Bartłomiej Jagiełło 254521

Piotr Kołpa 254557

Michał Najwer 254560

Agata Rudzka 242466

**Projekt Programistyczny IoT**

**System do obsługi kart parkingowych**

**Podstawy Internetu Rzeczy laboratorium 2021/2022**

Prowadzący:

dr inż. Krzysztof Chudzik

## 2. Spis treści

[2. Spis treści 2](#_Toc93147072)

[3. Wymagania projektowe 3](#_Toc93147073)

[1. Wymagania funkcjonalne 3](#_Toc93147074)

[2. Wymagania niefunkcjonalne 3](#_Toc93147075)

[3. Użytkownicy systemu 4](#_Toc93147076)

[4. Diagram Przypadków Użycia 4](#_Toc93147077)

[4. Schemat architektury systemu 5](#_Toc93147078)

[Schemat bazy danych 5](#_Toc93147079)

[5. Opis implementacji i zastosowanych rozwiązań 7](#_Toc93147080)

[5.1. Front-end 7](#_Toc93147081)

[5.2. Back-end 7](#_Toc93147082)

[Opis komunikacji mqtt 8](#_Toc93147083)

[Diagram sekwencji przesyłanych wiadomości 8](#_Toc93147084)

[Zabezpieczenia komunikacji mqtt 9](#_Toc93147085)

[Fragment kodu odpowiedzialny za ustawianie komunikacji mqtt na serwerze 10](#_Toc93147086)

[Obsługa wiadomości na serwerze 12](#_Toc93147087)

[Fragmenty kodu odpowiedzialny za ustawianie komunikacji mqtt na klientach (szlabanów i czytnika): 13](#_Toc93147088)

[Ustawienia brokera 14](#_Toc93147089)

[Przykład komunikacji 15](#_Toc93147090)

[6. Opis działania i prezentacja interfejsu 17](#_Toc93147091)

[7. Szczegółowy opis wkładu pracy Autorów 19](#_Toc93147092)

[8. Podsumowanie 21](#_Toc93147093)

[9. Literatura 22](#_Toc93147094)

[10. Aneks 23](#_Toc93147095)

## 3. Wymagania projektowe

System obsługi parkingu będzie zajmował się przechowywaniem w bazie danych informacji jakie osoby aktualnie korzystają z danego parkingu, będzie kontrolował szlabany wjazdowe i wyjazdowe. Każda osoba uprawniona do korzystania z parkingu będzie posiadała swoją unikatową kartę RFID przeznaczoną do identyfikacji.

### Wymagania funkcjonalne

1. Podnoszenie szlabanu po zeskanowaniu aktywnej karty RFID i opuszczenie po chwili.
2. Monitorowanie stanu zapełnienia parkingu i nie wpuszczanie nowych użytkowników jeśli jest pełny.
3. Monitorowanie wjazdów i wyjazdów z parkingu przy pomocy czytników kart RFID.
4. Blokowanie prób wielokrotnego wjazdu na tą samą kartę bez wyjazdu.
5. Dodawanie kart RFID skojarzonych z konkretnym użytkownikiem poprzez imię i nazwisko.
6. Blokowanie kart RFID.
7. Aktywowanie kart RFID.
8. Zmiana właściciela karty RFID.
9. Przeglądanie listy kart RFID.
10. Wyświetlanie danych karty RFID w tym danych o użytkowniku i historii wjazdów / wyjazdów.
11. Konto administratora odpowiedzialne za zarządzanie systemem kart RFID.
12. Dodawanie nowych szlabanów po identyfikatorze.

### Wymagania niefunkcjonalne

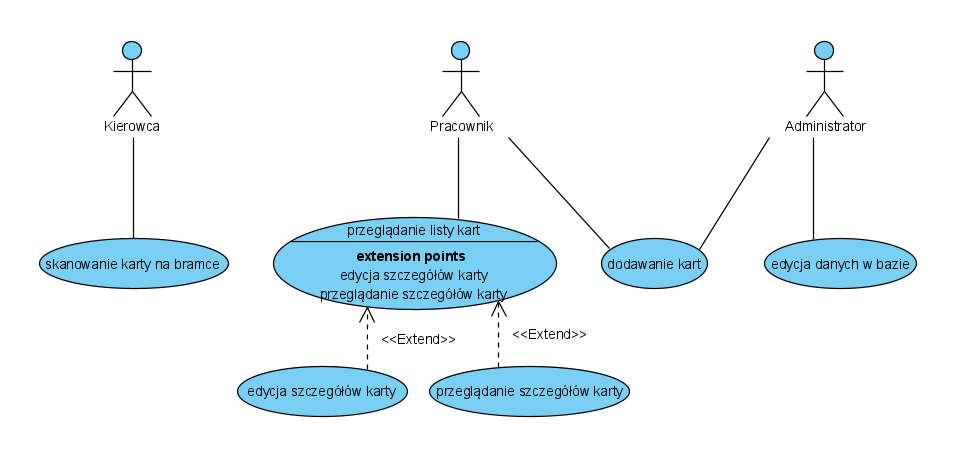
1. Aplikacja webowa działająca na przeglądarkach Google Chrome, Mozilla Firefox, Microsoft Edge, Safari.
2. Aplikacja webowa działająca na systemach windows (11, 10, 8, 7) i linux (przynajmniej ubuntu, debian, redhat).
3. Obsługa 24/7.
4. Możliwość rozszerzenia systemu o następne urządzenia: szlabany wjazdowe, wyjazdowe.
5. Obsługa wielu użytkowników jednocześnie.
6. Zabezpieczenie przed nieautoryzowanym połączeniem poprzez klucze.

