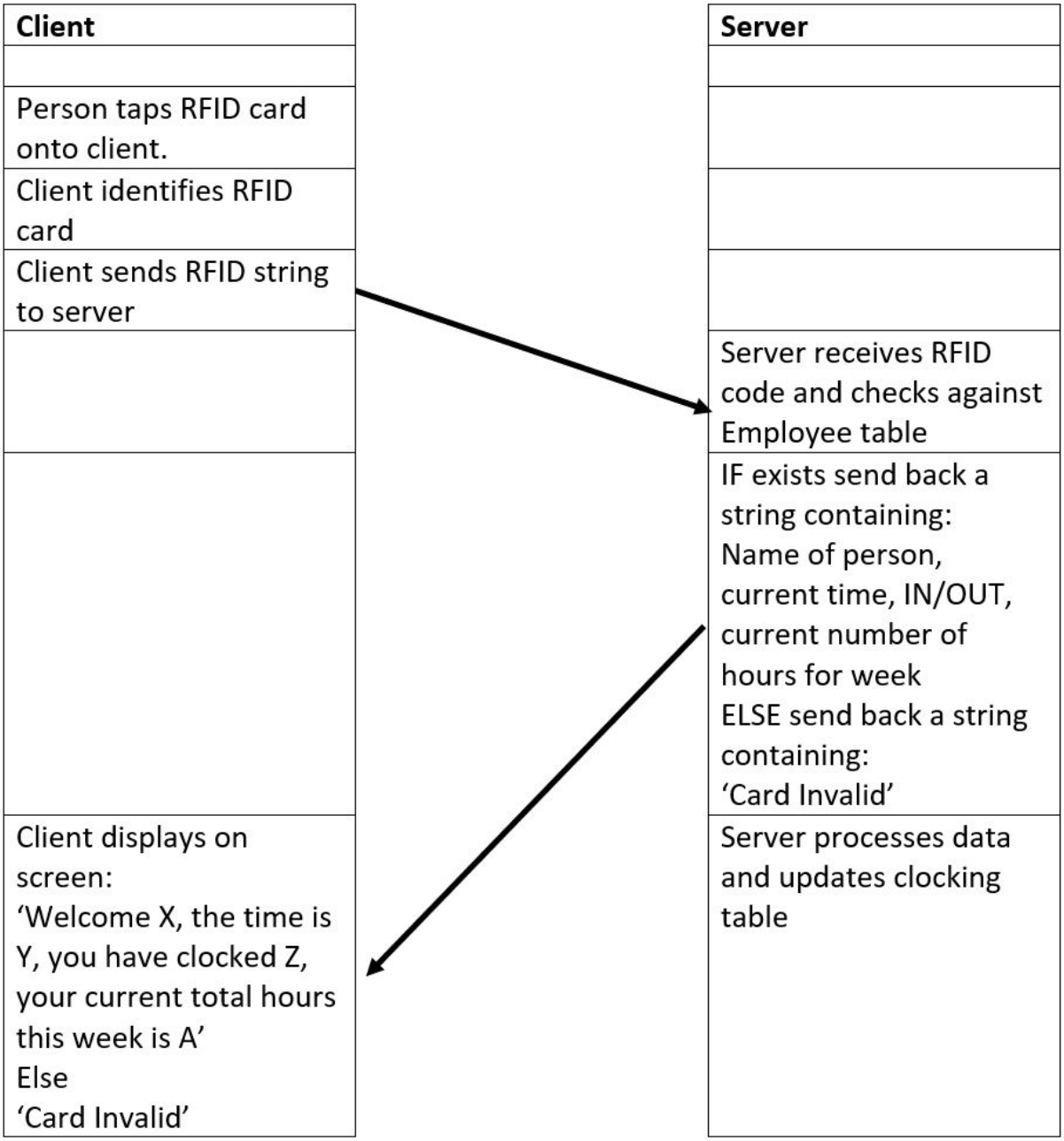
Documented Design

System Overview

What I will be coding and making will be a system that will allow employees of my parents company to intelligently clock in and out of work. This will bypass the current solution which is a punch card machine. This older system requires a lot of work to maintain and to collate all of the records. The new system I will be making will use a SQL database to allow the company staff to record their clocking in and outs digitally meaning they are much more likely to be paid correctly compared to going off the older system.

This new system will also allow the managers to access a list of everyone who is currently in the building. This would be good as it means in the event of a fire, there will be a list of everyone who has clocked into work in the morning but not clocked out yet. This system will use computers as remote clocking in stations to allow a better flow system in the building in the morning.

Main Communication Overview



Main Communication Overview

All traffic between server and client(s) will be encrypted. The proposed method is to use SHA256 from the hashlib Python library. Examples of this are discussed in the security section of the design section.

Protocol:

`COMMAND$DATA`

Messages sent between the client and server will be made up of two parts: COMMAND and DATA

COMMAND will be from the following list:

INIT - an initialisation request from the client (with a MAC address as data)

RFID - a RFID command from the client (with the RFID coda as the data)

OK - a command to the client to go to 'ready' state (with nothing as data)

CLOCKED - a command to the client that the RFID code has been process - the data will be a message to be displayed on the client

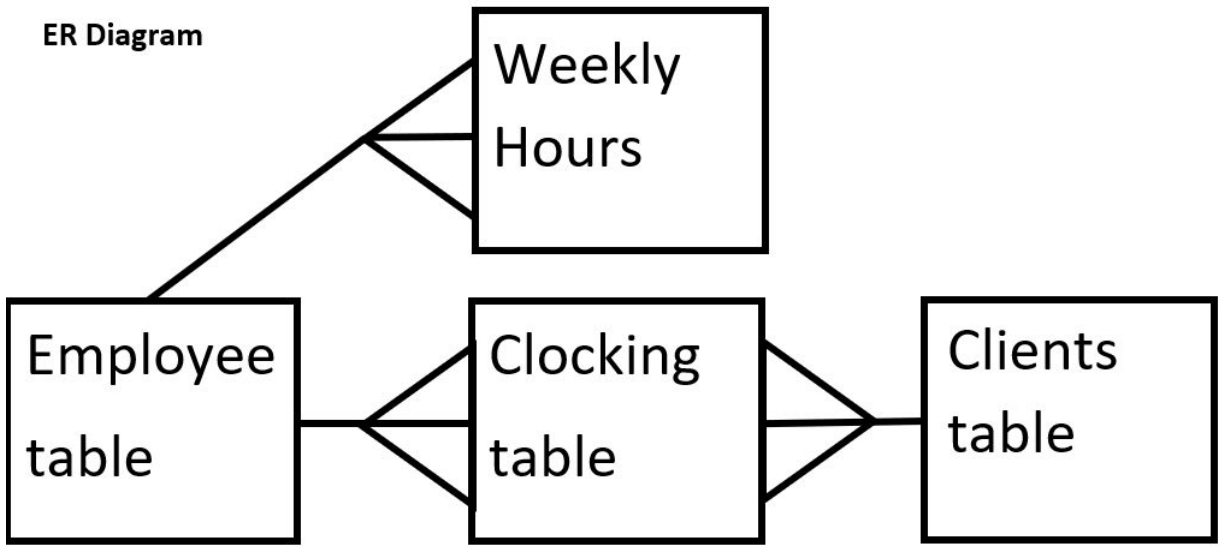
Networking:

For the networking side on python, I found a good tutorial on realpython.com (https://realpython.com/python-sockets/). This showed me how to make use of web sockets in python to create what is essentially a chat server. The way that it should work is that the Raspberry Pi should get it's IP Address from the local network's DHCP server. The server will have it's IP address static so that it is easily found and identifiable on the network. Communications between the clients and server will take place over port 1234 (I needed to make sure that there would be no other software using these ports as it would mean that wrong data gets sent to each program).

Database Design

The database design of this project is primarily to do with the data tables. This information will be stored on the server machine on the network that is running the server side program. This program will initialize the databases and will be able to create the databases for the program. These databases will be able to be exported from inside the program or copied off the server by going onto the SAMBA share. A username and password will be required for this as this data is confidential since it contains employee information such as names, departments and RFID badge numbers.

