



## Raport

### System ewidencji czasu pracy wykorzystujący technologie Internetu Rzeczy

Przedmiot:

**Podstawy Internetu Rzeczy  
Laboratorium**

Imię i nazwisko autora:

**Wojciech Kozłowski**

Nr indeksu:

**246992**

Semestr studiów:

**4**

Data ukończenia pracy:

**Maj 2020 r.**

Prowadzący laboratorium:

**mgr Kamil Nowak**



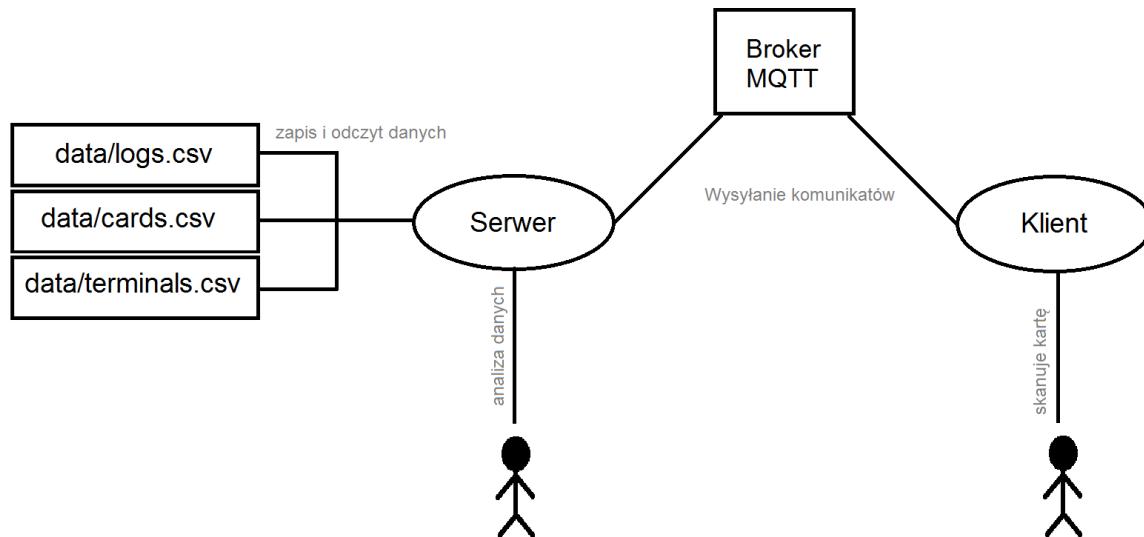
# Spis treści

<b>3 Wymagania projektowe</b>	<b>2</b>
<b>4 Opis architektury systemu</b>	<b>3</b>
<b>5 Opis implementacji i zastosowanych rozwiązań</b>	<b>3</b>
5.1 Serwer . . . . .	3
5.1.1 main.py . . . . .	3
5.1.2 server.py . . . . .	4
5.1.3 listener.py . . . . .	4
5.1.4 menu.py . . . . .	5
5.2 Klient . . . . .	5
5.2.1 client.py . . . . .	5
<b>6 Opis działania i prezentacja interfejsu</b>	<b>5</b>
6.1 Instalacja potrzebnych paczek . . . . .	5
6.2 Konfiguracja brokera mosquitto . . . . .	5
6.2.1 Tworzenie kluczy RSA . . . . .	5
6.2.2 Konfiguracja uwierzytelniania . . . . .	7
6.2.3 Konfiguracja autoryzacji . . . . .	8
6.3 Uruchomienie głównego programu . . . . .	8
<b>7 Podsumowanie</b>	<b>9</b>
<b>8 Literatura</b>	<b>9</b>
<b>9 Aneks</b>	<b>9</b>

## 3 Wymagania projektowe

- Program główny i programy symulujące czytniki RFID komunikują się za pomocą brokera mqtt.
- Serwer nasłuchiwa i gromadzi dane w plikach csv w podfolderze data.
- Serwer zapisuje wszystkie znane terminale (czytniki, które wysłały wiadomość początkową), wraz z ich miejscem (in/out).
- Serwer ma możliwość dodania nowej karty RFID i skojarzenia go z imieniem i nazwiskiem pracownika.
- Serwer każdą dostarczoną informację zapisuje w pliku logs.csv, kojarząc identyfikatory kart z ich właścicielami, a identyfikatory terminali z ich miejscem.
- Serwer ma możliwość zmiany właściciela karty RFID.
- Serwer rejestruje nieznane karty RFID zostawiając pole imienia i nazwiska puste.
- Wiadomości są szyfrowane za pomocą TLS.
- Komunikacja w systemie podlega autoryzacji.