### Użytkownicy systemu

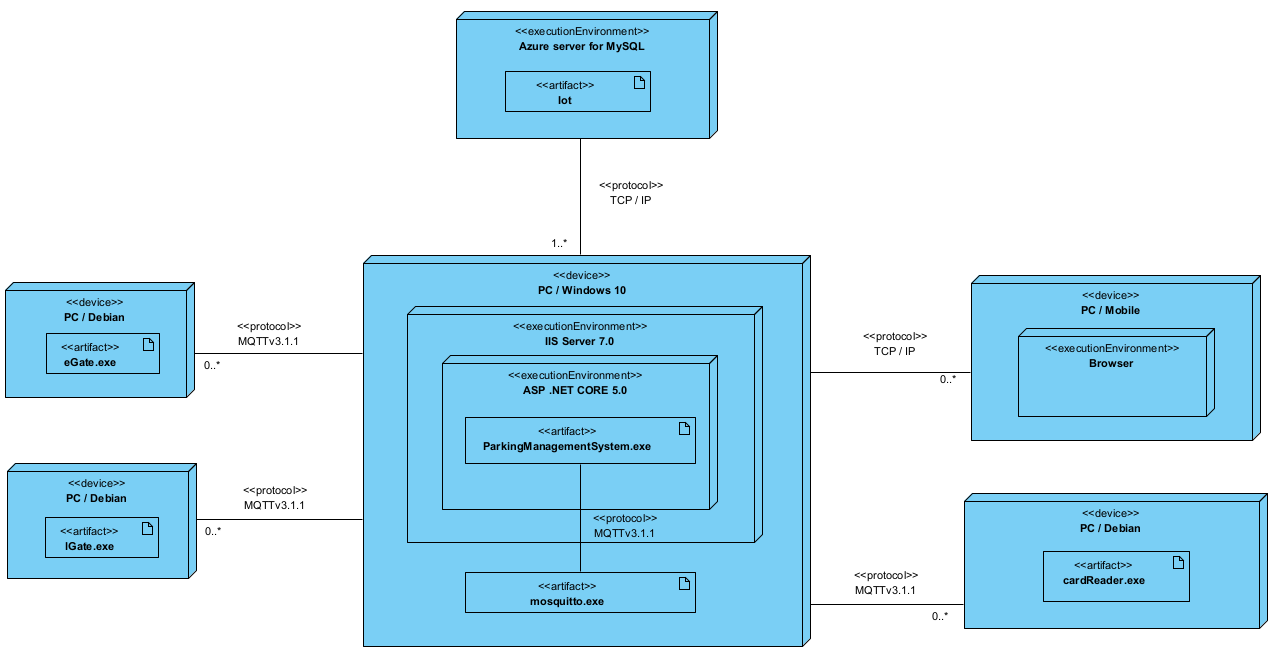
W niniejszym rozdziale zdefiniowano grupy użytkowników, korzystających z projektowanego systemu:

* Pracownik - osoba posiadająca uprawnienia pozwalające na dodawanie nowych kart do systemu; usuwanie, aktywowanie i blokowanie kart; przypisywanie właściciela do karty oraz przeglądanie danych kart parkingowych.
* Kierowca - osoba będąca właścicielem karty RFID, która ma możliwość skanować przy wjeździe i wyjeździe z parkingu.
* Administrator - użytkownik o uprawnieniach pozwalających na zarządzanie kartami RFID i dodawanie nowych szlabanów.