Data Dictionary



Employee

Field Name	Data Type	Validation	Example
ID	Integer	AutoNumber	1
RFID code	String	Unique	1234asdgn134
First Name	String	Presence Check	John
Surname	String	Presence Check	Smith
Department	String	Presence Check	Finance
Job Title	String	Presence Check	CSO

Hours Table (Possibly querying SQL database)

Field Name	Data Type	Validation	Example
Hour ID	Integer	AutoNumbers32	1
Week ID	Integer	ID for week	1
Employee ID	String	Unique	52
Total hours for week	String	Presence Check	40

Clocking Table

Field Name	Data Type	Validation	Example
Clocking ID	Integer	Unique	67475478
RFID code	String	Unique	1234asdgn134
Client ID	Integer	Presence Check	003
Date / Time	String	Presence Check	05/11/2019
In / Out	String	Presence Check	1

Clients Table

Field Name	Data Type	Validation	Example
Client ID	Integer	Unique	003
MAC Address	String	Unique	*00:0a:95:9d:68:16*
Description	String	Presence Check	Station in east hall
Last Seen	String	Automatic (added by code)	05/11/19
Authorised	Boolean	Automatic (added by code)	Default: False True

Pseudocode

SQL CODE FOR SETTING UP DB

CreateTableEmployee:

```
CREATE TABLE employee
Columns = employeeID, RFIDcode, Firstname, Surname, Department,
JobTitle
```

CreateTableHours:

```
CREATE TABLE hours
Columns = HourID, WeekID, EmployeeID, TotalHoursForWeek
```

CreateTableClocking:

```
CREATE TABLE clocking
Columns = ClockingID, RFIDcode, ClientID, ClockDate, Clocktime,
ComingIn
```

CreateTableClients

```
CREATE TABLE clients
Columns = ClientID, MacAddress, IpAddress, Description, LastSeen
```

AddClockingData:

```
Sqlstring = SELECT * FROM employee WHERE RFIDcode = {RFIDcode}
User = cursor-fetch one

If user[0] == None
```

```
Output = RFID code not in database

Else

Timenow = date-time.now

Current_time = timenow.time(H,M,S)

clockingIDnow = currentclockingID =+ 1

today = date.today

currentTime = day/month/year

sqlstring = INSERT INTO clocking (RFIDcode, clientID, clockdate,
clocktime, comingin

cursor.execute sqlstring
```

CLIENT SIDE CODE FOR DATA TRANSMISSION

ClientID = 1

S = socket of host

Host = socket.gethostname

Ip = socket.gethostbyname.host

Port = 9998

s.bing host,port

name = input(enter your name)

```
while true

messagetosend = input(str( user :: ))
```

Administration of Server

Server will always be running the server.py code

Run via SSH and the code: admin_server.py

Asks for a password (which is just hard-coded into Python)

[could use this library for password entry: https://pypi.org/project/getkey/]

Then displays a main menu:

```
Date: 12th Nov 2019          Time: 3:30pm

Current Server IP address: 192.168.31.42

Clients connected: 5

New clients: 1

MAIN MENU

1 – List Employees

2 – List clocking from today

3 – Who is currently 'in'

4 – Add, Remove, Edit employee

5 – List / Authorize, Reboot clients

6 – Enter SQL Database mode (advanced)

Q – Quit

**********************************
```

```
Please Log in to access administration
of server.

Password:
```

Further design for the admin side:

Some of the key components of the admin side are detailed below:

- SQL queries that will be needed

  - Queries needed are:
    - Who is In currently

      The design for this query can be seen below:

      ```
      SELECT a,b,c from d,e,f WHERE x,y,x
      ```

    - Hours for month of X employee

      The design for this query can be seen below:

    - Where do most employees clock in / out (what pi)

      The design for this query can be seen below:

- This is what the Administrator Console will look like for this query

The way that the query will display this for the user will be like this:

People who are on site at CURRENT:TIME

Employee 1

Employee 2

Employee 3

Employee 4

Employee 5

Employee 6

Employee 7

Employee 8

Employee 9

Employee 10

Employee 11

Employee 12

Employee 13

Employee 14

Employee 15

Press [N] for next page

Press [M] to go back to the menu

Exported as CSV file

- How will we page data?

  This is important as there might be up to 100 employees in the building and a console app can only show around 24 lines of text.

  - We can page data every 15 employees
  - Then press N for next page
  - Or press M to go back to the menu
  - Have text prompt at bottom of screen (see sketch above)

- How can we reboot clients? (research) – could send a message back to raspberry pi that client looks at and shutdown / resets….

  The admin side will use the setup network protocol to send a message to the client to say reboot.

  [how] Admin code will send ‘REBOOT’ Command to the client.

  When the client receives the reboot message it will run the reboot client as shown below:

```
import os          ← we need the OS library to perform a reboot
os.system('reboot')← placed in appropriate part of client code
```

- Time sheet for one employee

When the user accesses the timesheet, it will give the most recent month in a table like below. There will be a paging system showing only one month at a time. From the most recent month, the user can then press a key and the page will change to the month before that etc...

Week Start:	Monday	Tuesday	Wednesday	Thursday	Friday	Total
02/03	8	9	9	8	7	41
09/03	9	9	8	0	0	26
16/03	0	9	9	9	8	35
23/03	8	9	9	9	9	44
03/03						

SQL Queries for the admin program

These include:

1. Who is currently on site:
2. Current monthly hours for an employee
3. Total hours for an employee
4. Where do employees clock in / out the most

Who is currently on site at this moment?

look to see 'who is clocked in' - 'get all records for that date - order by time descending''
limit to 1 record per RFID code

This query works by selecting the rfid codes, employee surname, employee first name and clocking in status (1 or 0 for In or Out)

It then will need to inner join with the employee table to get the names as the clocking table only holds the RFID codes

The code will then need to fetch all data that is in the list

To make the array of people who are currently clocked in, the code will create a blank array then loop through and add people to the array if their latest clocking status is a 0

It will then display this for the user.

The code:

```
c = db.cursor()

    today = date.today()

    print(today)

    theTime = datetime.now()

    current_time = theTime.strftime("%H:%M")

    sqlQuery = "SELECT employee.rfidcode, employee.surname,
employee.firstname, clocking.comingIn, clocking.clocktime
FROM Clocking INNER JOIN employee ON employee.rfidcode =
clocking.rfidcode WHERE  clockdate=? order by clocktime
asc"

    c.execute( sqlQuery , (today,))

    allrows = c.fetchall()

    if len(allrows) == 0:

        print("There is no one clocked in currently at the
time of " + str(theTime) )

        allrows = []

    else:

        print(allrows)

        employee = []

        for row in allrows:

            if row[0] not in employee:
```

```
                employee.append(row[0])

                if row[3] ==1:

                    print(row[0], row[1], row[2], "is
currently on site at the time of " + str(theTime) + " NOW")
```

Hours of a specific employee for the current month:

- **Total hours of a specific employee:**

  Design:

  - Take in employee surname
  - Check for employee in DB
  - IF employee does not exist - error message
  - IF more than one employee with surname show a list and get user to enter ID for employee
  - Find out hours for each week using query below
  - Use a loop to work through returned records and sum up total hours

```
SELECT hours.totalhoursforweek, employee.surname,
employee.firstname, employee.employeeID FROM hours INNER
JOIN employee ON employee.employeeID = hours.employeeID
WHERE employee.surname= ** surname entered **
```

Where do employees clock in or out the most?

Design:

- Use the counting function to see how many records have the IP address for that clocking station

```
c = db.cursor()

    sqlquery = ("SELECT COUNT(clientID) ,clientID FROM clocking
group BY clientID")

    c.execute(sqlquery)

    allrows = c.fetchall()

    for row in allrows:

        print("Number of uses: " + str(row[0]) + "  " + "IP
Address: " + str(row[1]))

    print(allrows)
```

With these queries, the administrators of the company should easily be able to choose what they want to appear when they go onto the remote console for the server and run the administrator program. This would save them a lot of time since they do not need to go through a whole staff list checking off people as they see them – instead getting a generated list of everyone who was in and was not in.

This can also give them statistical data which can let them know if there is something that needs to be changed – for example, the usage of the data from ‘Most employee clock in location and out location’ might be able to tell the administrators to put another clocking station there to reduce waiting times for employees clocking in or out. It can also give personal statistics of how many hours a certain employee has worked compared to others and can even give a number of times that an employee has forgotten to clock in either before the standard 09:00 and 17:00. This can put a warning in the database which can let staff administrators know what is going on.