## 4 Opis architektury systemu



System składa się z serwera i dowolnej ilości klientów. Serwer jest zaimplementowany w plikach:

- `main.py`
- `server.py`
- `listener.py`

Korzysta także z plików `utils.py` oraz `menu.py`.

Klient to tylko jeden plik nazwany `client.py`.

## 5 Opis implementacji i zastosowanych rozwiązań

### 5.1 Serwer

#### 5.1.1 `main.py`

Plik `main.py` jest plikiem rozruchowym całego serwera. Znajdują się w nim także referencje do obiektów klas Server, Listener oraz Menu. Odpowiada on za obsługę wszystkich opcji w menu.

- Dodanie karty RFID
- Zmiana właściciela karty RFID
- Wylistowanie wszystkich znanych terminali
- Wylistowanie wszystkich kart RFID
- Wypisanie logów
- Wyjście

We wszystkich opcjach (za wyjątkiem ostatniej) obsługa polega tylko na odwołaniu się do pewnego pola, bądź wywołaniu odpowiedniej metody na obiekcie Server. Pętla odpowiedzialna za obsługę wyborów w menu.

### 5.1.2 server.py

W tym pliku znajduje się klasa o nazwie Server. Odpowiada ona za odczyt i zapis do plików wszystkich nadsyłanych informacji. W celu przyśpieszenia działania aplikacji dane na temat kart i terminali oprócz tego, że są na bieżąco zapisywane do plików, są także przechowywane w polach serwera. Co za tym idzie wylistowania danych oraz kojarzenia na przykład numeru karty RFID z imieniem i nazwiskiem właściciela nie muszą otwierać i czytać całych plików.

W konstruktorze zapisywane są ścieżki do plików z danymi, tworzone pola przechowujące dane kart i terminali oraz jeśli tworzony wraz z nagłówkiem plik `logs.csv` jeśli takowy nie istnieje. Metoda, która wykonuje się, kiedy ktośczyta kartę, nazywa się `get_update`.

```
def gate_update(self, ID, RFIDnum):
    def find_place(): return list(filter(lambda pair: pair[0] == ID,
                                         self.terminals))[0][1]

    if ID not in [term[0] for term in self.terminals]:
        return None

    name, surname = "", ""
    if RFIDnum in [num for num, _, _ in self.cards]:
        name, surname = self.find_person(RFIDnum)

    with open(self.logs_file, 'a') as file:
        date = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
        line = [date, RFIDnum, name, surname, ID, find_place()]
        file.write(','.join(line) + '\n')
```

Jak widać zdarzenie jest zapisywane, kiedy karta RFID jest nieznana, natomiast, kiedy terminal jest nieznany - zdarzenie jest ignorowane.

### 5.1.3 listener.py

Plik, który odpowiada za przechwytywanie wiadomości mqtt. Znajduje się w nim klasa o takim samym imieniu i przechowuje ona dane na temat nazwy broker'a, portu oraz kanału. Jej metody to

- `connect_to_broker` - połączenie się do brokerka
  - `process_message` - obsługiwanie wiadomości

- disconnect - odłączenie się od brokera

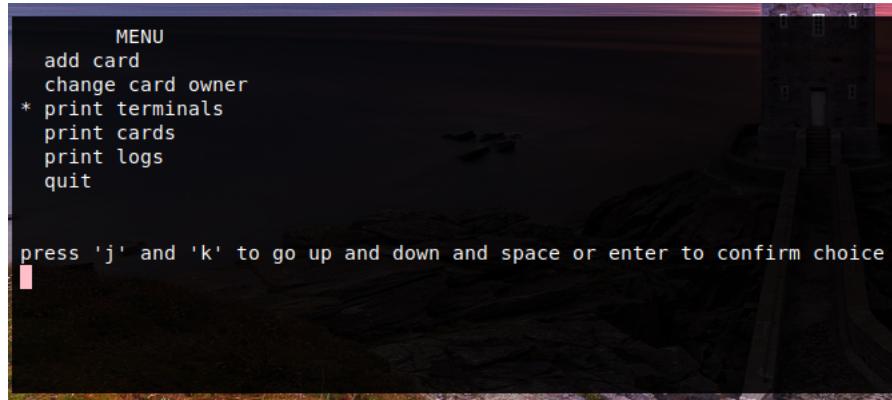
#### 5.1.4 menu.py

Plik odpowiada za wyświetlanie oraz nawigację w menu. Sterowanie jest vimowe tzn

- j - o jedną opcję niżej
- k - o jedną opcję wyżej
- g - pierwsza opcja
- G - ostatnia opcja
- <spacja> lub <enter> - zatwierdzenie

Aktualnie wybrana opcja ma znak \* przed nazwą.