### Diagram Przypadków Użycia



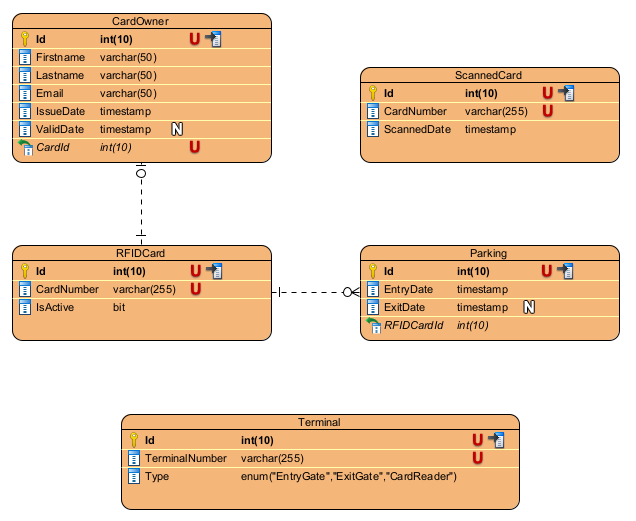
## 4. Schemat architektury systemu

****

**1** Schemat rozmieszczenia

### Schemat bazy danych

W ramach projektu zaprojektowano i zaimplementowano relacyjną bazę danych przechowująca numery kart, oraz ich właścicieli. Umożliwia ona sprawdzenie czy danej karcie nie skończył się jeszcze termin ważności oraz czy nie została zablokowana. W celu bezpieczniejszego dodawania kart do systemu, tymczasowo zeskanowane nowe karty są przechowywane w osobnej tabeli oczekując na zaakceptowanie przez administratora. Karty RFID mogą występować bez właściciela, lecz wtedy domyślnie są jako nieaktywne i nie da się z nich korzystać. W momencie wjazdu na parking dodaje się rekord do tabeli Parking, wraz z datą wjazdu oraz identyfikatorem karty. W momencie wyjazdu dany rekord jest uzupełniany o datę wyjazdu. W celu lepszej identyfikacji terminali ich numery wraz z funkcją są przechowywane w osobnej tabeli.



Schemat bazy danych

## 5. Opis implementacji i zastosowanych rozwiązań

### 5.1. Front-end

### 5.2. Back-end

Do implementacji back-endu zastosowano poniższe technologie:

Klient (czytnik kart):

* python w wersji 3.8.10 - język programowania wysokiego poziomu umożliwiając korzystanie z dużej ilości bibliotek znacznie usprawniających zadania takie jak generowanie interfejsu użytkownika.
* protokół MQTT v3.1.1 - prosty protokół transmisji danych oparty o wzorzec   
  publikacja - subskrypcja. Rozwiązanie umożliwia przesył informacji między urządzeniami w ramach zdefiniowanego tematu.
* biblioteka tkinter - biblioteka języka Python umożliwiająca i usprawniająca tworzenie interfejsu graficznego.
* biblioteka eclipse paho mqtt - biblioteka kliencka języka Python implementująca protokół MQTT i umożliwiająca komunikację z brokerem.

Server:

* framework ASP .NET Core 5.0
* framework Entity Framework Core
* baza danych w technologii MySQL
* biblioteka MQTTnet 3.1.1 - biblioteka pozwalająca na wykorzystanie protokołu mqtt w języku c# do komunikacji

Implementację oraz testy implementacji po stronie czytnika kart wykonano przy użyciu programu Visual Studio Code z zainstalowanym rozszerzeniem Remote-SSH umożliwiającym testowanie kodu na “osobnych” maszynach.

### Opis komunikacji mqtt

Komunikacja odbywa się na porcie 8883 który jest domyślnym portem protokołu mqtt przy użyciu protokołu tls.

Klienci mqtt wysyłają wiadomości ze swoim tematem

W przypadku bram szlabanów są to odpowiednio gate/e/id\_klienta - dla bramy wjazdowej oraz gate/l/id\_klienta - dla bramy wyjazdowej

Bramy subskrybują się na wiadomości o tematach gate/e/id\_klienta/r oraz gate/l/id\_klienta/r odpowiednio dla bramy wjazdowej i wyjazdowej - na te tematy serwer odsyła odpowiedź czy szlaban należy podnieść czy nie

Czytnik kart wysyła wiadomości z tematem reader/id\_klienta

Serwer mqtt subskrybuje się na wszystkie wiadomości z tematami zaczynającymi się od gate/e, gate/l, reader i zawierającymi id klientów.

Następnie po skomunikowaniu z bazą danych i ustaleniu odpowiedzi przesyła odpowiedź na tematy gate/e/id\_klienta/r oraz gate/l/id\_klienta/r odpowiednio dla bramy wjazdowej i wyjazdowej.

Wszystkie wiadomości wysyłane są z flagą qos=2 co zapewnia, że broker dostarczy każdą wiadomość co najwyżej raz. Ilość przesyłanych danych w systemie nie jest duża, natomiast ważne jest aby każda wiadomość dotarła tylko raz, inaczej może dojść do sytuacji w których szlabany otrzymają kilkukrotnie polecenie otwarcia po zeskanowaniu tej samej karty.

Użyta jest również flaga retain=false żeby broker nie przesyłał wiadomości jeśli klienci byli rozłączeni. Np. jeśli szlaban utraci połączenie z siecią i odnowi po 5 minutach, nie powinna do niego przyjść wiadomość sprzed 5 minut nakazująca otworzenie.

Tak samo serwer nie musi sprawdzać wiadomości jeśli był rozłączony, kierowca albo zrezygnował już z wjazdu na parking albo spróbował ponownie, nie ma sensu przetwarzać starego żądania, eliminuje to też problem jak w przypadku szlabanu.

#### Diagram sekwencji przesyłanych wiadomości

Diagram dotyczy wiadomości przesyłanych przez szlabany (zarówno wjazdowe jak i wyjazdowe) do serwera

Diagram dla wiadomości przesyłanych przez czytnik kart wygląda podobnie, z tą różnicą, że serwer nie odpowiada wiadomością publish zatem komunikacja kończy się na wiadomości 2.1.2.

