

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
3. Wymagania projektowe	3
1. Wymagania funkcjonalne	3
2. Wymagania niefunkcjonalne	3
3. Użytkownicy systemu	4
4. Diagram Przypadków Użycia	4
4. Schemat architektury systemu	5
Schemat bazy danych	5
5. Opis implementacji i zastosowanych rozwiązań	7
5.1. Front-end	7
5.2. Back-end	7
Opis komunikacji mqtt	8
Diagram sekwencji przesyłanych wiadomości	8
Zabezpieczenia komunikacji mqtt	9
Fragment kodu odpowiedzialny za ustawianie komunikacji mqtt na serwerze	10
Obsługa wiadomości na serwerze	12
Fragmenty kodu odpowiedzialny za ustawianie komunikacji mqtt na klientach (szlabanów i czytelnika):	13
Ustawienia brokera	14
Przykład komunikacji	15
6. Opis działania i prezentacja interfejsu	19
7. Szczegółowy opis wkładu pracy Autorów	22
8. Podsumowanie	23
9. Literatura	24
10. Aneks	25

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.

1. Wymagania funkcjonalne

1. Podnoszenie szlabanu po zeskanowaniu aktywnej karty RFID i opuszczenie po chwili.
2. Monitorowanie stanu zapelnienia 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.

2. Wymagania niefunkcjonalne

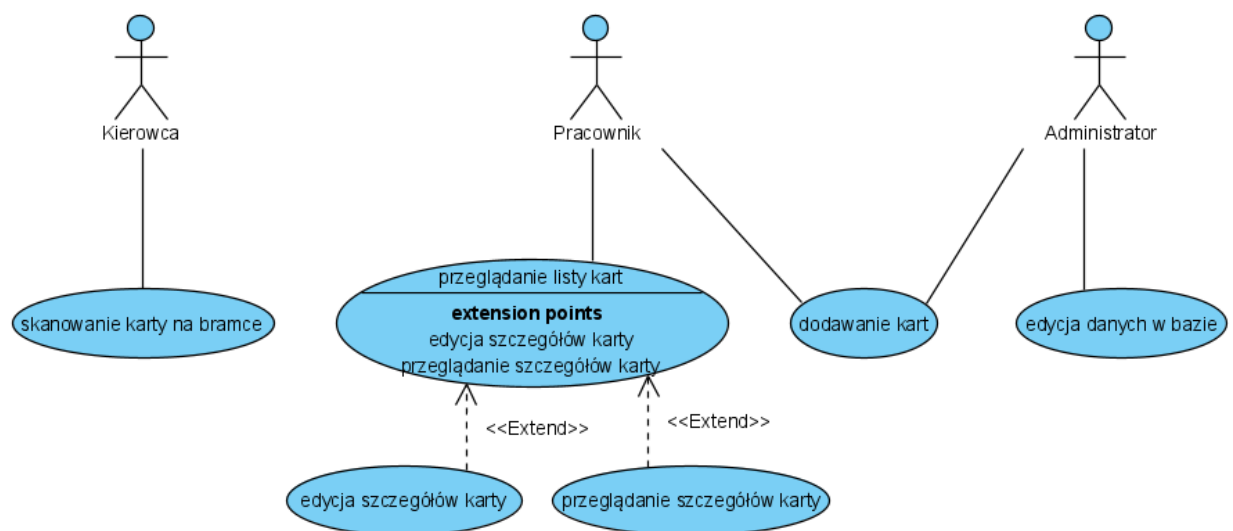
1. Aplikacja webowa działająca na przeglądarkach Google Chrome, Mozilla Firefox, Microsoft Edge, Safari.
2. Aplikacja webowa działająca na systemach windows (11, 10, 8, 7) i linux (przynajmniej ubuntu, debian, redhat).
3. Obsługa 24/7.
4. Możliwość rozszerzenia systemu o następne urządzenia: szlabany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