Menu wygląda tak:



## 5.2 Klient

### 5.2.1 client.py

Plik, który jest jedyną składową klienta. Posiada dane na temat brokera mosquitto i wysyła klientowi numer RFID podany z klawiatury. Na początku program prosi o podanie ID, które jest identyfikatorem terminala, który symuluje ten program. Następnie należy podać miejsce położenie terminala. Dostępne są dwie opcje in lub out. Kiedy wpiszemy inny tekst domyślnie zostanie wybrana opcja in. Dalej program w pętli pyta o numery zczytywanych kart RFID, które wysyła do brokera mqtt.

# 6 Opis działania i prezentacja interfejsu

## 6.1 Instalacja potrzebnych paczek

Zakładam, że użytkownik posiada system debian. Cały projekt napisany jest w języku python3, więc dla pewności zacznijmy od tego. W celu zainstalowania pythona, użytkownik musi włączyć terminal i napisać

```
$ sudo apt install python3
```

Następnie przejdźmy do zainstalowania paczek mosquitto i mosquitto-clients.

```
$ sudo apt install mosquitto mosquitto-clients
```

Choć debian powinien mieć domyślnie paczkę openssl, nie zaszkodzi także spróbować jej zainstalować - jeśli była zainstalowana, nic się nie wydarzy.

```
$ sudo apt install openssl
```

Teraz, kiedy mamy już wszystkie potrzebne paczki, możemy przejść do konfiguracji.

## 6.2 Konfiguracja brokera mosquitto

### 6.2.1 Tworzenie kluczy RSA

W zasadzie, komendy potrzebne do poprawnego skonfigurowania mosquitto są takie same jak dla windowsa, z wyjątkiem folderu, w którym będą przechowywane klucze (będzie to /etc/mosquitto/certs/).

Poniżej przedstawię komendy do konfiguracji serwera mosquitto.

```
$ openssl genrsa -des3 -out ca.key 2048
```

Podczas generacji należy wpisać i potwierdzić dowolne hasło np. politechnika. Ta komenda stworzyła nam klucz RSA o długości 2048 bitów.

Kolejnym krokiem jest stworzenie certyfikatu uwierzytelnienia.

```
$ openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
```

Hasło jest takie, jakie ustaliliśmy w poprzednim kroku. W pole County Name wpisujemy PL, a resztę pól zostawiamy pustych przyciskając enter.

Następnie tworzymy parę kluczy używanych przez broker

```
$ openssl genrsa -out server.key 2048
```

Dalej tworzymy żądanie podpisania certyfikatu.

```
$ openssl req -new -out server.csr -key server.key
```

Pole Common Name wypełniamy ponownie wartością PL, a w pole Common Name wpisujemy nazwę naszego komputera (podobnie jak w windowsie, możemy ją sprawdzić wpisując komendę hostname).

Ostatnim krokiem jest wpisanie tej komendy:

```
$ openssl x509 -req -in server.csr -CA ca.crt  
-CAkey ca.key -CAcreateserial -out server.crt -days 360
```

Zostaniemy jeszcze raz poproszeni o podanie hasła, które ustaliliśmy w pierwszym kroku.

Komendą ls możemy sprawdzić, czy stworzyły nam się podane pliki:

- ca.crt
- ca.key
- ca.srl
- server.crt
- server.csr
- server.key

W przypadku, gdy tworzyliśmy je w katalogu domowym i ciężko jest nam sprawdzić, czy dane pliki się stworzyły można przefiltrować wyszukiwania w taki sposób:

```
$ ls | grep "ca.crt"
```

Jeśli plik istnieje, to powinien się pokazać.