Obraz zawierający stół

Opis wygenerowany automatycznie

Diagram sekwencji przesyłanych wiadomości

#### Zabezpieczenia komunikacji mqtt

Całość komunikacji odbywa się przy użyciu protokołu tls co sprawia, że wiadomości są zaszyfrowane. Klienci szlabanów i czytnika korzystają z wersji 1.3, serwer z wersji 1.2, broker pozwala zarówno na wersję 1.3 jak i 1.2. Ponadto do autentykacji klientów, serwera i brokera używane są certyfikaty x.509, każdy klient posiada swój własny certyfikat podpisany przez urząd certyfikacji (CA), broker i serwer posiadają oddzielne certyfikaty. Zatem każda ze stron weryfikuje czy druga strona komunikacji ma certyfikat podpisany przez zaufany urząd certfikacji.

Uwaga: Z powodu pewnych ograniczeń w kodzie pythona, aktualnie weryfikacja tożsamości brokera przez klientów jest wyłączona. Jest to spowodowane wyłącznie tym, że roboczo certyfikaty są podpisywane lokalnie i serwery CA nie rozpoznają lokalnego CA i cała komunikacja odbywa się wewnątrz sieci lokalnej. Jednakże podpisanie certyfikatu przez znany CA kosztuje dlatego na potrzeby projektu użyty został lokalny CA. Po stronie serwera udało się to obejść poprzed nadpisanie funkcji do weryfikacji tożsamości i podanie jej certyfikatu lokalnego CA. Podobnie na brokerze podany jest certyfikat lokalnego CA.

Jeśli certyfikaty są prawidłowe następnym etapem autentykacji jest sprawdzenie czy login i hasło podane przez klienta/serwer zgadzają się z tymi zapisanymi na brokerze.

Jeśli powyższe warunki zostaną spełnione może dojść do komunikacji na następujących zasadac:

Serwer może czytać wiadomości wysłane na tematy:

gate/e/id\_klienta

gate/l/id\_klienta

reader/id\_klienta

i wysyłać wiadomości na tematy:

gate/e/id\_klienta/r

gate/l/id\_klienta/r

Klienci mogą czytać wiadomości wysłane na tematy:

gate/e/id\_klienta/r

gate/l/id\_klienta/r

i wysyłać wiadomości na tematy:

gate/e/id\_klienta

gate/l/id\_klienta

reader/id\_klienta

(Klienci mogą czytać/subskrybować na tematy teoretycznie nie związane z nimi, jest to spowodowane ograniczeniami acl gdzie bez podawania konkretnej nazwy użytkownika nie można inaczej wydzielić tematów, np. W jakiś sposób grupując użytkowników )

#### Fragment kodu odpowiedzialny za ustawianie komunikacji mqtt na serwerze

Obraz zawierający tekst

Opis wygenerowany automatycznie

Kod ustawiający komunikację mqtt na serwerze

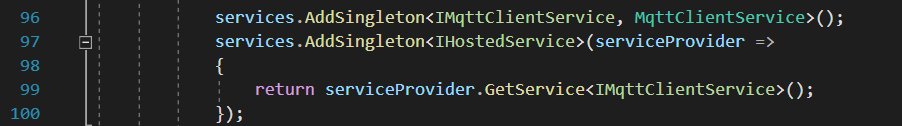
Ustawiane są nazwa użytkownika, hasła, id. Clean session ustala, że broker ma nie przechowywać wiadomości dla serwera jeśli nie jest on podłączony (żądania obsługiwane są natychmiast albo wcale). Keep alive period określa, że co 30 sekund ma być wysyłany komunikat między serwerem i brokerem (jeśli w tym czasie nie są przesyłane inne wiadomości) potwierdzający, że oba są podłączone. Ustawiany jest adres brokera i port do komunikacji. Następnie ustawiany jest protokół tls w wersji 1.2 i podawany jest certyfikat używany do autentykacji.

Dane wczytywane z pliku konfiguracyjnego appsettings.json, ale nic nie stoi na przeszkodzie aby hasła były wymagane od użytkownika przy starcie serwera:

Obraz zawierający tekst

Opis wygenerowany automatycznie

Plik konfiguracyjny serwera

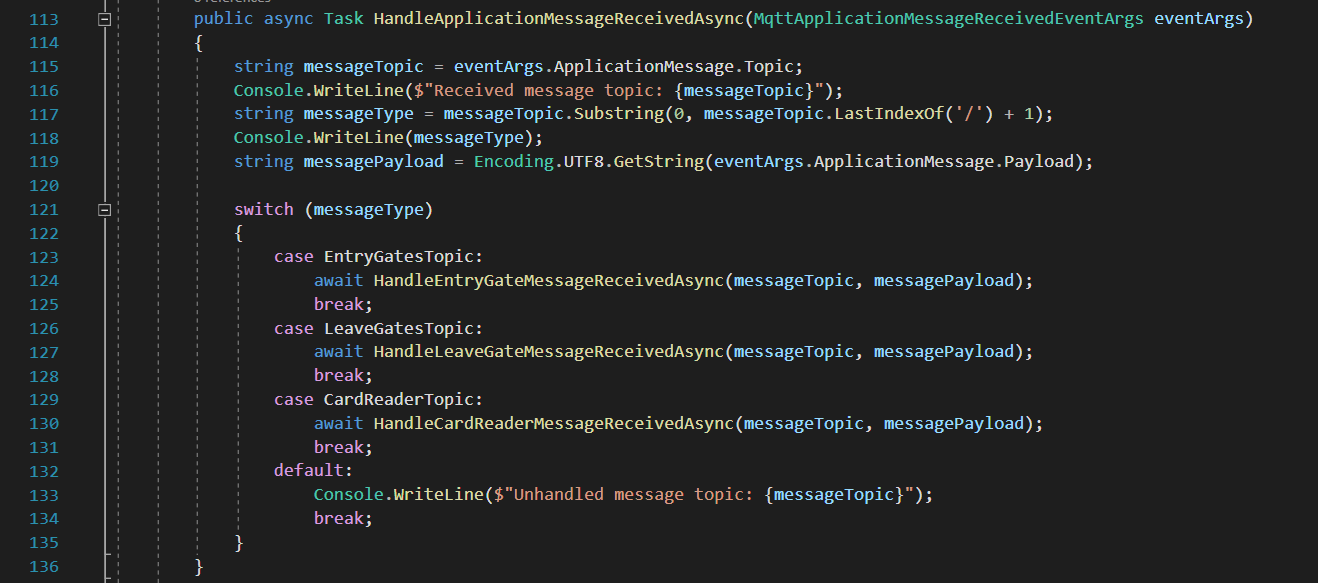


Tworzenie usługi klienta mqtt

#### Obsługa wiadomości na serwerze

Za obsługę wiadomości i innych zdarzeń związanych z mqtt odpowiedzialna jest klasa MqttClientService

Metoda odpowiedzialna za obsłużenie przychodzących wiadomości odczytuje temat wiadomości oraz treść i wywołuje odpowiednią metodę do obsługi klienta



Obsługa przychodzących wiadomości - ogólnie

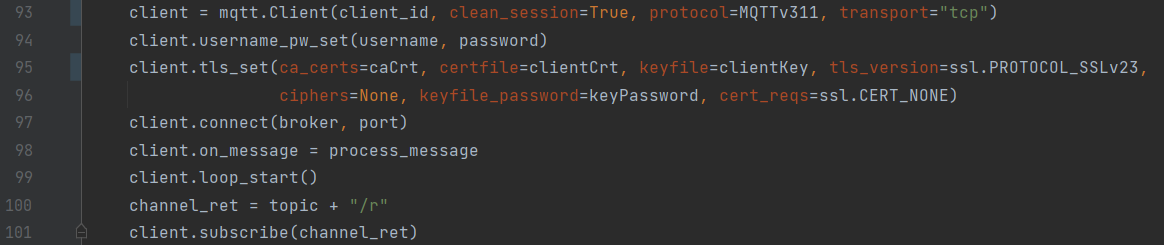
Metoda odpowiedzialna za obsługę szlabanów wjazdowych odczytuje identyfiaktor szlabanu (z tematu) i numer karty (z wiadomości), następnie wywołuje metodę CheckEntry która sprawdza czy szlaban należy otworzyć i odsyła tę informację do klienta szlabanu

Obraz zawierający tekst

Opis wygenerowany automatycznie

Obsługa wiadomości od szlabanów wjazdowych

#### Fragmenty kodu odpowiedzialny za ustawianie komunikacji mqtt na klientach (szlabanów i czytnika):



Kod ustawiający komunikację mqtt u klienta

Tworzony jest klient o podanym id, nazwie użytkownika i haśle. ustawiane są wersje protokołu mqtt i protokołu do transportu oraz clean\_session=True aby po rozłączeniu się klienta nie były zapisywane żadne wysłane do niego wiadomości (podwójne zabezpieczenie, drugie takie ustawienie jest na brokerze).

ca\_certs - certyfikat CA który weryfikuje tożsamość brokera

certfile - certfikat klienta keyfile - klucz klienta

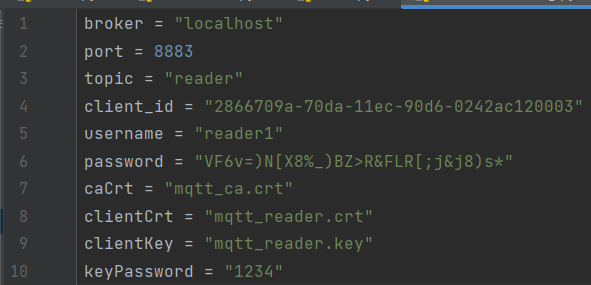
tls\_version - klient będzie korzystał z najlepszej dostępnej wersji tls (lub ssl) na brokerze w tym wypadku 1.3

ciphers - użycie domyślnych algorytmów do szyfrowania wiadomości

keyfile\_password - hasło do pliku z kluczem klienta

cert\_reqs - czy weryfikować tożsamość brokera, aktualnie nie weryfikowana z powodu podpisywania certyfikatu brokera lokalnie, w środowisku produkcyjnym CERT\_REQUIRED

Następnie następuje połączenie z brokerem o adresie ip i porcie wczytanym z pliku, ustawiona zostaje metoda do obsługi przychodzących wiadomości, uruchamiany klient i subskrypcja na odpowiednie tematy (dla czytnika kart nie ma ostatnich 2 linijek i metody on\_message)



Plik konfiguracyjny klienta

W środowisku produkcyjnym wartości client\_id, username i password oraz ścieżki do plików powinny być przechowywane w pamięci tak aby odczytanie ich było niemożliwe.

