Raport

System ewidencji czasu pracy
wykorzystujący technologie Internetu Rzeczy

Krzysztof Ruczkowski

14 maja 2020

Spis treści

1 Dane raportu

Przedmiot:	Podstawy Internetu Rzeczy
Imię i nazwisko autora:	Krzysztof Ruczkowski
Nr indeksu:	246739
Semestr studiów:	4
Data ukończenia pracy:	Maj 2020 r.
Prowadzący labolatorium:	mgr inż. Piotr Jóźwiak

2 Wymagania projektowe

2.1 Wymagania funkcjonalne

1. Rejestracja i usuwanie danych terminali odczytujących karty
2. Rejestracja i usuwanie danych pracowników
3. Rejestracja i usuwanie danych kart RFID
4. Przypisywanie i usuwanie przypisań pracowników do kart RFID
5. Rejestrowanie odczytów kart RFID
6. Generowanie pliku csv zawierającego czasy przychodzenia i wychodzenia z pracy pracowników

2.2 Wymagania niefunkcjonalne

1. Użycie Pythona 3 do implementacji
2. Użycie MQTT do komunikacji
3. Użycie tkinter do wyświetlania GUI
4. Użycie bazy danych SQLite do trwałego przechowania danych
5. Bezpieczny przesył danych (autoryzacja i uwierzytelnianie)

3 Opis architektury systemu

System (`server.py`) to aplikacja nasłuchująca uwierzytelnionych wiadomości przesyłanych przez MQTT zawierającyh informacje dotyczące odczytu karty. Aplikacja jest wyposażona w GUI, za pomocą którego możemy spełnić resztę wymagań funkcjonalnych. Możemy zasymulować odczyt karty za pomocą `client.py`.

4 Opis implementacji i zastosowanych rozwiązań

4.1 Baza danych

Została wykorzystana baza danych SQLite, przykład z pliku `src/appdata.py`:

```
def fetchall(sql):
    return c.execute(sql).fetchall()

def get_employees():
    return fetchall("SELECT * FROM Employee")

def get_cards():
    return fetchall("SELECT * FROM Card")

def get_card_readings():
   return fetchall(
"""
SELECT
Employee.id, Employee.name, Terminal.name, readTime
FROM CardReading
INNER JOIN Card ON CardReading.rfid = Card.rfid
INNER JOIN Terminal ON CardReading.terminalId = Terminal.id
LEFT JOIN Employee ON Card.employeeId = Employee.id
""")

def get_terminals():
    return fetchall("SELECT * FROM Terminal")

def insert_employee(name):
    c.execute("INSERT INTO Employee (name) VALUES (?)", (name,))
    conn.commit()
```

4.2 Generacja pliku csv z czasami

Za generację pliku odpowiada funkcja w `src/app/controller.py`:

```
def generate_time_report():
    logger.log("Generating time report...")

    cr = data.get_card_readings()

    employee_ids_on_readings = set(x[0] for x in cr)
    cr_by_employee_id = {y:[(x[2], x[3]) for x in cr if x[0] == y] for y in employee_
    cr_by_employee_id_paired = {k:list(zip(v[::2], v[1::2])) for (k, v) in cr_by_empl
    i = cr_by_employee_id_paired.items()
```

```
    csv = "\n".join([",".join([str(k or "None"), t[0][0], t[0][1], t[1][0], t[1][1]])

    with open("time_report.csv", "w") as file:
        file.write(csv)

    logger.log("Time report generated.")
```

Może wyglądać nieco skomplikowanie, dlatego skupmy się na nazwach zmiennych. Najpierw wyciągamy dane z bazy odnośnie odczytów kart do zmiennej `cr`. Operujemy na tej zmiennej, najpierw wyciągając zbiór ID pracowników na odczytach. Następnie tworzymy słownik, którego kluczem jest ID pracownika, a wartością jego odczyty z karty. Potem wystarczy połączyć odczyty w pary, gdzie pierwszy odczyt jest czasem przyjścia do pracy a drugi czasem wyjścia, sformatować te pary do formatu csv i zapisać plik.

5 Opis działania i prezentacja interfejsu

5.1 Przygotowanie środowiska

5.1.1 Konfiguracja Mosquitto

Należy skonfigurować MQTT dodając uwierzytelnianie i autoryzację, tworząc użytkowników 'client' i 'server' oraz używając certyfikatów z folderu `certs`.

Zawartość aclfile.conf:

```
user server
topic read app/card_reading

user client
topic app/card_reading
```

5.1.2 Pobranie zależności Python'a

```
cd src
./setupenv.sh
source venv/bin/activate
```

5.2 Uruchomienie aplikacji

Sprawdzić działanie aplikacji można uruchamiając serwer i klientów symulujących terminale:

```
./server.py &; ./client.py &; ./client.py &;
```

5.3 Screeny

Screeny przedstawiające działanie aplikacji znajdują się w folderze `screens`.

6 Podsumowanie

Projekt jest zgodny ze wszystkimi wymaganiami funkcjonalnymi.
Trudnością w implementacji było użycie tkinter'a z bazą danych, której trzeba otwarcie umożliwić działanie na wielu wątkach. Samo zdebugowanie tego problemu wymagało czasu ze względu na domyślne ignorowanie wyjątków przez tkinter (dopiero użycie pdb (python debugger) pomogło).
Rozbudowa projektu nie powinna stanowić problemu ze względu na modularność i rozdzielenie odpowiedzialności między moduły.

7 Aneks

Kod projektu znajduje się na eportalu.