Folder, w którym musimy przenieść nasze nowe klucze, powinien być stworzony podczas instalacji pakietu mosquitto, więc wystarczy tylko, że przeniesiemy nowe pliki do tego foldera.

```
$ sudo mv ca.crt server.crt server.key /etc/mosquitto/certs/
```

Jako że plik mosquitto.conf jest długi najłatwiej będzie go pobrać z plików, które załączylem do projektu. Należy go tylko umieścić w odpowiednie miejsce.

```
$ sudo mv mosquitto.conf /etc/mosquitto/
```

Będzie on już wypełniony. Jego poprawność można sprawdzić za pomocą komendy:

```
$ sed 's/#.*//g; /^$/ d' mosquitto.conf
```

Aby włączyć broker mosquitto należy wpisać odpowiednią komendę:

```
$ systemctl start mosquitto
```

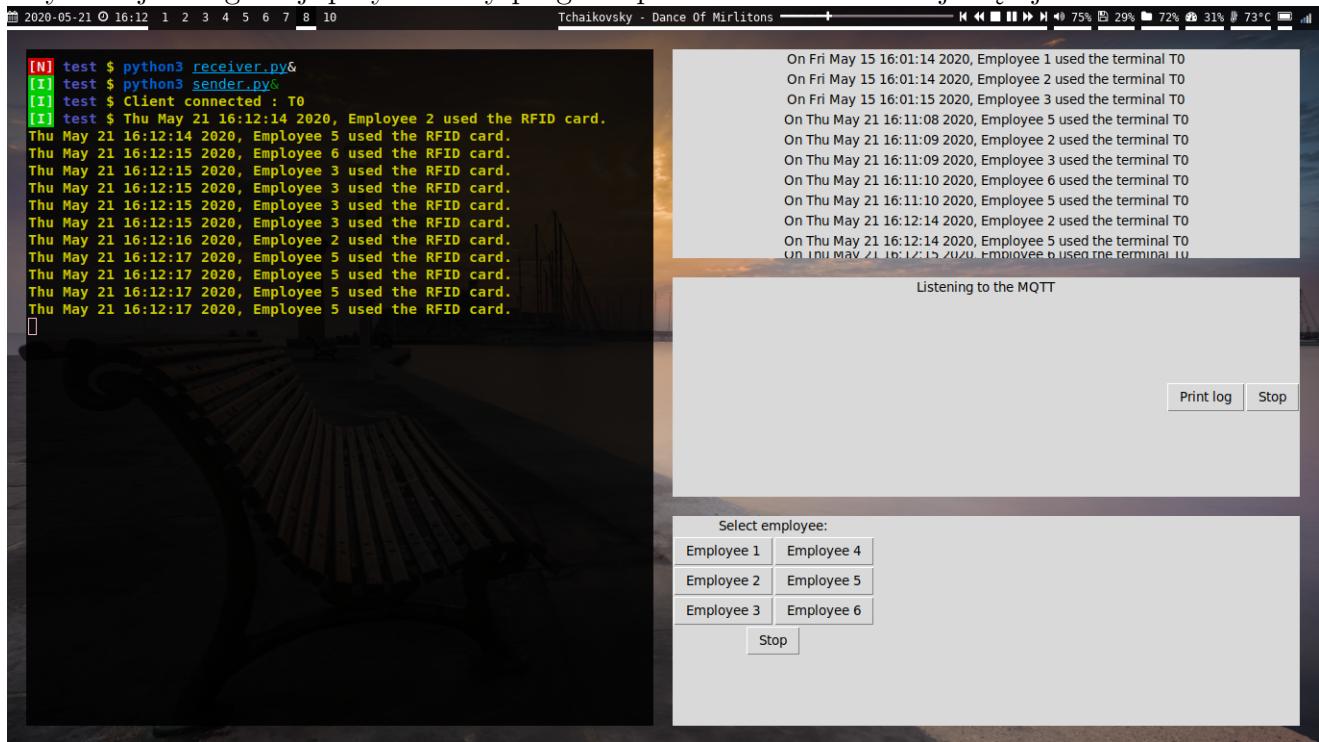
Żeby sprawdzić działanie brokera posłużymy się przykładowym programem, który został opublikowany na eportalu. Należy tylko zmienić wartość brokera na nazwę komputera, port na 8883 i funkcję connect\_to\_broker na taką:

---

```
def connect_to_broker():
    client.tls_set("/etc/mosquitto/certs/ca.crt")
    # Connect to the broker.
    client.connect(broker, port)
    # Send message about connection.
    client.on_message = process_message
    # Starts client and subscribe.
    client.loop_start()
    client.subscribe("worker/name")
```