#### Ustawienia brokera

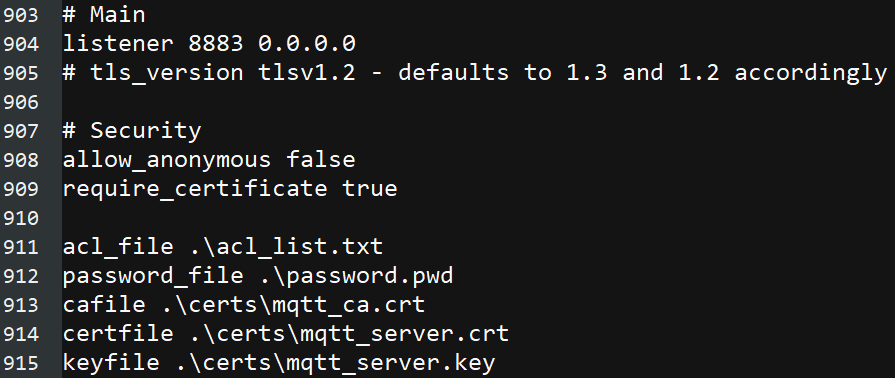
Aktualnie broker korzysta z portu 8883 na adresie localhost (w środowisku produkcyjnym powinien to być adres maszyny na której uruchomiony będzie broker). Obsługiwane są protokoły tls 1.2 i 1.3. Zabronione jest połączenie klientów bez loginu i hasła, a podane dane muszą być zgodne z tymi przechowywanymi na brokerze.

Acl\_file - plik acl ustalający tematy dla połączonych urządzeń

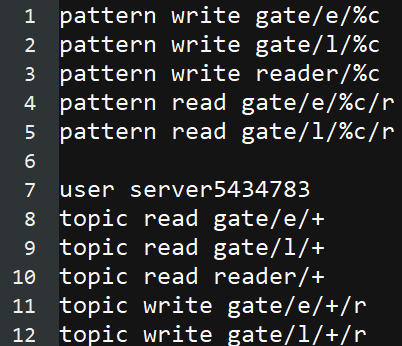
Password\_file - plik z loginami i (zaszyfrowanymi) hasłami klientów

Broker wymaga też poprawnych certyfikatów od wszystkich swoich klientów podpisanych przez CA o sygnaturze podanej w pliku ustawianym przez cafile.

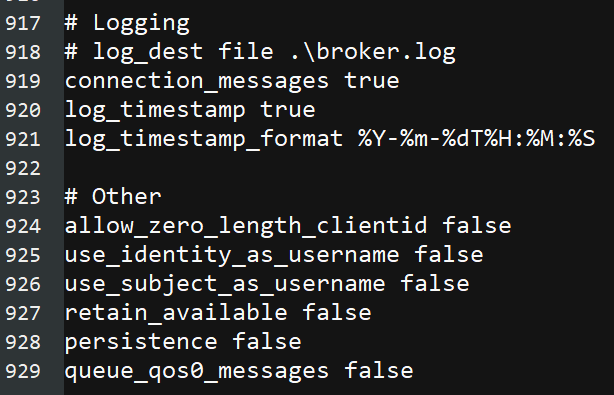
Certfile i keyfile to odpowiednio certyfikat i klucz prywatny brokera.



Ustawienia zabezpieczeń brokera



Plik acl określający dostępne tematy



Inne ustawienia brokera

#### Przykład komunikacji

Dzięki programowi Wireshark możemy zobaczyć jak wygląda ruch pakietów tcp na porcie 8883 (komunikację mqtt).

Po uruchomieniu brokera i serwera widzimy komunikację przy użyciu TLS1.2. Wymieniane i sprawdzane są certyfikaty, ustanawiane jest połączenie. Następnie w application data następuje subskrypcja serwera.