Client Design

When client boots up it gets a dynamic IP address and displays ‘INITIATING’ on screen (due to the fact that the client.py program will be set to automatically run upon boot).

It then communicates this to the server along with its MAC address so server now knows a client is online.

[If server has not seen this MAC address before it will add as a new client – new clients will not operate until authorized on the server]

Server responds with an acknowledgement so client can now starts processing and displays ‘READY’ on screen.

Client now awaits the recording of an RFID code.

Diagram showing process for booting client:

CLIENT **SERVER**

[Start Client]

[Display 'INITIATING' on screen]

[Send 'HELLO' to server along with MAC address]

[Receives 'HELLO' and MAC address]

[Searches for MAC address in clients tables]

[If found sends 'OK' back to client]
[If not found sends 'WAIT' back to client]

[If 'OK' received put 'Please tap fob to the reader' onto screen and start waiting for a RFID tag]
[If 'WAIT' received put 'TRY DIFFERENT READER' onto Screen]

[If fob scanned is successful, put 'Card Read Successful']

[When clocking in: put 'Hello

(EMPLOYEE NAME)

Clocking in at

TI:ME'

[When clocking out: put 'Goodbye

(EMPLOYEE NAME)

Clocking out at

TI:ME'

[If fob scanned is not in the database, put 'Card Read Unsuccessful Please try again']

Diagram showing process for accepting RFID tags:

1. Client receives rfid code through card reader
2. Client displays Card Read Successfully
3. Client sends off card RFID number to server
4. Server checks in database for RFID card number
5. If server doesn't find the card - responds to the client with message of invalid card
6. If the server finds the card's code in the database, it will look at their current status of clocking. If their last status was clocking out, the server will clock them in for the current internet time
7. Server will then respond back to the client with the employees name
8. The client will then display the employees name like '

   Hello

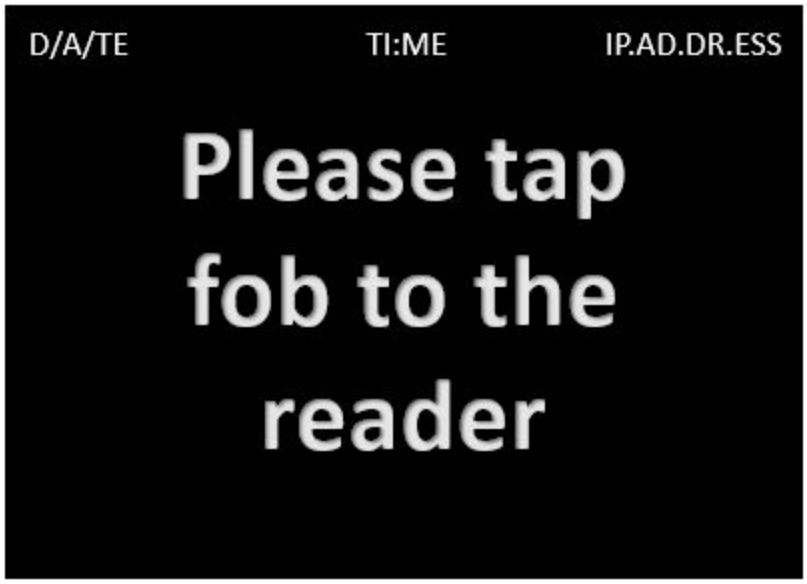
```
Employee Name

Clocking in at

TI:ME
```

Client User Interface Design

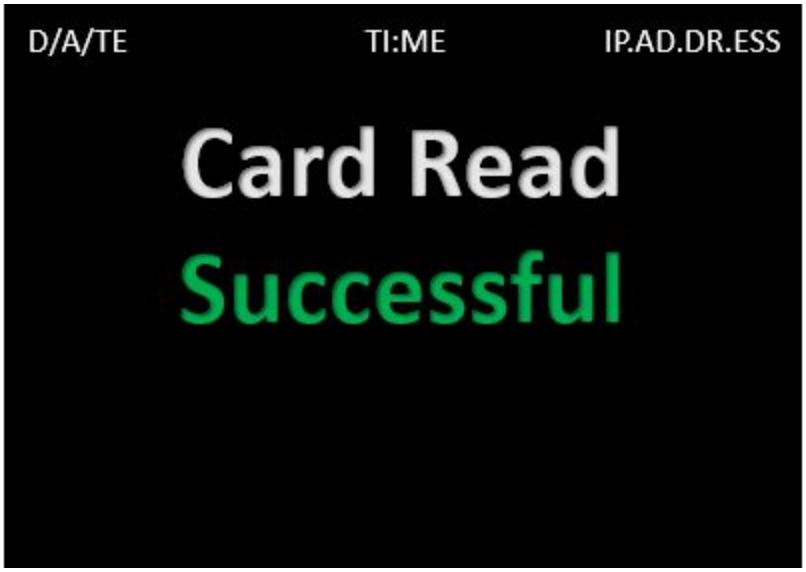
The client will be using these screens: https://shop.pimoroni.com/products/2-8-tft-touch-shield-for-arduino-with-resistive-touch-screen

The resolution for the screen is 240 x 320. This is the design that I have come up with the help and input of the client after we had an online meeting about how the displays should look for

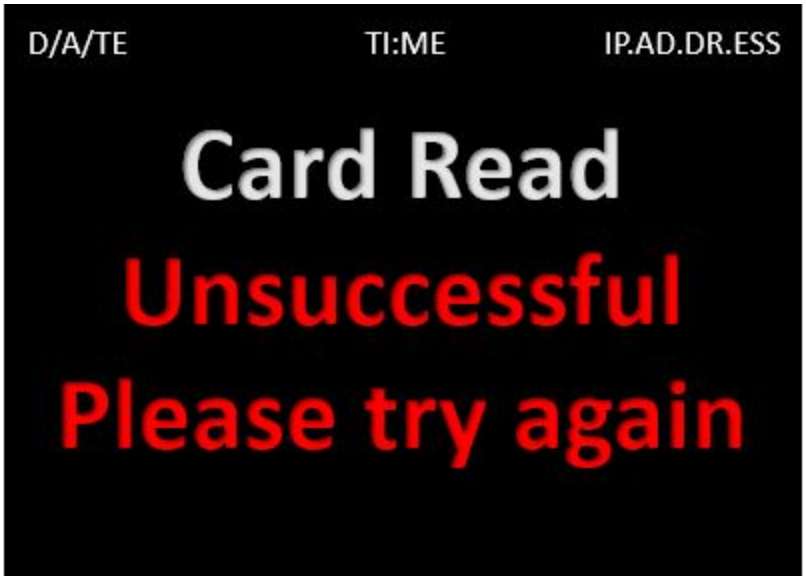
This image is what the client will display when looking for the server.



This is what the Raspberry Pi will display when waiting for a user to tap their RFID fob to the reader to clock in or out.



This is what will be displayed when the card that has been read is a valid card in the database. This will let the user know that their card is working correctly.



If the user receives this message, it means that the card that the user has used is not in the database and the user should either try again or contact human resources as their card is not valid.

This is the welcoming message that will be displayed when the user is clocking in. It will say their name and the time that they are clocking in. This will let them know that they are coming in at the correct time for work.

This is what will be displayed when the employees are leaving work. It will tell them the time that they are leaving so that they can check that against the amount of hours they worked during the day if needed.

