



Raport

System ewidencji czasu pracy wykorzystujący technologie Internetu Rzeczy

Przedmiot:	Podstawy Internetu Rzeczy Laboratorium
Imię i nazwisko autora:	Łukasz Gajerski
Nr indeksu:	xxxxxx
Semestr studiów:	4
Data ukończenia pracy:	08.05.2020 r.
Prowadzący laboratorium:	dr inż. Kamil Nowak



2. Spis treści

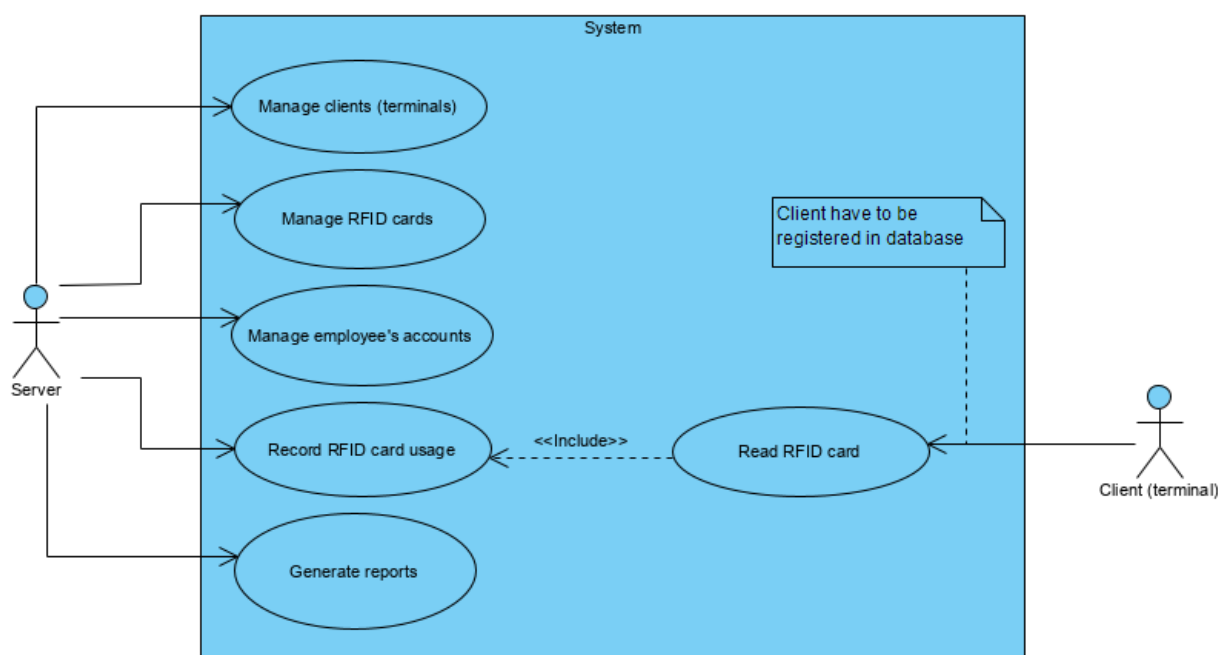
Raport.....	1
3. Wymagania projektowe	3
Wymagania funkcjonalne:.....	3
Wymagania нефunkcjonalne:.....	4
4. Opis architektury systemu	4
Opis architektury systemu:	4
5. Opis implementacji i zastosowanych rozwiązań.....	6
Dokumentacja wszystkich funkcji systemu:	6
Implementacja protokołu MQTT:.....	7
Uwierzytelnienie i autoryzacja:	9
6. Opis działania i prezentacja interfejsu	10
Instrukcja użycia:	10
Prezentacja działania oraz interfejsu aplikacji	11
7. Podsumowanie.....	18
Wymagania, trudności.....	18
Możliwości i predyspozycje do rozbudowy projektu:	18

3. Wymagania projektowe

Wymagania funkcjonalne:

- Dodawanie klientów (terminali RFID, którymi są laboratoryjne zestawy RaspberryPi).
- Usuwanie klientów.
- Przypisywanie kart RFID do pracownika.
- Usuwanie karty RFID przypisanej pracownikowi.
- Dodawanie pracowników
- Usuwanie pracowników
- Rejestrowanie terminu przybycia oraz opuszczenia przez pracownika miejsca pracy, wraz z terminalem RFID, którego użył pracownik.
- Zapisywanie konfiguracji do pliku.
- Odczytywanie konfiguracji z pliku.
- Generowanie raportów z logami z poszczególnego dnia.
- Generowanie raportów z logami z poszczególnego dnia, dla poszczególnych pracowników.
- Generowanie raportów z czasem pracy dla wszystkich pracowników z poszczególnego dnia.
- Generowanie raportów z czasem pracy dla poszczególnego pracownika i dnia.
- Generowanie raportów z czasem pracy.
- Generowanie generalnych raportów z czasem pracy dla wszystkich pracowników, ze wszystkich dostępnych logów w systemie