Obraz zawierający tekst

Opis wygenerowany automatycznie

14 Połączenie serwera z brokerem

Jeśli zajrzymy do wnętrza okaże się, że dane są nieczytelne i niezrozumiałe

Obraz zawierający tekst

Opis wygenerowany automatycznie

15 Dane z połączenia serwera z brokerem

Następnie uruchamiam klienta mqtt (szlaban wjazdowy w tym przypadku) i następuje podobny proces (inna wersja protokołu)

Obraz zawierający tekst

Opis wygenerowany automatycznie

16 Połączenie klienta z brokerem

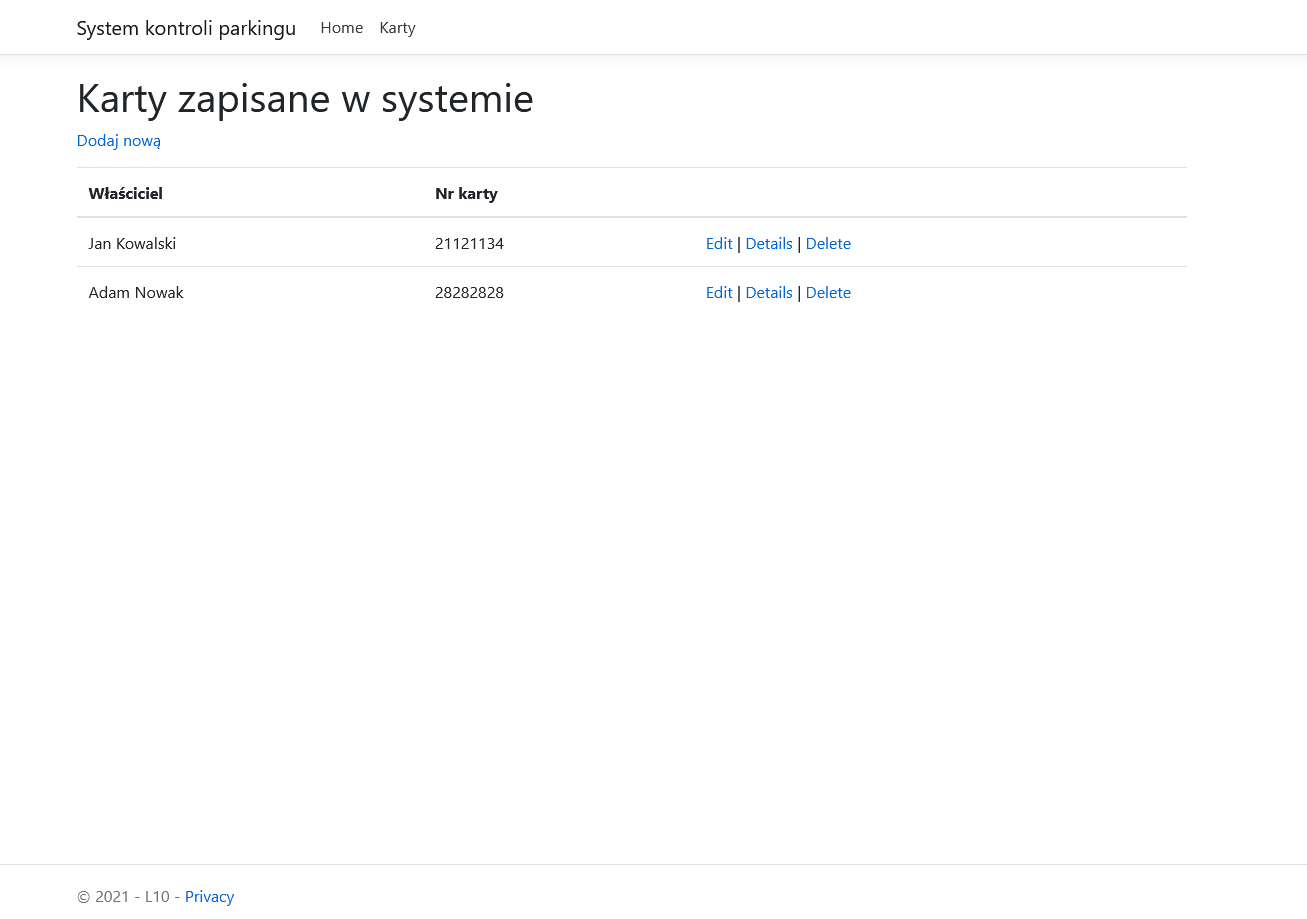
Następnie wysyłany jest numer odczytanej karty RFID, ponieważ w całej komunikacji ustawiony jest QoS 2, oprócz danych wysyłanych jest też dużo potwierdzeń, jednak po wersjach protokołu tls i numerach portów łatwo można zauważyć, że w zaznaczonym fragmencie dane o wczytanej karcie trafiają na serwer i niemożliwe jest odczytanie treści (treść wiadomości to „card:25425”).