---

Przy takiej konfiguracji przykładowy program powinien działać mniej więcej tak:



### 6.2.2 Konfiguracja uwierzytelniania

Następnym etapem jest konfiguracja uwierzytelniania. W celu dodania dwóch użytkowników należy wykonać podane komendy:

```
$ sudo mosquitto_passwd -c /etc/mosquitto/passwd server  
$ sudo mosquitto_passwd -b /etc/mosquitto/passwd client password
```

Wpisane hasło do serwera należy zapamiętać. Aby wyświetlić i zarazem sprawdzić wewnętrzne pliku passwd można wpisać w termianlu:

```
$ cat /etc/mosquitto/passwd
```

Wynik tej komendy powinien wyglądać mniej więcej tak:

```
server:$6$kUvn8RbzHzfMFsH$raez7f4bdSnAbNVvpe  
yeRBDvg3XY9JpzXXZB2jZ5xaAmWa9Qy0TKoWkbZFhR2UpoQ0  
KDCxY0GZBoNv7eX2sbQA==  
client:$6$1NHgYSyJNWBTWwUP$sH8XjGQLKNYPyfjDGMpp  
8EefYLm547p1YiqN6e2dJRyBKWZ1LzV3JmeM1uLpIb60DhMfr/  
dQWIXoT6wrpf1MIA==
```

Następnie trzeba odkomentować linijki 651 i 670 w mosquitto.conf oraz zrestartować broker komendą:

```
$ systemctl restart mosquitto
```

Żeby sprawdzić efekt do pliku `sender.py` i `receiver.py` dopiszemy po jednej linijce w funkcji `connect_to_broker`.

---

```
def connect_to_broker():  
    client.tls_set("/etc/mosquitto/certs/ca.crt")  
    # Authorization  
    client.username_pw_set(username='client', password='password')  
    # Connect to the broker.  
    client.connect(broker, port)  
    # Send message about connection.  
    client.on_message = process_message  
    # Starts client and subscribe.  
    client.loop_start()  
    client.subscribe("worker/name")
```

---

Wszystko powinno działać tak samo, jednak, gdy zmienimy wartość hasła na inną, to komunikacja zostanie przerwana.

### 6.2.3 Konfiguracja autoryzacji

Najpierw należy utworzyć plik aclfile w folderze /etc/mosquitto oraz wypełnić go w ten sposób:

```
# This only affects clients with username "server".
user server
topic server/name
topic worker/name

# This only affects clients with username "client".
user client
topic read server/name
topic worker/name
```

By plik był widoczny należy odkomentować linijkę 729 w pliku mosquitto.conf. Przy takiej konfiguracji można uruchomić główny program.