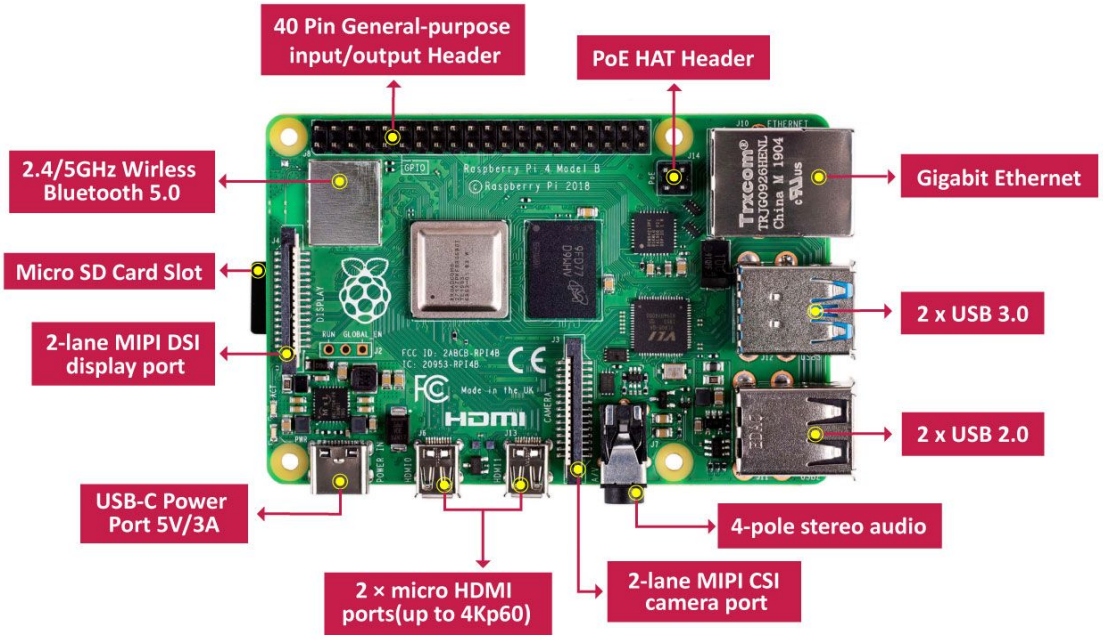
To implement the GUI on the client raspberry pi's I will make use of the simplegui library and the information based at: https://pysimplegui.readthedocs.io/en/latest/cookbook

Hardware selection for this project:

The hardware that I have selected for this project will be Raspberry Pi's. I have chosen these as they are decent specifications for the price that you are paying for them. The Raspberry Pi Model 4 2GB costs £44 (at the time of writing 14/11/19). This is very competitive pricing for a single board computer. The specifications are a 1.5GHz 64-bit quad-core CPU, 4 GHz and 5.0 GHz , 802.11ac wireless, Bluetooth 5.0, BLE, Gigabit Ethernet, 2x USB 3.0 ports; 2x USB 2.0 ports, 2 × micro-HDMI ports (up to 4kp60 supported), Micro-SD card slot for loading operating system and data storage, 5V DC via USB-C connector, Power over Ethernet (PoE) enabled. The 2GB model should be perfectly capable as one of the client computers as it has a decent quad core processor,

2GB of RAM and gigabit ethernet. The other parts that I would need for the project would be micro SD cards for storage of the operating system, POE adapter and a screen run off the GPIO headers. The price over all for each system would be £44 for the raspberry pi, + £5 for the SD card, + £5 for the POE adapter, £10 for a RFID reader and a final £32 for the screen. All in total that comes to £96 for one unit. That is much cheaper than the existing solutions that I have mentioned above. The server would be slightly different than this configuration. The server would have 4GB of RAM and would not need a screen or POE adapter however the SD card would need to be larger from 16GB to 128GB. This should allow for plenty of storage. In total the server edition would come up to £74. With the 4GB RAM model of Raspberry Pi at £54 and the 128GB SD card coming up to £20.

That is all the hardware that is needed for the system as the clients only send data to the server and show messages on the screen. The server processes the data however 4GB of RAM should be plenty.



This picture above is the features that the Raspberry Pi 4 has. As it says in the paragraph above, the higher ram and faster networking along with processor speed should mean that it should be ideal for this project.

Security

The encryption for my project would mainly be for the user data that is stored and transmitted across the network. This would be people's first name, surname, employee ID and RFID badge number. These are all critical pieces of data that cannot be leaked or read by unauthorized people. For example, the RFID badge number can be used to gain unauthorized access to the building by creating a RFID badge that has the same string that the actual employee does. This would not be good as someone else has the same copy of the badge as the employee who works for the company. This would enable them into the building during work hours.

A method of encryption for python and SQL databases is the SHA256 library that is available for download from the internet. This enables me to encrypt the data that is a; flowing over the network and b; the data on the databases. If the data is encrypted, it would not allow someone to easily access the data going across the network by packet sniffing, likewise with the database that is secured, the unauthorized person would not be able to read information off the database as plain text.

Data Structures

The data structures for the program will be something along the lines of:

User Input of fob à Client sends data to serverà server looks up data à IF server sees data à responds to client with name and time IF server doesn't see data à responds to client with data not found à data to client shows their name and time clocking in or out à the server adds to the clocking in or out table to show their presence for the day.

The data needing to be transported over the network is: EmployeeFirstName, EmployeeSecondName, RfidBadgeNumber, TimeCurrent, ClockingStatus, EmployeeID.

The tables will be: WeeklyHoursTbl, EmployeeHoursTbl (daily hours), ClockingTbl and finally ClientsTbl. These tables will allow me to query all of the above functions efficiently as I will be able to make sure that I do not mix any of the data up between tables.

These queries will be able to be downloaded from the server as individual .csv files. This will allow the administrators to put the data in whatever format they need for their payroll software as it most likely needs the data in a specified format so that it can calculate how many hours have been done by each employee and therefore how much money they get paid.

For a company of about 100 people, the amount of data being stored will be:

```
CHAR(10) RfidBadgeNumber

VARCHAR(20) EmployeeFirstName

VARCHAR(20) EmployeeSecondName



VARCHAR(7) EmployeeID

TIME(4) TimeCurrent

VARCHAR(4) ClockingStatus
```

.

Overall, the data needing to be stored every day will not be this much as the data on the tables will be using an employee ID not the employee first name and second name. In a company of 100 people, this should save quite a lot of storage over a few years as unnecessary data is not being put into a table twice a day. However, the amount of data held for a single employee twice a day is still a very small amount of data as it is only up to (EmployeeID – 7CH, RfidBadgeNumber – 10, TimeCurrent – 4 and finally ClockingStatus – 1) a total of up to 22 characters – which is 22 bytes. For one day the

amount of data being written is: 22 bytes * 100 and *2 for twice a day – which amounts to 2.2Kb.

Writing these files will require that every stage of the writing process is saved – meaning that for every employee clock in or out, that individual record must be added and closed before anything else can happen to the database. Doing this removes ambiguity and allows for smoother database operation. The database will not allow two things to be edited on it whilst one is open. This will make sure that no records are missing, corrupted or not saved as this could mean losing an employees hours for the week record which is their pay. This would also have to be held in a secure location with a password on the database to prevent unwanted access to the database meaning that people can change their hour records and earn more money than they would otherwise. The way that I would go about doing this is by using a built in feature of SQL called record locking. This allows one record to be changed by one person at a time and means that multiple people cannot edit the record. As I have said above, this can prevent

The amount of data being written is quite critical as the Raspberry Pi's use a micro SD card for their storage. These are very small SSDs and like standard SSDs have a read and write limit – however this limit is a lot lower than a standard SSD due to the size and compactness of the SD card.

Testing Section

Overview

I will conduct the tests in an environment close to how the system would be in use in a real working environment. I will do this as it means that I should be able to point out any flaws that would mean that the system might not function as well. This means that I will be testing the system with data that is meant to be inputted, boundary data which is on the edge of realistic data and finally erroneous data which is completely different to what the inputs should be.