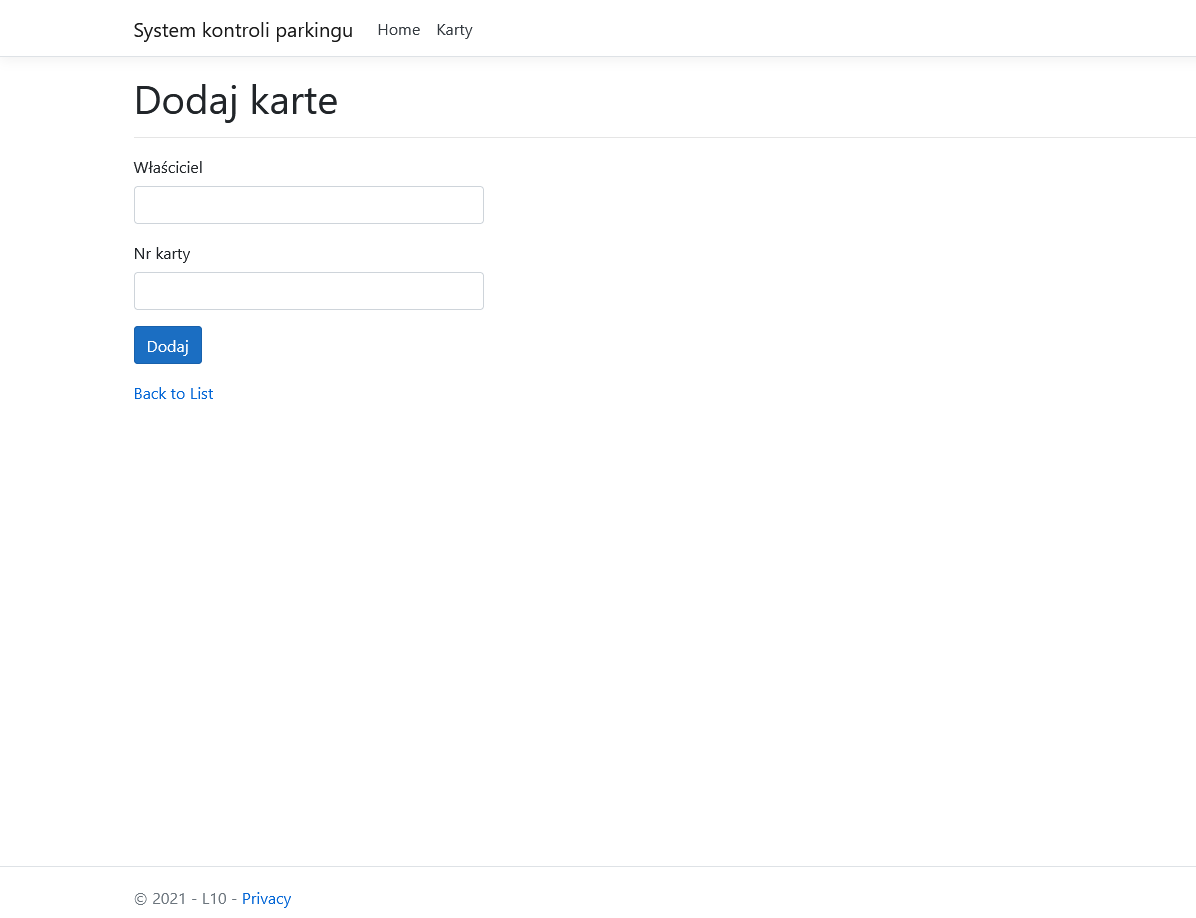
Obraz zawierający tekst

Opis wygenerowany automatycznie

17 Dane przesłane między klientem, a serwerem

## 6. Opis działania i prezentacja interfejsu





## 7. Szczegółowy opis wkładu pracy Autorów

Bartłomiej Jagiełło: - konfiguracja brokera mqtt

- ustalanie formatu wiadomości mqtt 50%

- rejestracja i obsługa serwera mqtt w .net

- obsługa wiadomości po stronie klienta mvc i przekazanie do logiki

- zabezpieczenia protokołu mqtt

- dokumentacja mqtt w Visual Paradigm

- testy integracyjne mqtt

- testy jednostkowe mqtt w .net i brokera

Agata Rudzka: - implementacja obsługi czytnika kart w pythonie (pobieranie wartości,  
 przesył i obsługa informacji zwrotnej)

* dokumentacja przypadków użycia w Visual Paradigm
* testy implementacji
* pisanie dokumentacji, definiowanie wymagań i użytkowników

Michał Najwer:   
  
Piotr Kołpa: - projekt bazy danych

- dokumentacja diagramu ERD bazy danych w Visual Paradigm

- implementacja modeli MVC   
- implementacja bazy danych za pomocą Entity Framework

- wygenerowanie certyfikatów x.509 odpowiedzialnych za zabezpieczenie połączenia między klientem mvc a bazą danych (WIP)

- ustalanie formatu wiadomości mqtt 50%  
- implementacji analizy otrzymanych wiadomości pod względem poprawności numerów oraz aktywności kart, dat wjazdów i wyjazdów, oraz numerów i funkcji terminali

- implementacja kodów błędów precyzujących dlaczego nie nastąpi otwarcie szlabanu (WIP)

## 

## 8. Podsumowanie

## 

## 9. Literatura

1. [Dokumentacja mqqt w .net](https://github.com/dotnet/MQTTnet)
2. [Zabezpieczanie protokołu mqtt](https://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl/)
3. [Ustawianie ACL dla mqtt](https://medium.com/jungletronics/mosquitto-acls-ac062aea3f9)
4. [Dokumentacja mosquitto dla .conf](https://mosquitto.org/man/mosquitto-conf-5.html)

## 

## 10. Aneks