Diagram przypadków użycia (use case diagram):



Rysunek 1: Diagram przypadków użycia

Wymagania niefunkcjonalne:

Użyte wzorce projektowe / architektoniczne:

- MVC (*Model-View-Controller*)
- Publikacja-subskrypcja (*Publisher-Subscriber*)

Użyty model architektoniczny:

- Klient-serwer

Inne własności:

- Język programowania: *Python 3.7*
- Protokół komunikacyjny: *MQTT 3.1*
- Dokumentacja: w postaci spakowanej do archiwum zip stron internetowej (plik *doc.zip*).
- Multiplatformowość: Program nie wymaga kompilacji pod daną platformę ani maszynę wirtualną. Może być uruchomiony na każdym urządzeniu posiadającym zainstalowany interpreter *Python*, przynajmniej w wersji 3.
- W przypadku awarii zasilania system nie traci rekordów użyć kart RFID.
- Klient nie ma dostępu do danych wysyłanych przez innych klientów
- Pracownik ma możliwość posiadania wielu kart RFID

Baza danych i pliki konfiguracyjne:

- Baza danych w postaci plików w formacie *json* (tekstowym formacie wymiany danych komputerowych, bazującym na podzbiorze języka programowania JavaScript)

4. Opis architektury systemu

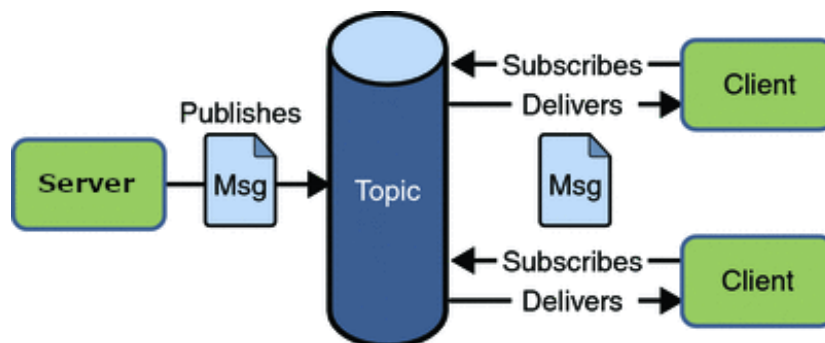
Opis architektury systemu:

System został podzielony na dwie aplikacje:

- Aplikacja centralna, zwana dalej serwerem, która gromadzi i przetwarza dane nadsyłane od klientów
- Aplikacja kliencka, zwana dalej klientem, uruchomiona na zestawie *RaspberryPi*, która wyposażona jest w czytnik kart RFID, komunikująca się z serwerem za pośrednictwem protokołu MQTT

Architektura klient-serwer:

System został stworzony na kształt architektury klient-serwer. Został zastosowany protokół komunikacyjny MQTT (Message Queue Telemetry Transport), oparty o wzorec architektoniczny publikacja/subskrypcja



Rysunek 2: Wzorec publikacja - subskrypcja

Protokół ten przeznaczony jest bardzo często wykorzystywany do komunikacji pomiędzy urządzeniami niewymagającymi dużej przepustowości. Jest też zwykle pierwszym wyborem w przypadku urządzeń Internetu Rzeczy.

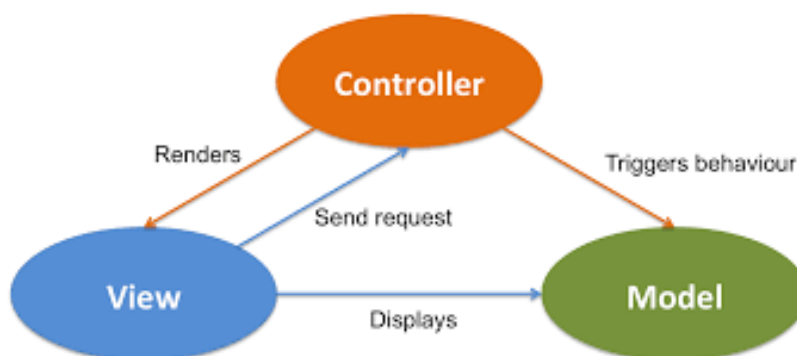
Bezpieczeństwo komunikacji:

Bezpieczeństwo komunikacji zostało zapewnione poprzez użycie szyfrowania w oparciu o protokół SSL oraz dostępu do brokera opartego o system autentykacji klientów

Wzorec architektoniczny MVC:

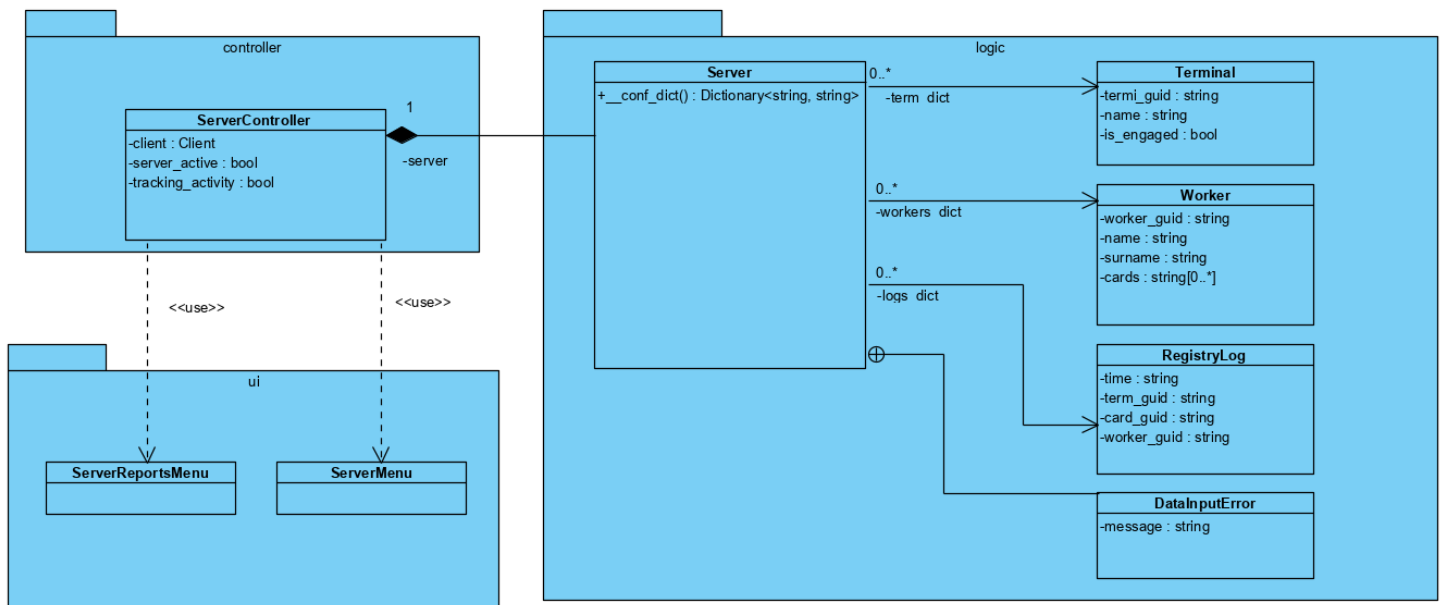
W celu stosownego oddzielenia logiki aplikacji od interfejsu użytkownika, zastosowano uproszczoną wersję wzorca architektonicznego MVC (Model-View-Controller)

- *Model* – model danych - opis struktur danych i powiązań pomiędzy nimi
- *View* - interfejs – warstwa prezentacji, widoczna dla użytkownika
- *Controller* - kontroler – warstwa łącząca interfejs użytkownika z logiką biznesową aplikacji



Rysunek 3: Schemat poglądowy wzorca Model-View-Controller

Diagram klas



Rysunek 4: Uproszczony diagram klas: aplikacja serwera

5. Opis implementacji i zastosowanych rozwiązań

Dokumentacja wszystkich funkcji systemu:

Wszystkie zaimplementowane funkcje systemu posiadają przygotowaną pełną dokumentację w formie strony internetowej, jak i komentarzy w kodzie źródłowym programu

```
def report_log_from_day worker(self, worker_guid, with_saving, date=datetime.now()):  
    """  
    Generate report with all logs added in given date which relate with worker with given GUID  
    :param worker_guid: GUID of worker for whose we generate reports  
    :param with_saving: True - save report in database, False - not save report in database  
    :param date: Returned only logs with date equals <date>  
    :return: List of <RegistryLog> objects  
    """  
    logs_from_day = self.report_log_from_day(False, date)  
    filtered_logs = list(filter(lambda l: l.worker_guid == worker_guid, logs_from_day))  
    if with_saving:  
        report_name = "Report_[LOGS]_" + worker_guid + "_" + date.date().__str__()  
        Server.write_reports(report_name, filtered_logs)  
    return filtered_logs
```

Rysunek 5: Przykład komentarza do dokumentacji w kodzie źródłowym programu

Index

Super-module

app.controller

Classes

ServerController

- available_term_query
- card_reading_query
- connect_to_broker
- disconnect_from_broker
- get_menu_option
- incorrect_option_msg
- log_from_day
- log_from_day_for_worker
- open_menu_option
- process_message
- read_digit
- read_literal
- read_player_date_input
- run
- show_add_terminal_ui
- show_add_worker_ui
- show_assign_card_ui
- show_card_usage_msg
- show_data
- show_remove_terminal_ui
- show_remove_worker_ui
- show_reports_generate_menu
- show_tracking_activity_menu
- show_unassign_card_ui