3. Użytkownicy systemu

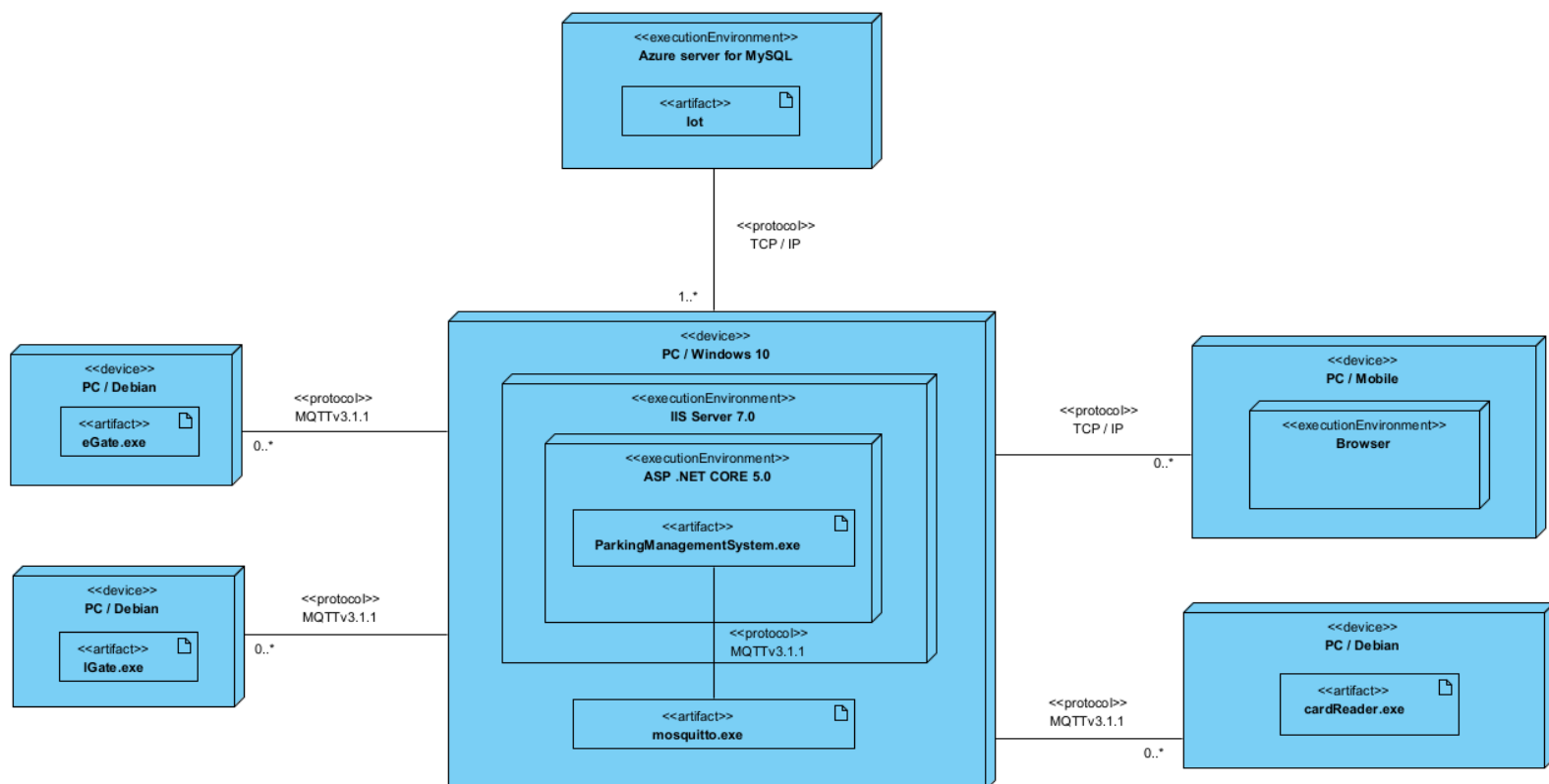
W niniejszym rozdziale zdefiniowano grupy użytkowników, korzystających z projektowanego systemu:

- 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.