Video link

Youtube video of full video testing:
https://www.youtube.com/watch?v=TDN8lrnu_nE

List of tests

Test Number	**Description of test**	**Outcome of test**	**Evidence**
1	Using a valid RFID card on the reader with a registered employee for the first time that day	Client should say: 1: Card read successfully 2: Hello Employee Clocking in at TI:ME'	Video: Time link: 0:10
2	Using an invalid RFID card on the reader with an unregistered employee	Client should say 'Card read unsuccessful	Video: Time link: 1:00
3	Using a valid RFID card for the 2nd time that day with a registered employee	Client should say: 1: Card read successfully 2:Goodbye Employee Clocking out at TI:ME	Video: Time link: 2:37
4	Administrator program is able to run SQL queries	Administrator will be able to run SQL queries on	Video: Time link: 3:26

		employees etc	
4a	If a user clocks into work at the start of the day the database will record them coming in	Database should update showing that they have clocked in	Video: Time link: 4:00
4b	If a user clocks out of work, the database will record them finishing work for the day	Database should update showing that they have clocked out	Video: Time link: 4:38
4c	By going onto the administrator program and selecting the option for the on site list - this will display everyone who is on site	SQL query should show everyone who has clocked in but not out yet that day	Video: Time link: 5:08
4d	By going onto the administrator program and selecting the option for the hours of specific employee per month - this will show the amount of hours for the employee	SQL query should show the hours of that employee for their most recent month at the company	Video: Time link: 5:47
4e	By going onto the administrator program and selecting the option of total hours for a specific employee - this should show the total amount of hours for the specific employee	SQL query should show the hours for that employee's whole time at the company	Video: Time link: 6:10
4f	By going onto the administrator program and selecting the option for where do employees clock in or out the most - this will show the most used clocking stations	SQL query to find IP address with the most uses	Video: Time link: 6:33
4g	By going onto the administrator program and selecting the option for SQL - this should allow the user to type and execute SQL	SQL queries to be typed in and executed in the database	Video: Time link: 6:49
5	Make sure that the exported CSV files are correct in that the columns and data types are correct	CSV file should be understandable and correctly made	Video: Time link: 7:19
6	Make sure that client will boot up	The raspberry pi	Video:

	and get to the correct state to start reading cards	should have the ‘Please tap fob to the reader’ screen	Time link: 8:46

Evaluation

Evaluation of whole project

I think that the project on a whole went successfully. I think this because I ended up making a system that can function as what my client has wanted. Looking back, I think that starting the coding earlier and by putting a lot more effort in at the start could have meant that I would have been done by this point. I did end up using a few tutorials on how to do some aspects of the coding such as sockets in python and the graphical user interface for the clients. Unfortunately, I did end up running out of time to finish many of the features that I would have wanted to. These include mainly getting the Raspberry Pi's integrated with the system - this involved getting the scanner set up and the screens.

Evaluation against objectives

I was able to get most of the SQL queries done apart from the query for monthly hours for an employee. The administrator program ran quite well and had a clear and easy to use interface. The networking side of the project works perfectly. The client and server programs work well and are able to connect and sync up with the database which includes fetching and writing information to it.

OBJECTIVE:

1: Using a valid RFID card on the reader with a registered employee for the first time that day

> I was able to partially complete this objective. I proved that I can read an RFID code from a card. I also proved that the client code can pass an RFID code to the server to be processed. These two parts, however, were not integrated due to the project time ending early.

2: Using a valid RFID card on the reader with a registered employee for the first time that day

> I was able to complete this objective as the first time that a user clocks in, the system will clock them in for the day as it looks at the previous clocking in state and does the opposite to it.

3: By going onto the administrator program and selecting the option for the on site list - this will display everyone who is on site

> This objective is completed as the program will export and show the user the list of who is currently clocked in. This is confirmed working as I showed my client the program and they were pleased by how it works.

[Rest of objectives would have been evaluated against if project time did not end early]

End-user Feedback

This is what my end user had to say:

I think that it is good that you can access the administrator code from a laptop as it means that you are able to access the list of everyone in the building without being inside the building. The exporting of CSV files is a nice feature as it means that for a fire drill we can print out the table and have it on hand for when we do the registers. It would be good to have hours for the pay periods such as being 2 weeks or monthly. I would want a variable in the code so that we can change the pay periods. Having the scanning of the RFID card would be nice. I think the user interface is very simple for the administrator program - but then again it doesn't need to be complex as it conveys the information sufficiently. I do like the interface for the Raspberry Pi as it is clear and concise. It is clear to see what is happening and shows the employees what is happening..

Response to end-user feedback

For the hours in the pay period, I would have been able to do this before the official deadline - however due to the current situation with this country and the world, I was unable to complete this. I would have had the option in the code for looking at the time sheets for an employee for every two weeks or every calendar month. Same story with the scanners and Raspberry Pi's. I would have been able to get them working with the system however I did need to finish the code before I could really start working on them.

Improvements & Extensions

If I was to improve this project, I would add security to the networking aspect of it. I would do this as it is possible for unauthorised users to intercept the information from the server to the clients which would contain information such as people's names, their RFID number and what their clocking ID is. This would leave the system susceptible to people without the proper credentials changing information on the databases etc. I would solve this by using an encryption library on python to make the information going across the network not plain text as I don't want the information to be readable by people.

Technical Solution

On the following pages the code for the project is provided.
The table below identifies where in the code certain segments of code can be found.

Evidence of skill	**Group**	**Page**
Multi-table SQL queries (with JOINS)	A	55-57
Networking skills (use of Python sockets)	A	59-62
CSV work	B	50,55,57
Complex client-server model (processing both on client and server)	A	59-67
Complex multi-table database	A	47-49
Use of libraries (such as pysimplegui for a gui)	B	59-64