Module **app.controller.controller**

► EXPAND SOURCE CODE

Classes

class ServerController

► EXPAND SOURCE CODE

Static methods

def get_menu_option()

Read user input for menu option choose, until given input is digit :return: Selected menu number

▼ EXPAND SOURCE CODE

```
@staticmethod
def get_menu_option():
    """
    Read user input for menu option choose, until given input is digit
    :return: Selected menu number
    """
    user_input = input(ui.CHOOSE_MENU_OPTION)
    while not user_input.isdigit():
        ServerController.incorrect_option_msg()
        user_input = input(ui.CHOOSE_MENU_OPTION)
    return int(user_input)
```

def incorrect_option_msg()

Show message in CLI, that given menu option in unknown / incorrect

► EXPAND SOURCE CODE

def read_digit(prompt)

Read user input data until it is not empty and literal is digit (param: prompt: Prompt text used

Rysunek 5: Zrzut ekranu z fragmentu dokumentacji zapisanej w formie strony internetowej

Implementacja protokołu MQTT:

Ustanowienie połączenia:

Serwer:

```
def connect_to_broker(self):
    """
    Establish and setup connection with broker
    """
    config = self.server.get_configs()
    self.__client.tls_set(config["cert_path"])
    self.__client.username_pw_set(username=config["username"], password=config["password"])
    self.__client.connect(config["broker"], config["port"])
    self.__client.on_message = self.process_message
    self.__client.subscribe(config["server_topic"])
    self.__client.loop_start()
```

Klient:

```
def connect_to_broker(self):
    """
    Connect to server using MQTT, subscribe `terminal` topic and send message to server about connecting
    """
    self.__client.tls_set(self.config["cert_path"])
    self.__client.username_pw_set(username=self.config["username"], password=self.config["password"])
    self.__client.connect(self.config["broker"], self.config["port"])
    self.__client.subscribe(self.config["term_topic"])
    self.__client.on_message = self.process_message
    self.__client.publish(self.config["server_topic"], TERM_CONNECTING_QUERY)
```

```
{
  "broker": "ukasz09-os",
  "port": 8883,
  "cert_path": "config/ca.crt",
  "term_topic": "app/terminal",
  "server_topic": "app/server",
  "username": "client",
  "password": "client_password"
}
```

Połączenie się z brokerem MQTT na numerze portu oraz nazwie hosta pobranych z pliku konfiguracyjnego (po lewej plik konfiguracyjny klienta).

Subskrypcja odpowiedniego tematu (pobranego z pliku konfiguracyjnego, pod etykietą „server_topic” oraz „term_topic”). W przypadku serwera uruchomienie pętli głównej odpowiedzialnej za odbieranie komunikatów

Zakończenie połączenia:

Serwer:

```
def disconnect_from_broker(self):
    """
    Disconnect server from broker
    :return:
    """
    self.__client.loop_stop()
    self.__client.disconnect()
```

Serwer zatrzymuje główną pętlę programu i rozłącza się z brokerem MQTT

Klient:

```
def disconnect_from_broker(self):
    """
    Disconnect client terminal from server
    """
    self.__client.publish(self.config["server_topic"], TERM_DISCONNECTING_QUERY + "." + self.__term_guid)
    self.__client.disconnect()
```

Klient przed zakończeniem komunikacji wysyła o tym wiadomość do serwera, a następnie rozłącza się z brokerem MQTT

Przetwarzanie wiadomości z protokołu MQTT:

Serwer:

```
def process_message(self, client, userdata, message):
    """
    Decode and process message from terminal
    :param message: message to process
    """
    decoded = (str(message.payload.decode("utf-8"))).split(".")
    if decoded[0] == TERM_READING_QUERY:
        self.term_reading_query()
    elif decoded[0] == TERM_CONNECTING_QUERY:
        self.term_connecting_query(decoded[0])
    elif decoded[0] == TERM_DISCONNECTING_QUERY:
        self.term_disconnection_query(decoded[0], decoded[1])
    elif decoded[0] == TERM_SELECTED_QUERY:
        self.term_selected_query(decoded[1], decoded[2])
    elif decoded[0] == CARD_READING_QUERY:
        self.card_reading_query(decoded[1], decoded[2])
    else:
        ui.show_msg("Unknown query")
```

Po odbiorze wiadomości w zależności od jej treści program serwera wywołuje poszczególne funkcjonalności systemu

Klient

```
def process_message(self, client, userdata, message):  
    """  
    Decode and process message from server (reading available terminals list)  
    :param message: Message to process  
    """  
  
    # Getting terminal  
    if self.__not_logged:  
        self.__term_list = (str(message.payload.decode("utf-8"))).split(".")
```

Przetwarzanie odebranych wiadomości z serwera sprowadza się tylko do odebrania listy dostępnych terminali

Uwierzytelnienie i autoryzacja:

Podczas komunikacji, wiadomości przesyłane są do brokera MQTT (Mosquitto) ten filtruje przychodzące wiadomości i przekazuje tylko te z tematów które nasłuchuje dany program (w przypadku naszej aplikacji klient (terminal) bądź serwer). Komunikacja pomiędzy brokerem Mosquitto a serwerem / klientem oparta jest o protokół TLS (Transport Layer Security), dzięki któremu jest ona zaszyfrowana. Nasz program zyskuje dzięki temu pewność, że podłączył się do odpowiedniego brokera, oraz że nikt się pod niego podszywa. Kolejnym mechanizmem zabezpieczającym jest wymaganie autoryzacji klientów dołączających się brokera MQTT (login oraz hasło). W plikach konfiguracyjnych brokera ograniczony jest również dostęp do tematu (topic) na którym zachodzi komunikacja

```
def connect_to_broker(self):  
    """  
    Establish and setup connection with broker  
    """  
  
    config = self.server.get_configs()  
    self.__client.tls_set(config["cert_path"])  
    self.__client.username_pw_set(username=config["username"], password=config["password"])  
    self.__client.connect(config["broker"], config["port"])  
    self.__client.on_message = self.process_message  
    self.__client.subscribe(config["server_topic"])  
    self.__client.loop_start()
```

- Ustawienie danych uwierzytelniania w kliencie i serwerze (certyfikat TLS, autoryzacja użytkownika – nazwa i hasło. Jeżeli nazwa użytkownika nie została ujęta w pliku konfiguracyjnym MQTT, bądź hasło jest nieprawidłowe to połączenie nie zostanie nawiązane).
- Połączenie do hosta o nazwie i porcie pobranym z pliku konfiguracyjnego (jeśli nie będą takie same jak w plikach konfiguracyjnych MQTT to połączenie również zostanie odrzucone)
- Subskrypcja tematów (*topic*). Tematy również ujęte są w plikach konfiguracyjnych i muszą być z nimi zgodne by połączenie przebiegło pomyślnie

6. Opis działania i prezentacja interfejsu

Instrukcja użycia:

1. **Przejsć do folderu z aplikacją**
2. **Instalacja wymaganych pakietów MQTT** (jeśli nie ma już zainstalowanych w systemie)

W konsoli wykonać polecenie:

```
pip install -r requirements.txt --user
```

(Windows) Jeżeli pojawia się błąd informujący o nieznanym poleceniu pip wywołaj ją w ten sposób:

```
pip.exe install paho-mqtt (pip.exe znajduje się w głównym katalogu programu)
```

3. **Przygotowanie certyfikatu (przed pierwszym użyciem aplikacji):**

Wygenerowanie pary kluczy:

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

Korzystając z utworzonych kluczy, tworzymy certyfikat:

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

Utworzenie kolejnej pary kluczy, wykorzystywanych przez broker:

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

Utworzenie żądania podpisania certyfikatu:

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

Podpisanie i zweryfikowanie certyfikatów:

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

4. **Utworzenie pliku z użytkownikami i hasłami**

```
mosquitto_passwd -c passwd.conf server (W folderze instalacyjnym Mosquitto)
```

Tworzymy folder certs i kopiujemy do niego pliku: ca.crt, server.crt i server.key.

Modyfikujemy plik mosquitto.conf (albo lokalny plik dla użytkownika w folderze conf.d)

Należy tam dopisać następujące linijki:

```
allow_anonymous false
```

```
password_file //ścieżka do pliku passwd.conf//
```

```
port 8883
```

```
cafile //ścieżka do pliku ca.crt//
```

```
certfile //ścieżka do pliku server.crt//
```

```
keyfile //ścieżka do pliku server.key//
```

5. **Uruchomić skrypt:**

Windows: **run_windows.bat** (dwukrotnie kliknąć LPM)

Linux: **./run_linux.sh** (wpisane w konsoli)

Lub wykonać poleceniem: **python3 main.py**

Prezentacja działania oraz interfejsu aplikacji

Dodawanie terminala do systemu:

```
-----
1) Add new terminal to database
2) Remove terminal to database
3) Add worker to database
4) Remove worker from database
5) Add card to worker
6) Remove card from worker
7) Print workers database
8) Print database records log
9) Print terminals saved in database
10) Generate reports
11) Track activity - show interactive logs with terminals activity
12) Exit - press enter without giving any text to console
-----

Choose menu option: 1
Enter terminal GUID: 987
Enter terminal name: Station 2
Added terminal to server database
Press any key ...
-----
```

W przypadku próby dodania terminala który już istnieje w bazie zostanie pokazany stosowny komunikat, a operacja zostanie przerwana