## 6.3 Uruchomienie głównego programu

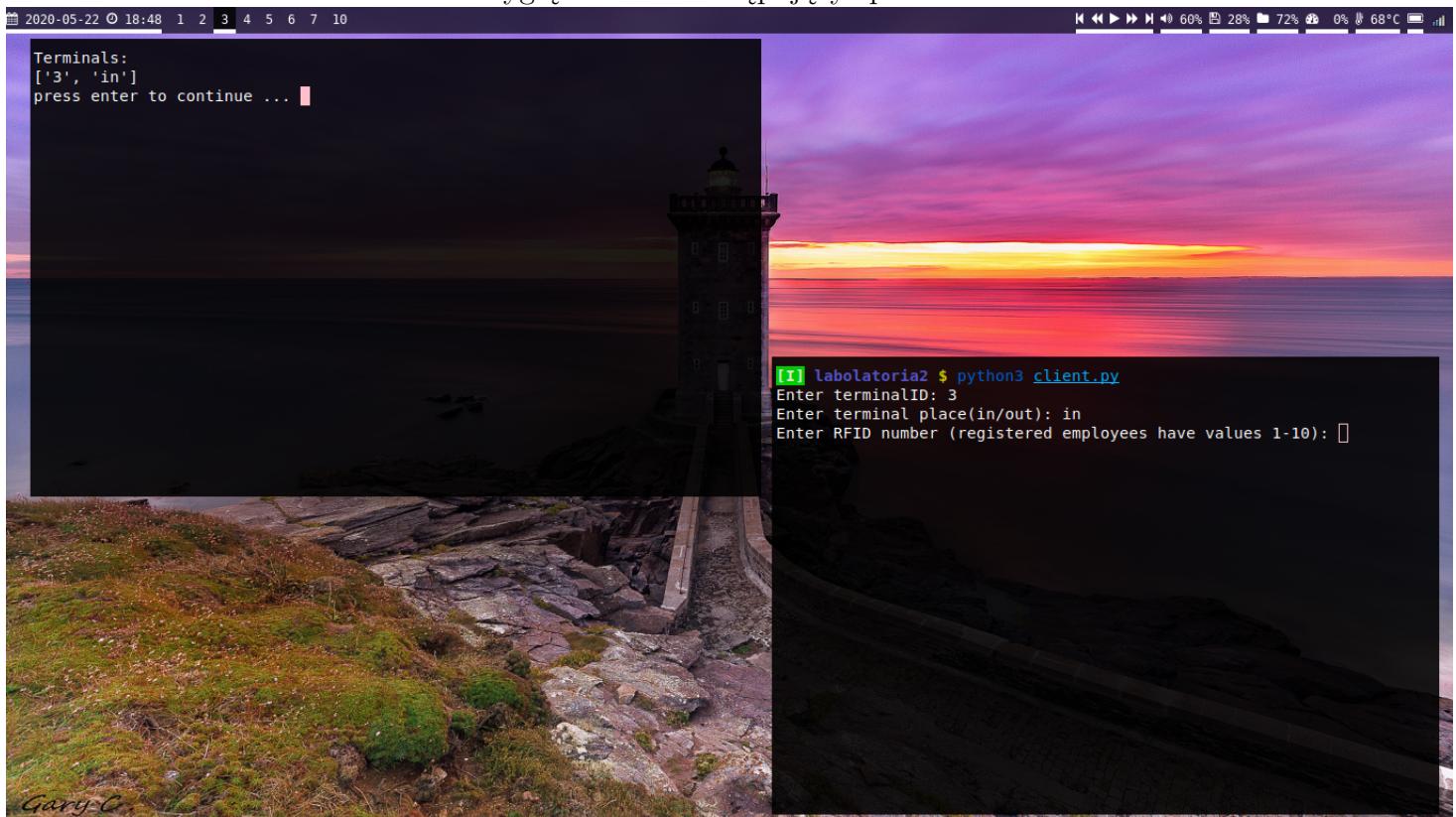
W pierwszej kolejności należy włączyć serwer poleceniem

```
$ python3 main.py
```