PROGRAM NAME: will_db.py

```
import sqlite3
import os
from datetime import datetime, date

DEBUG = True

db = sqlite3.connect('mydb.db')
cursor = db.cursor()

def CreateTableEmployee():
    try:
        cursor.execute('''
    CREATE TABLE employee( employeeID int PRIMARY KEY, RFIDcode varchar(10),
firstname varchar(50),
    surname varchar(50), department varchar(50), jobtitle varchar(50))
    ''')
        db.commit()
```

```
    except:
        if DEBUG:
            print('\n [DEBUG] TableEmployee already exists. \n ')



def CreateTableHours():
    try:
        cursor.execute('''
        CREATE TABLE hours( hourID int PRIMARY KEY, weekID int, employeeID int,
totalhoursforweek real)
        ''')
        db.commit()
    except:
        if DEBUG:
            print('\n [DEBUG] TableHours already exists. \n ')


def CreateTableClocking():
    try:
        cursor.execute('''
        CREATE TABLE clocking( clockingID integer PRIMARY KEY AUTOINCREMENT,
RFIDcode varchar(10), clientID int, clockdate datetime,clocktime datetime,
comingIn boolean)
        ''')
        db.commit()
    except:
        if DEBUG:
            print('\n [DEBUG] TableClocking already exists. \n ')

def CreateTableClients():
    try:
        cursor.execute('''
        CREATE TABLE clients( clientID int, MacAddress string, IpAddress
string, Description string, LastSeen string)
        ''')
        db.commit()
    except:
        if DEBUG:
            print('\n [DEBUG] TableClients already exists. \n ')

def menu():
    importcsvdata()
    option = "x"
    while option != "0":
        print("Initialise Database")
        print("1. CreateTableEmployee ")
        print("2. CreateTableHours ")
```

```
        print("3. CreateTableClocking ")
        print("4. CreateTableClients ")
        print("4.5. Import newest csv data")
        print("5. Turn Debug mode off")
        print("0. Quit")
        option = input(":: ")
        if option == "1":
            init_suppliers()
        elif option == "2":
            create_supplier(check_number_of_suppliers())
        elif option == "3":
            debug = False
        elif option == "4":
            EmployeeHours()
        elif option == "4.5":
            importcsvdata()
        elif option == "0":
            db.commit()
        print()
        print()



def AddDataClocking(RFIDcode, clientID):
    '''server-code: having received RFID code from a client
        if RFID code does not exist send back error code to client
        otherwise add data to table
    '''
    sqlstring = f"SELECT * FROM employee WHERE RFIDcode = '{RFIDcode}'"
    #print(sqlstring)
    cursor.execute(sqlstring)
    user = cursor.fetchone()
    #print(user)
    if user[0] == None:
        if DEBUG:
            print('RFID code not in database')
    else:   #now add data
        TimeNow = datetime.now()
        current_time = TimeNow.strftime("%H:%M:%S")
        currentClockingID = 1
        clockingIDNOW = currentClockingID =+ 1
        today = date.today()
        currenttime = f'{today.day}/{today.month}/{today.year}'
        sqlstring = f"INSERT INTO clocking (RFIDcode, clientID, clockdate,
clocktime, comingin) VALUES ({RFIDcode}, '{clientID}', '{today}',
'{current_time}', 1)"
        #print(sqlstring)
```

```
        cursor.execute(sqlstring)
        print("test print")
        db.commit()


def importcsvdata():
    csvfile = open("data.csv", "r")
    csvfile.readline()
    data = csvfile.readlines()
    for lineofdata in data:
        lineofdata = lineofdata.split(",")
        AddDataClocking(lineofdata[0], lineofdata[1])
    csvfile.close()
    csvfile = open("data.csv","w")
    csvfile.write("RFID Code" + "," + "Client ID" + "," + "\n")
    csvfile.close()

def TestQuery():
    c = db.cursor()
    c.execute('SELECT * FROM {tn} WHERE employeeID = {cn} '.\
              format(tn='employee', cn=1))
    all_rows = c.fetchall()
    print('1):', all_rows)


def findRFID(RFIDcode):
    c = db.cursor()
    c.execute('SELECT * FROM {tn} WHERE employeeID = {cn} '.\
              format(tn='employee', cn=RFIDcode))
    all_rows = c.fetchall()
    #print(len(all_rows))
    #print(all_rows)
    if len(all_rows) == 0:
        return ('UNKNOWN')
    else:
        return (all_rows[0][2],all_rows[0][3])


def inorout(employeeID):
    datetoday = str(date.today())
    c = db.cursor()
    c.execute('SELECT * FROM {tn} WHERE RFIDcode = {cn} AND clockdate =
"{today}" order by clocktime'.\
              format(tn='clocking', cn=employeeID, today = datetoday))
    all_rows = c.fetchall()
    print(datetoday)
    print(len(all_rows))
    print(all_rows)
```

```
    if len(all_rows) == 0:
        return "OUT"
    else:
        return "OUT" if all_rows[-1][5]==0 else "IN"

def EmployeeHours():
    c = db.cursor()
    print("Please enter the surname of the employee")
    enteredname = input(":: ")
    #c.execute("SELECT employee.surname, employee.employeeID FROM  employee
WHERE employee.surname=?", (enteredname,))
    c.execute("SELECT surname, firstname, employeeID FROM employee WHERE
surname=?", (enteredname,))
    allrows = c.fetchall()
    if len(allrows) == 0:
        print("Employee not found")
        allrows = []
    elif len(allrows) == 1:
        print("Only one employee of this name found, printing data for them
now")
        c.execute("SELECT hours.totalhoursforweek, employee.surname,
employee.firstname, employee.employeeID FROM hours INNER JOIN employee ON
employee.employeeID = hours.employeeID WHERE employee.surname=?",
(enteredname,))
        allrows = c.fetchall()
        print(allrows)
    else:
        print("There is more than one person with this surname, please type in
their ID")
        for row in allrows:
            print("Name", row[1], row[0], "EmployeeID", row[2])
        enteredID = input(":: ")
        c.execute("SELECT hours.totalhoursforweek, employee.surname,
employee.firstname, employee.employeeID FROM hours INNER JOIN employee ON
employee.employeeID = hours.employeeID WHERE employee.employeeID=?",
(enteredID,))
        allrows = c.fetchall()
        totalhours = 0
        for row in allrows:
            totalhours += row[0]
        if len(allrows) != 0:
            print("In their lifetime at the company", allrows[0][2],
allrows[0][1], "has worked", totalhours, "hours")



def clockinorout(employeeID, inorout, clientID):
    datetoday = str(date.today())
```

```
    TimeNow = datetime.now()
    current_time = TimeNow.strftime("%H:%M:%S")
    c = db.cursor()
    c.execute('INSERT INTO clocking (RFIDcode, clientID, clockdate, clocktime,
comingin)VALUES ({RFIDcode},"{cID}", "{today}", "{ct}", {io})'.\
              format(RFIDcode=employeeID, today = datetoday, ct = current_time,
io=inorout,cID=clientID))
    db.commit()


def hoursperday(employeeID, thedate):
    c = db.cursor()
    c.execute('SELECT * FROM clocking WHERE RFIDcode = {eID} AND clockdate =
"{td}" order by clocktime'.\
            format(eID = employeeID, td = thedate))



    all_rows = c.fetchall()
    print(thedate)
    print(len(all_rows))
    print(all_rows)

    FMT = '%H:%M:%S'
    total = datetime.strptime("00:00:00",FMT)

    if len(all_rows)%2 != 0:
        print('HELP')


    for loop in range(0,len(all_rows)-1,2):
        print(all_rows[loop][4],all_rows[loop+1][4])

        difference = datetime.strptime(all_rows[loop+1][4], FMT)  -
datetime.strptime(all_rows[loop][4], FMT)
        total += difference
        print('diff: ', difference)
    print('total: ',total)
#def AddData
```