Usunięcie terminala z systemu:

```
-----
1) Add new terminal to database
2) Remove terminal to database
3) Add worker to database
4) Remove worker from database
5) Add card to worker
6) Remove card from worker
7) Print workers database
8) Print database records log
9) Print terminals saved in database
10) Generate reports
11) Track activity - show interactive logs with terminals activity
12) Exit - press enter without giving any text to console
-----

Choose menu option: 2
Enter terminal GUID: 987
Removed terminal from server database
Press any key ...|
-----
```

W przypadku próby usunięcia terminala który nie istnieje w bazie zostanie pokazany stosowny komunikat, a operacja zostanie przerwana

Dodanie pracownika:

```
-----  
1) Add new terminal to database  
2) Remove terminal to database  
3) Add worker to database  
4) Remove worker from database  
5) Add card to worker  
6) Remove card from worker  
7) Print workers database  
8) Print database records log  
9) Print terminals saved in database  
10) Generate reports  
11) Track activity - show interactive logs with terminals activity  
12) Exit - press enter without giving any text to console  
-----
```

```
Choose menu option: 3  
Enter worker GUID: 6543  
Enter worker name: asd  
Enter worker surname: as  
Worker with GUID: 6543 already exist in database  
Press any key ...|
```

W przypadku próby dodania pracownika który już istnieje w bazie, lub o nieprawidłowym imieniu , nazwisku lub numerze GUID, zostanie pokazany stosowny komunikat, a operacja zostanie przerwana

Usunięcie pracownika z systemu:

```
-----  
1) Add new terminal to database  
2) Remove terminal to database  
3) Add worker to database  
4) Remove worker from database  
5) Add card to worker  
6) Remove card from worker  
7) Print workers database  
8) Print database records log  
9) Print terminals saved in database  
10) Generate reports  
11) Track activity - show interactive logs with terminals activity  
12) Exit - press enter without giving any text to console  
-----
```

```
Choose menu option: 4  
Enter worker GUID: 6543  
Removed worker from server database  
Press any key ...|
```

W przypadku próby usunięcia pracownika który nie istnieje w bazie danych zostanie pokazany stosowny komunikat, a operacja zostanie przerwana

Przypisanie karty RFID do pracownika:

```
-----
1) Add new terminal to database
2) Remove terminal to database
3) Add worker to database
4) Remove worker from database
5) Add card to worker
6) Remove card from worker
7) Print workers database
8) Print database records log
9) Print terminals saved in database
10) Generate reports
11) Track activity - show interactive logs with terminals activity
12) Exit - press enter without giving any text to console
-----

Choose menu option: 5
Enter worker GUID: 4040
Enter card GUID: 452
Assigned card to worker
Press any key ...|
```

W przypadku próby dodania karty RFID do pracownika który nie istnieje, bądź karta którą chcemy dodać jest już zarejestrowana w bazie danych zostanie pokazany stosowny komunikat, a operacja zostanie przerwana

Usunięcie przypisanej karty RFID do pracownika:

```
-----
1) Add new terminal to database
2) Remove terminal to database
3) Add worker to database
4) Remove worker from database
5) Add card to worker
6) Remove card from worker
7) Print workers database
8) Print database records log
9) Print terminals saved in database
10) Generate reports
11) Track activity - show interactive logs with terminals activity
12) Exit - press enter without giving any text to console
-----

Choose menu option: 6
Enter card GUID: 452
Card with id: 452 hasn't been signed to any worker
Press any key ...|
```

W przypadku próby usunięcia karty RFID który nie została przypisana do żadnego pracownika w bazie danych zostanie pokazany stosowny komunikat, a operacja zostanie przerwana

Wyświetlenie zapisanych pracowników w systemie:

```
-----  
Choose menu option: 7  
-----
```

```
GUID: 4040  
Full name: JOHN COOPER  
Cards: ['500500']  
-----
```

```
GUID: 987  
Full name: JOCKO WILLINK SMITH  
Cards: ['1234']  
-----
```

```
Press any key ...|
```

Wyświetlenie logów przedstawiających użycie kart RFID:

```
-----  
Time: 2020-05-09 08:39:34.308578  
Terminal GUID: 123414  
Worker GUID: unknown (not registered)  
Card GUID: 352  
-----
```

```
Time: 2020-05-09 08:39:37.409811  
Terminal GUID: 98765678  
Worker GUID: unknown (not registered)  
Card GUID: 241  
-----
```