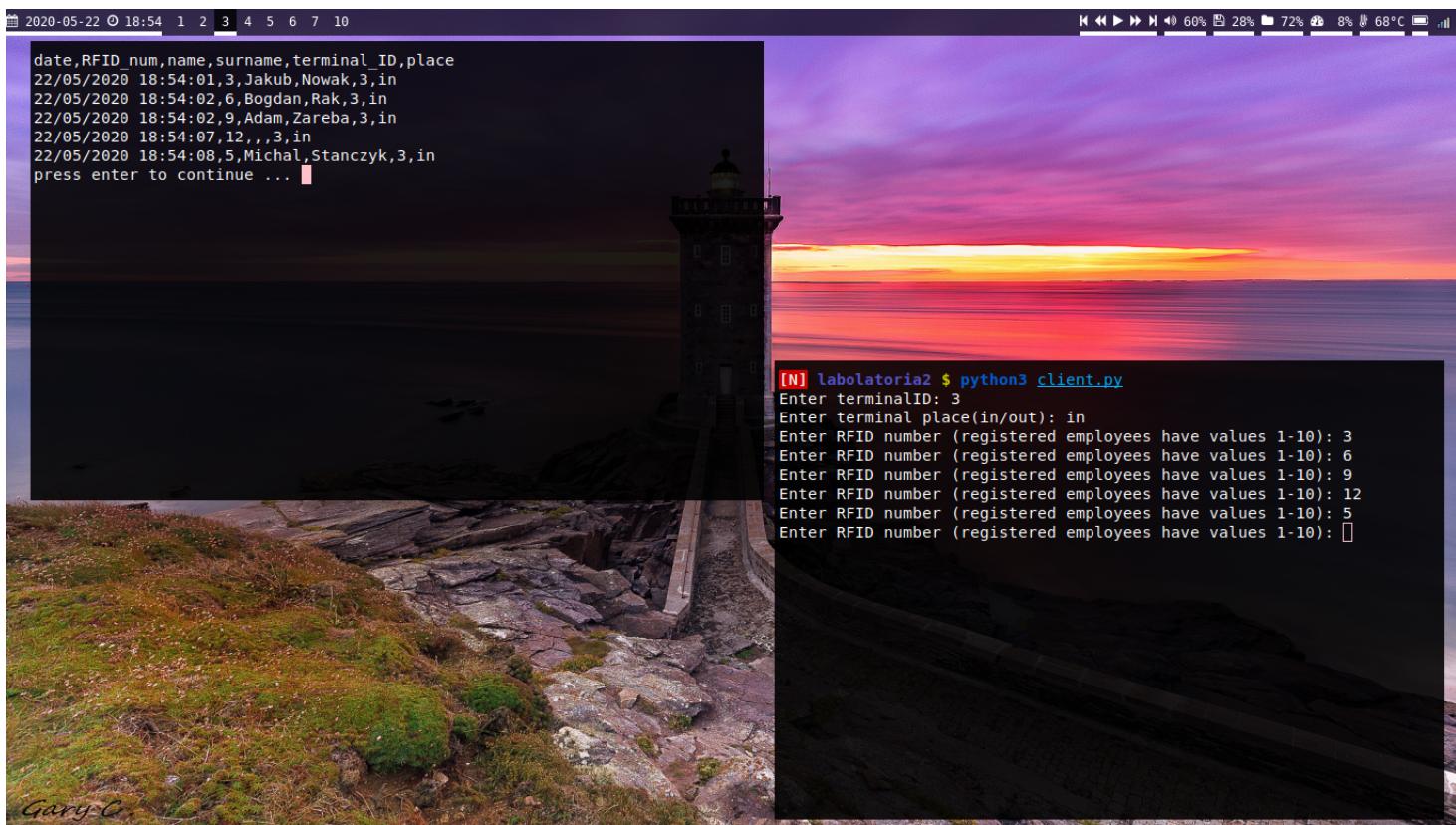
Domyślnie będzie dziesięć zarejestrowanych kart RFID, ale jeśli chcemy możemy stworzyć kilka nowych wybierając pierwszą opcję w menu. Możemy także zmodyfikować właściciela jednej z kart wybierając opcję drugą (należy nacisnąć 'j' a potem zatwierdzić enterem). Zmiany możemy sprawdzić wybierając czwartą. Opcji trzecia, która wylistowuje wszystkie terminale, powinna być póki co pusta. By to zmienić należy w innym terminalu w folderze z kodem źródłowym napisać komendę:

```
$ python3 client.py
```

Tam wpisujemy dowolny tekst (preferowana liczba naturalna ale nie ma takiej konieczności) jako ID terminala. Dalej wpisujemy albo in albo out, by określić położenie nowego terminala. Jeśli wpiszemy coś innego zostanie wybrana opcja in. Następnie program czeka na wysłanie numeru zeskanowanej karty RFID. Zanim wyślemy jakąś możemy sprawdzić, czy w serwerze lista terminali została zaktualizowana. Może wyglądać to w następujący sposób:



Wyślijmy więc jakieś dane do systemu. Część z nich może być z przedziału 1-10, a część nie. Dodatkowo można wysłać id, które wcześniej zarejestrowaliśmy w serwerze, aby sprawdzić, czy nowa karta RFID została poprawnie dodana. Po wysłaniu warto sprawdzić logi, które zostały zapisane w pliku logs.csv. Najwygodniej będzie to zrobić wybierając piątą opcję w menu. Poniżej prezentuję możliwy scenariusz:



Znane karty RFID serwer wypełnił danymi posiadaczy, a w przypadku nieznanych kart pola imię i nazwisko zostawił pustę. W celu wyjścia z klienta najprościej wcisnąć <Ctrl>+C. Aby zakończyć działanie serwera wybieramy opcję ostatnią.

## 7 Podsumowanie

Projekt nauczył nas nie tyle programowania w arduino i raspberry pi co programowania w ogólnosci. Rolę pierwszoplanową odegrał tutaj protokół MQTT. Choć mi wcześniej zdarzyło się korzystać z tego protokołu, to z pewnością moja wiedza uległa ugruntowaniu. Bardzo doceniam też daną nam swobodę wykonywania zadania, aczkolwiek nie mogę być przez to pewny co do poprawności wykonanego projektu.

## 8 Literatura

<https://github.com/mqtt/mqtt.github.io/wiki>

<https://docs.python.org/3/>

## 9 Aneks

<https://github.com/WojciechKoz/mqtt>