PROGRAM NAME: admin.py

```
import sqlite3
import os
from datetime import datetime, date
import csv
import uuid
import socket

db = sqlite3.connect('mydb.db')
cursor = db.cursor()
currentversion = str('1.01.274')
theTime = datetime.now()

def get_Host_name_IP():
    try:
        host_name = socket.gethostname()
        host_ip = socket.gethostbyname(host_name)
        print("The Hostname is:  ",host_name)
        print("The IP address is: ",host_ip)
    except:
        print("Unable to get Hostname and IP")

# Driver code
    get_Host_name_IP() #Function call

# joins elements of getnode() after each 2 digits.

    print ("The MAC address is: ", end="")
    print (':'.join(['{:02x}'.format((uuid.getnode() >> ele) & 0xff)
    for ele in range(0,8*6,8)][::-1]))

def mainMenu():
    host_name = socket.gethostname()
    host_ip = socket.gethostbyname(host_name)

    print('\n')
    print('============================')
    print('Version ' + currentversion + ' BETA edition')
    print('============================')
    print('\n')
    print('************************************************')
    print('    Administrator Program: Authorised use only')
    print('************************************************')
    print('\n')
```

```
    print('IP   = ' + str(host_ip))
    print('Time = ' + str(theTime))
    print('\n')
    print('++++++++++++++++++++++++++++')
    print('1: Database Menu')
    print('2: User Access Controls')
    print('3: Client Hardware Controls')
    print('4: About')
    print('Q: Quit')
    print('++++++++++++++++++++++++++++')
    print('\n')

def dbMenu():
    doQuit = False
    while not doQuit:
        print('\n')
        print('**************')
        print('Database Menu')
        print('**************')
        print('\n')
        print('+++++++++++++++++++++++++++++++++++++++++++++')
        print('1: Who is currently on site: ')
        print('2: Hours of a specific employee per month: ')
        print('3: Total hours of a specific employee: ')
        print('4: Where do employees clock in / out most: ')
        print('5: Live SQL mode')
        print('M: go back to main menu')
        print('+++++++++++++++++++++++++++++++++++++++++++++')
        print('\n')
        print('\n')
        choice = input(': ').upper()
        if choice == '1':
            dbMenu1()
        elif choice == '2':
            dbMenu2()
        elif choice == '3':
            dbMenu3()
        elif choice == '4':
            dbMenu4()
        elif choice == '5':
            dbMenu5()
        elif choice == 'M':
            doQuit = True
        else:
            print('The character you inputted might not be fully supported yet:
Please try again')

def dbMenu1():
```

```
    print('\n \n')
    print('\n \n')
    #who is currently in the building or on site
    c = db.cursor()
    today = date.today()
    print(today)
    theTime = datetime.now()
    current_time = theTime.strftime("%H:%M")
    sqlQuery = "SELECT employee.rfidcode, employee.surname, employee.firstname,
clocking.comingIn, clocking.clocktime FROM Clocking INNER JOIN employee ON
employee.rfidcode = clocking.rfidcode WHERE  clockdate=? order by clocktime
asc"
    c.execute( sqlQuery , (today,))
    allrows = c.fetchall()
    if len(allrows) == 0:
        print("There is no one clocked in currently at the time of " +
str(theTime) )
        allrows = []
    else:
        #print(allrows)
        employee = []
        for row in allrows:
            if row[0] not in employee:
                employee.append(row[0])
                if row[3] ==1:
                    print(row[0], row[1], row[2], "is currently on site at the
time of " + str(theTime) + " NOW")

    with open('ONSITE_EXPORT.csv', 'w', newline='') as f_handle:
        writer = csv.writer(f_handle)
        # Add the header/column names
        header = ['RFID Number', 'First Name', 'Second Name', 'Clocking Status
(1 = IN)', 'Clocking in time']
        writer.writerow(header)
        # Iterate over `data`  and  write to the csv file
        for row in allrows:
            writer.writerow(row)

def dbMenu2():
    print('\n \n')
    print('\n \n')
    print('This function is not available yet:' )
    print('Stay tuned for version 1.02 (This version is version number ' +
currentversion + ')')
    pass

def dbMenu3():
    print('\n \n')
```

```
    print('\n \n')
    c = db.cursor()
    print("Please enter the surname of the employee")
    enteredname = input(":: ")
    #c.execute("SELECT employee.surname, employee.employeeID FROM  employee
WHERE employee.surname=?", (enteredname,))
    c.execute("SELECT surname, firstname, employeeID FROM employee WHERE
surname=?", (enteredname,))
    allrows = c.fetchall()
    if len(allrows) == 0:
        print("Employee not found")
        allrows = []
    elif len(allrows) == 1:
        print("Only one employee of this name found, printing data for them
now")
        c.execute("SELECT hours.totalhoursforweek, employee.surname,
employee.firstname, employee.employeeID FROM hours INNER JOIN employee ON
employee.employeeID = hours.employeeID WHERE employee.surname=?",
(enteredname,))
        allrows = c.fetchall()
        print(allrows)
    else:
        print("There is more than one person with this surname, please type in
their ID")
        for row in allrows:
            print("Name", row[1], row[0], "EmployeeID", row[2])
        enteredID = input(":: ")
        c.execute("SELECT hours.totalhoursforweek, employee.surname,
employee.firstname, employee.employeeID FROM hours INNER JOIN employee ON
employee.employeeID = hours.employeeID WHERE employee.employeeID=?",
(enteredID,))
        allrows = c.fetchall()
        totalhours = 0
        for row in allrows:
            totalhours += row[0]
        if len(allrows) != 0:
            print("In their lifetime at the company", allrows[0][2],
allrows[0][1], "has worked", totalhours, "hours")

    with open('TOTALHOURS_EXPORT.csv', 'w', newline='') as f_handle:
        writer = csv.writer(f_handle)
        # Add the header/column names
        header = ['Total Hours', 'Second Name', 'First Name', 'Employee ID']
        writer.writerow(header)
        # Iterate over `data`  and  write to the csv file
        for row in allrows:
            writer.writerow(row)
```

```
def dbMenu4():
    print('\n \n')
    print('\n \n')
    c = db.cursor()
    #What client station do employees use most

    sqlquery = ("SELECT COUNT(clientID) ,clientID FROM clocking group BY
clientID")
    c.execute(sqlquery)
    allrows = c.fetchall()

    for row in allrows:
        print("Number of uses: " + str(row[0]) + "  " + "IP Address: " +
str(row[1]))

    with open('CLIENT_MACHINE_EXPORT.csv', 'w', newline='') as f_handle:
        writer = csv.writer(f_handle)
        # Add the header/column names
        header = ['Number of uses', 'IP Address Number']
        writer.writerow(header)
        # Iterate over `data`  and  write to the csv file
        for row in allrows:
            writer.writerow(row)

def dbMenu5():
    c = db.cursor()
    adminsql = input("Please enter SQL commands to execute:  ")
    c.execute(adminsql)
    allrows = c.fetchall()
    print(allrows)

def clientMenu():
    print('\n \n')
    print('\n \n')
    print('Client Menu \n \n')
    print('This is coming in a future update:')
    print('Ability to restart, authorise and stop client machines')
    print('Stay tuned for version 1.02 (This version is version number ' +
currentversion + ')')

def UserAccessMenu():
    print('\n \n')
    print('\n \n')
    print('User Access Menu \n \n')
    print('This is coming in a future update:')
    print('Ability to add, remove, edit users')
    print('Ability to add and remove RFID cards')
```

```
    print('Stay tuned for version 1.02 (This version is version number ' +
currentversion + ')')

def About():
    print('\n \n')
    print('This version of the program is BETA ' + currentversion)
    print('Github link to the project and all of the code:
https://github.com/williamd002/A-level-NEA')
    print("Copyright William Puchy 2020")
    print('\n \n')
    print(",--.   ,--.,--.,--.,--.,--.                    ,------.
,--.               ")
    print("|  |   |  |`--'|  ||  |`--' ,--,--.,--,--,--.    |  .--. ',--.,--.
,---.|  ,---. ,--. ,--. ")
    print("|  |.'.|  |,--.|  ||  |,--.' ,-.  ||        |    |  '--' ||  ||  ||
.--'|  .-.  | \  '  /  ")
    print("|   ,'.   ||  ||  ||  ||  |\ '-'  ||  |  |  |    |  | --' '  ''  '\
`--.|  | |  |  \   '   ")
    print("'--'   '--'`--'`--'`--'`--' `--`--'`--`--`--'    `--'      `----'
`---'`--' `--'.-'  /    ")
    print("
`---'     ")

print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++")

print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++")

print("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++")
    print('\n \n')
    print('\n \n')

#main
doQuit = False
while not doQuit:
    mainMenu()
    choice = input(': ').upper()
    if choice == '1':
        dbMenu()
    elif choice == '2':
        UserAccessMenu()
    elif choice == '3':
        clientMenu()
    elif choice == '4':
        About()
    elif choice == 'Q':
```

```
        doQuit = True
    else:
        print('Sorry, this character is not currently implemented in the
program, Try again...')
print('Closing Program')
print('It is now safe to close this window')
```