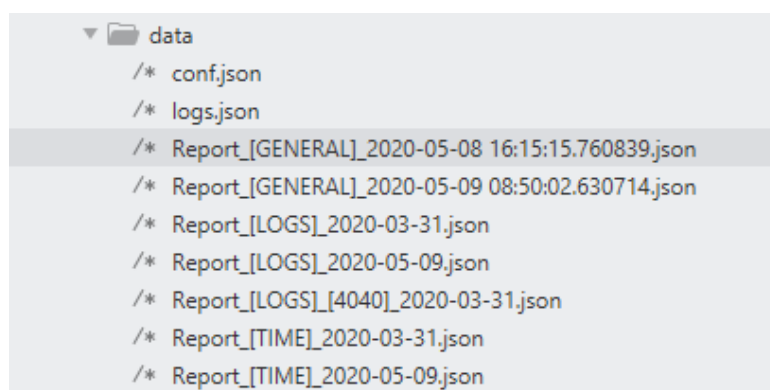
```
Time: 2020-05-09 08:39:39.896468  
Terminal GUID: 123414  
Worker GUID: unknown (not registered)  
Card GUID: 421  
-----
```

```
Time: 2020-05-09 08:39:47.097458  
Terminal GUID: 98765678  
Worker GUID: unknown (not registered)  
Card GUID: 11  
-----
```

```
Time: 2020-05-09 08:39:53.035529  
Terminal GUID: 98765678  
Worker GUID: unknown (not registered)  
Card GUID: 43  
-----
```

```
Press any key ...|
```

Fragment logów z użyciem kart RFID w systemie



Wszystkie wygenerowane raporty są zapisywane w plikach w formacie json

Wyświetlenie zapisanych w bazie terminali RFID:

```
-----  
1) Add new terminal to database  
2) Remove terminal to database  
3) Add worker to database  
4) Remove worker from database  
5) Add card to worker  
6) Remove card from worker  
7) Print workers database  
8) Print database records log  
9) Print terminals saved in database  
10) Generate reports  
11) Track activity - show interactive logs with terminals activity  
12) Exit - press enter without giving any text to console  
-----
```

```
Choose menu option: 9  
-----
```

```
Terminal GUID: 123414  
Terminal name: STATION 1  
-----
```

```
Terminal GUID: 98765678  
Terminal name: Office Hall  
-----
```

```
Press any key ...|
```

Wyświetlenie submenu generowania raportów z wybranym raportem godzinowym z dnia

```
-----  
1) Generate logs from given day  
2) Generate logs from given day and worker  
3) Generate work time report for given worker and day  
4) Generate work time report for all workers for given day  
5) Generate general work time report for all  
-----
```

```
Choose menu option: 4  
Enter date in format YYYY-MM-DD (or nothing for choosing current day) and press enter: 2020-03-31  
-----
```

```
Worker GUID: 4040 Work time: 11:57:00.000900  
-----
```

W przypadku podania nieprawidłowej danych lub daty o nieprawidłowym formacie zostanie pokazany stosowny komunikat, a operacja zostanie przerwana

Generalny raport serwera:

- ```

1) Generate logs from given day
2) Generate logs from given day and worker
3) Generate work time report for given worker and day
4) Generate work time report for all workers for given day
5) Generate general work time report for all

```

```
Choose menu option: 5

```

```
Worker GUID: 4040 Work time: 15:14:42.343601

```

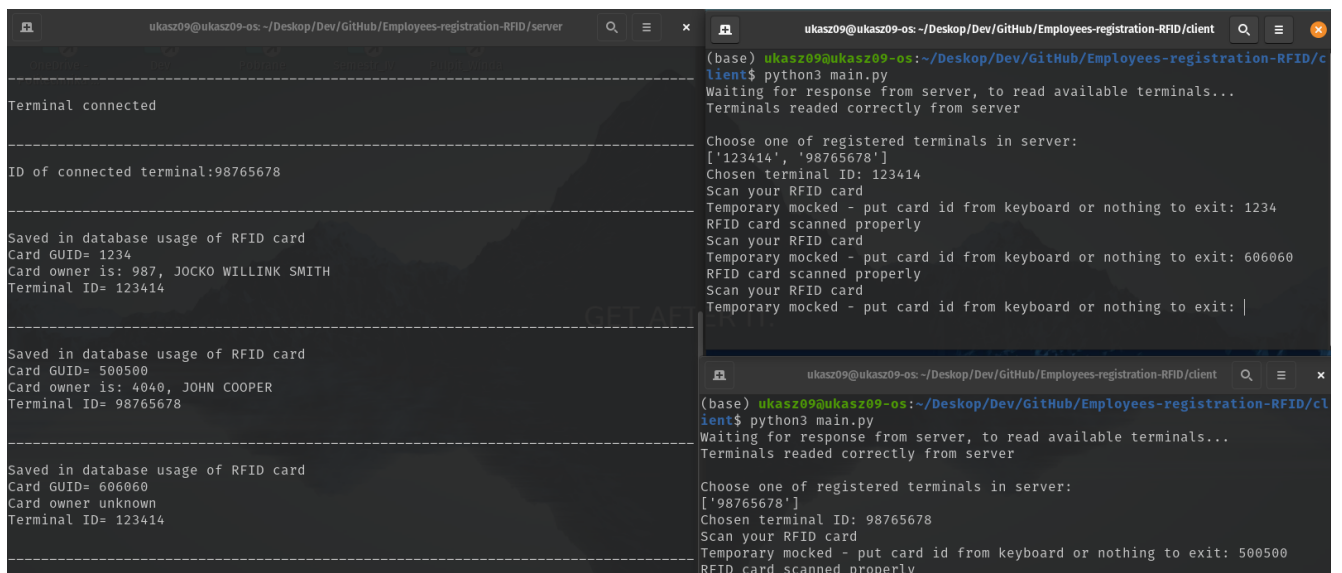
```
Worker GUID: 987 Work time: 11 days, 0:44:32.645723