PROGRAM NAME: client_gui.py

```
import PySimpleGUI as sg
import sys
import random
import socket
import selectors
import types
import datetime
import uuid
import threading

SERVER = '127.0.0.1'    #current address of server
PORT = 1234             #current port used by server

#find client MAC address
myMac = (':'.join(['{:02x}'.format((uuid.getnode() >> ele) & 0xff) for ele in
range(0,8*6,8)][::-1]))

client_states=['INIT','WAIT','READY','READING','CLOCKING']
current_state = 0

sel = selectors.DefaultSelector()

rfid_read = False
rfid_value = ''
#for testing purposes before integration of RFID reader
test_rfids = [987612,147829,196301,749172]

def fakeRfid(): #to be replaced with RFID reader code
    '''code to setup a test rfid value before RFID reader integrated'''
    global rfid_read, rfid_value
    print('fake RFID')
```

```
    rfid_read = True
    rfid_value = random.choice(test_rfids)


def rfid_thread():
    '''thread to run the processing of RFID values
       when a value is ready to be process rfid_read will be True
    '''
    global current_state, rfid_read, rfid_value
    print('start of RFID thread')
    server_addr = (SERVER, PORT)
    print('starting RFID connection to',server_addr)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setblocking(False)
    sock.connect_ex(server_addr)
    while current_state == 3:
        if rfid_read:
            sel = selectors.DefaultSelector()
            print('RFID READING')
            rfid_read = False
            print(rfid_value)
            message = "RFID$"+str(rfid_value)
            events = selectors.EVENT_READ | selectors.EVENT_WRITE

            data = types.SimpleNamespace(
                connid = 1,
                msg_total = len(message),
                recv_total=0,
                messages=[message.encode()],
                outb=b"",
                )
            sel.register(sock, events, data=data)


            current_state = 4 #clocking
            try:
                done = False
                while (not done) and current_state == 4:
                    events = sel.select(timeout=1)
                    if events:
                        for key,mask in events:
                            service_connection(key,mask)

            except:
                print('Caught keyboard - exiting')

            sel.close()
    print('end of RFID thread')
```

```
def init_thread():
    '''thread to make initial contact with the server'''
    global current_state
    print('start of init thread')
    init_connection()
    try:
        done = False
        while (not done) and current_state != 2:
            events = sel.select(timeout=1)
            if events:
                for key,mask in events:
                    service_connection(key,mask)

    except:
        print('Caught keyboard - exitin')
    finally:
        pass
        sel.close()
    print('end of init thread')


def init_connection():
    #init connection to server and await response
    print('start of init connection')
    server_addr = (SERVER, PORT)
    message = f'INIT${myMac}'
    print('starting INIT connection to',server_addr)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.setblocking(False)
    sock.connect_ex(server_addr)
    events = selectors.EVENT_READ | selectors.EVENT_WRITE
    data = types.SimpleNamespace(
        connid = 1,
        msg_total = len(message),
        recv_total=0,
        messages=[message.encode()],
        outb=b"",
        )
    sel.register(sock, events, data=data)
    print('end of init connection')

#code ideas taken from https://realpython.com/python-sockets/
def service_connection(key, mask):
```

```
    global current_state
    #print('start of service connection')
    sock = key.fileobj
    data = key.data
    if mask & selectors.EVENT_READ:
        recv_data = sock.recv(1024)  # Should be ready to read
        if recv_data:
            print("received", repr(recv_data), "from connection", data.connid)
            data.recv_total += len(recv_data)

            recvData = recv_data.decode().split("$")
            recvCommand = recvData[0]

            print(recvData)
            print(recvCommand)
            if recvCommand == "OK":
                current_state = 2
                print(current_state)
            if recvCommand == "ERROR":
                current_state = 3
                window['_OUTPUT_'].update(recvData[1])
            if recvCommand == "CLOCKED":
                current_state = 3
                window['_OUTPUT_'].update(recvData[1])


        if not recv_data or data.recv_total == data.msg_total:
            print("closing connection", data.connid)
            sel.unregister(sock)
            sock.close()
    if mask & selectors.EVENT_WRITE:
        if not data.outb and data.messages:
            data.outb = data.messages.pop(0)
        if data.outb:
            print("sending", repr(data.outb), "to connection", data.connid)
            sent = sock.send(data.outb)  # Should be ready to write
            data.outb = data.outb[sent:]
    #print('end of service connection')

thread1 = threading.Thread(target=init_thread)
thread1.start()

today = datetime.date.today()
theTime = datetime.datetime.now()
current_time = theTime.strftime("%H:%M")
sg.theme('Dark')

host_ip="."  #initially we display dots as we initiate
```

```
#Setup layout
layout = [
        [sg.Text(today, font=("Helvetica", 15), size=(9,
1)),sg.Text(current_time, font=("Helvetica", 15), size=(9,
1),key='_TIME_'),sg.Text(host_ip, font=("Helvetica", 15), size=(17,
1),key="_IP_")],
          [sg.Text('INITIATING', size=(25, 2), font=('Helvetica', 20),
justification='left', key='_OUTPUT_')],
        [sg.Button('Read'), sg.Exit()]]

# Show the Window to the user
window = sg.Window('Clock In Client', layout, size=(320,240) )

#Main loop
while True:
    # Read the Window
    time = datetime.datetime.now()
    current_time = time.strftime("%H:%M:%S")
    if current_state <2:            #if initialising draw more dots
        host_ip = host_ip + "."
        if len(host_ip) == 5:
            host_ip="."
    else:
        host_ip = myMac             #otherwise show MAC address

    window['_IP_'].update(host_ip)            #update IP area of window
    window['_TIME_'].update(current_time)     #update TIME area of window
    event, value = window.read(timeout=100)   #read any event from the GUI
    if event in ('Quit', None):               #if window is closed then exit
        break
    if event in (None, 'Exit'):               #if EXIT button clicked then exit
        break
    if event in ('Read', None):               #if 'fake' read button clicked
        fakeRfid()
    if current_state == 2:                    #if we are now in READY state
move to READING state
        current_state = 3
        window['_OUTPUT_'].update('READY')
        thread1 = threading.Thread(target=rfid_thread) #start the RFID
processing thread
        thread1.start()

window.close()
```

PROGRAM NAME: server_code.py

```
#!/usr/bin/env python3

import sys
import socket
import selectors
import types
import sqlite3
import os
from datetime import datetime, date

sel = selectors.DefaultSelector()
db = sqlite3.connect('mydb.db')
cursor = db.cursor()

def findRFID(RFIDcode):
    '''a function to search for an RFIDcode in the employee table
       if employee exists return the firstnamr and surname
       if employee does not exists return 'UNKNOWN'
    '''
    print('searching for',RFIDcode)
    db = sqlite3.connect('mydb.db')
    cursor = db.cursor()
    c = db.cursor()
    c.execute('SELECT * FROM {tn} WHERE RFIDcode = {cn} '.\
              format(tn='employee', cn=RFIDcode))
    all_rows = c.fetchall()
    db.close()
    if len(all_rows) == 0:
        return ('UNKNOWN')
    else:
        return (all_rows[0][2],all_rows[0][3])


def inorout(rfidCode):
    '''a function that will take an rfidCode and
       return 'OUT' if the employee is clocked out (or not in today)
       reuturn 'IN' if the employee is currently clocked in
    '''
    datetoday = str(date.today())
    c = db.cursor()
    c.execute('SELECT * FROM {tn} WHERE RFIDcode = {cn} AND clockdate =
"{today}" order by clocktime'.\
              format(tn='clocking', cn=rfidCode, today = datetoday))
```

```
    all_rows = c.fetchall()
    if len(all_rows) == 0:
        return "OUT"
    else:
        return "OUT" if all_rows[-1][5]==0 else "IN"

def clockinorout(employeeID, inorout, clientID):
    datetoday = str(date.today())
    TimeNow = datetime.now()
    current_time = TimeNow.strftime("%H:%M:%S")
    c = db.cursor()
    c.execute('INSERT INTO clocking (RFIDcode, clientID, clockdate, clocktime,
comingin)VALUES ({RFIDcode},"{cID}", "{today}", "{ct}", {io})'.\
              format(RFIDcode=employeeID, today = datetoday, ct = current_time,
io=inorout,cID=clientID))
    db.commit()

#code ideas taken from https://realpython.com/python-sockets/
def accept_wrapper(sock):
  conn, addr = sock.accept() # Should be ready to read
  print("accepted connection from", addr)
  conn.setblocking(False)
  data = types.SimpleNamespace(addr=addr, inb=b"", outb=b"")
  events = selectors.EVENT_READ | selectors.EVENT_WRITE
  sel.register(conn, events, data=data)

def service_connection(key, mask):
  sock = key.fileobj
  data = key.data
  if mask & selectors.EVENT_READ:
    recv_data = sock.recv(1024) # Should be ready to read
    if recv_data:
      data.outb += recv_data
    else:
      print("closing connection to", data.addr)
      sel.unregister(sock)
      sock.close()
  if mask & selectors.EVENT_WRITE:
    if data.outb:
      commCode = str(data.outb)
      print(commCode)
      commCode = commCode.split('$')
      commCommand = commCode[0][2:]
      print('Command:',commCommand)
      commData = commCode[1][:-1]
      print(commData)

      #make choice depending on what the commCode is
```

```
      #INIT - need to check if client is authorised
      #RFID - need to process an RFID code

      if commCommand == "INIT":
          print('Processing INIT command')
          #for now just return OK
          #really need to search to see if client is authorised
          data.outb = "OK$ ".encode()
      elif commCommand == "RFID":
          print('Processing RFID command')
          person = findRFID(commData)
          if person == "UNKNOWN":
            print('Unknown person')
            data.outb = f"ERROR$Unknown fob - please see HR".encode()
          else:
            inout = inorout(commData)
            if inout == 'OUT':
                message = 'CLOCKED IN'
            else:
                message = 'CLOCKED OUT'
            inout = 1 if inout=='OUT' else 0
            data.outb = f"CLOCKED$Welcome {person[0]}
{person[1]}\n{message}".encode()
            clockinorout(commData,inout,'192.168.1.3')
            print("echoing", repr(data.outb), "to", data.addr)
      sent = sock.send(data.outb) # Should be ready to write
      data.outb = data.outb[sent:]



if len(sys.argv) != 3:
  print("usage:", sys.argv[0], "<host> <port>")
  sys.exit(1)

host, port = sys.argv[1], int(sys.argv[2])
lsock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
lsock.bind((host, port))
lsock.listen()
print("listening on", (host, port))
lsock.setblocking(False)
sel.register(lsock, selectors.EVENT_READ, data=None)

try:
  while True:
    events = sel.select(timeout=None)
    for key, mask in events:
      if key.data is None:
        accept_wrapper(key.fileobj)
```

```
      else:
        service_connection(key, mask)
except KeyboardInterrupt:
  print("caught keyboard interrupt, exiting")
finally:
  sel.close()
```