```

```
Press any key ...|
```

Raport generalny przedstawia czas pracy wszystkich pracowników ze wszystkich dostępnych logów zarejestrowanych w systemie

## Rejestrowanie użyc kart RFID



```
ukasz09@ukasz09-os: ~/Desktop/Dev/GitHub/Employees-registration-RFID/server

Terminal connected

ID of connected terminal:98765678

Saved in database usage of RFID card
Card GUID= 1234
Card owner is: 987, JOCKO WILLINK SMITH
Terminal ID= 123414

Saved in database usage of RFID card
Card GUID= 500500
Card owner is: 4040, JOHN COOPER
Terminal ID= 98765678

Saved in database usage of RFID card
Card GUID= 606060
Card owner unknown
Terminal ID= 123414

ukasz09@ukasz09-os: ~/Desktop/Dev/GitHub/Employees-registration-RFID/client
(base) ukasz09@ukasz09-os:~/Desktop/Dev/GitHub/Employees-registration-RFID/c
lient$ python3 main.py
Waiting for response from server, to read available terminals...
Terminals readed correctly from server

Choose one of registered terminals in server:
['123414', '98765678']
Chosen terminal ID: 123414
Scan your RFID card
Temporary mocked - put card id from keyboard or nothing to exit: 1234
RFID card scanned properly
Scan your RFID card
Temporary mocked - put card id from keyboard or nothing to exit: 606060
RFID card scanned properly
Scan your RFID card
Temporary mocked - put card id from keyboard or nothing to exit: |

ukasz09@ukasz09-os: ~/Desktop/Dev/GitHub/Employees-registration-RFID/client
(base) ukasz09@ukasz09-os:~/Desktop/Dev/GitHub/Employees-registration-RFID/cl
ient$ python3 main.py
Waiting for response from server, to read available terminals...
Terminals readed correctly from server

Choose one of registered terminals in server:
['98765678']
Chosen terminal ID: 98765678
Scan your RFID card
Temporary mocked - put card id from keyboard or nothing to exit: 500500
RFID card scanned properly
```

Po wybraniu z menu serwera „track activities” zostanie wyświetlone okno w którym na bieżąco (*runtime*) będą wyświetlane rejestrowane użycia kart RIFD. Na ilustracji powyżej przykład działania serwera po podpięciu do niego dwóch klientów

## 7. Podsumowanie

---

### Wymagania, trudności

Projekt spełnia wszystkie postawione uprzednio wymagania funkcjonalne. Podczas implementacji nie napotkano na istotne trudności

### Możliwości i predyspozycje do rozbudowy projektu:

Kod jest elastyczny i przygotowany na zamiany. Dzięki wydzieleniu warstw przy użyciu wzorca MVC w prosty sposób możliwa jest podmiana interfejsu użytkownika z CLI (Command Line Interface) na GUI (Graphic User Interface). Kolejną możliwością rozbudowy projektu jest zastąpienie plików bazy danych z formatem json na system bazodanowy (np. poprzez użycie biblioteki bazodanowej sqlite dla języka Python)

### 8. Literatura

[https://eportal.pwr.edu.pl/pluginfile.php/271724/mod\\_resource/content/1/IoT\\_lab\\_5\\_PL.pdf](https://eportal.pwr.edu.pl/pluginfile.php/271724/mod_resource/content/1/IoT_lab_5_PL.pdf)

[https://eportal.pwr.edu.pl/pluginfile.php/297452/mod\\_resource/content/2/MQTT - security - PL v2.pdf](https://eportal.pwr.edu.pl/pluginfile.php/297452/mod_resource/content/2/MQTT - security - PL v2.pdf)

[https://eportal.pwr.edu.pl/pluginfile.php/304348/mod\\_resource/content/1/MQTT - security - authentication authorization - PL.pdf](https://eportal.pwr.edu.pl/pluginfile.php/304348/mod_resource/content/1/MQTT - security - authentication authorization - PL.pdf)

<https://dzone.com/articles/jms-activemq>

<https://www.javacodegeeks.com/2017/09/mvc-delivery-mechanism-domain-model.html>

<https://www.astor.com.pl/poradnikautomatyka/protokol-mqtt-jak-latwo-zbudowac-rozproszony-system-telemetrii/>

<https://pl.wikipedia.org/wiki/JSON>

### 9. Aneks

Kod projektu dostępny na moim repozytorium GitHub:

<https://github.com/Ukasz09/RFID-card-reading>