4. Diagram Przypadków Użycia



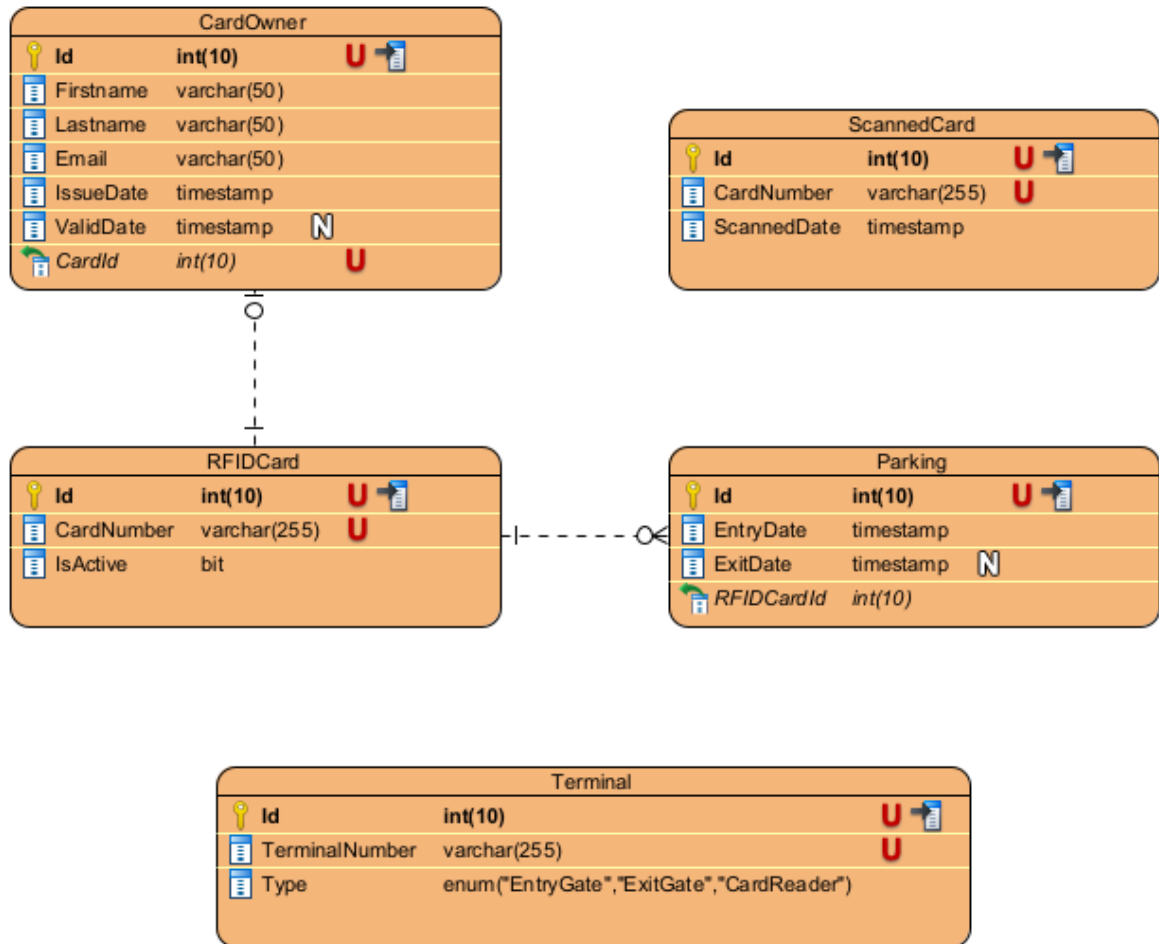
4. Schemat architektury systemu



1 Schemat rozmieszczenia

Schemat bazy danych

W ramach projektu zaprojektowano i zaimplementowano relacyjną bazę danych przechowującą 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.



2 Schemat bazy danych

5. Opis implementacji i zastosowanych rozwiązań

5.1. Front-end

Front-end aplikacji został zaimplementowany w technologii MVC (Model-View-Controller) .NET 5.0. Modele oraz kontrolery są napisane w języku C#, natomiast widoki są zaimplementowane w technologii Razor pages.

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 umożliwiającą komunikację z brokerem.

Server:

- framework ASP .NET Core 5.0
- framework Entity Framework Core
- baza danych w technologii MySQL działająca w usłudze Azure
- 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 kilkakrotnie polecenie otwarcia po zeskanowaniu tej samej karty.

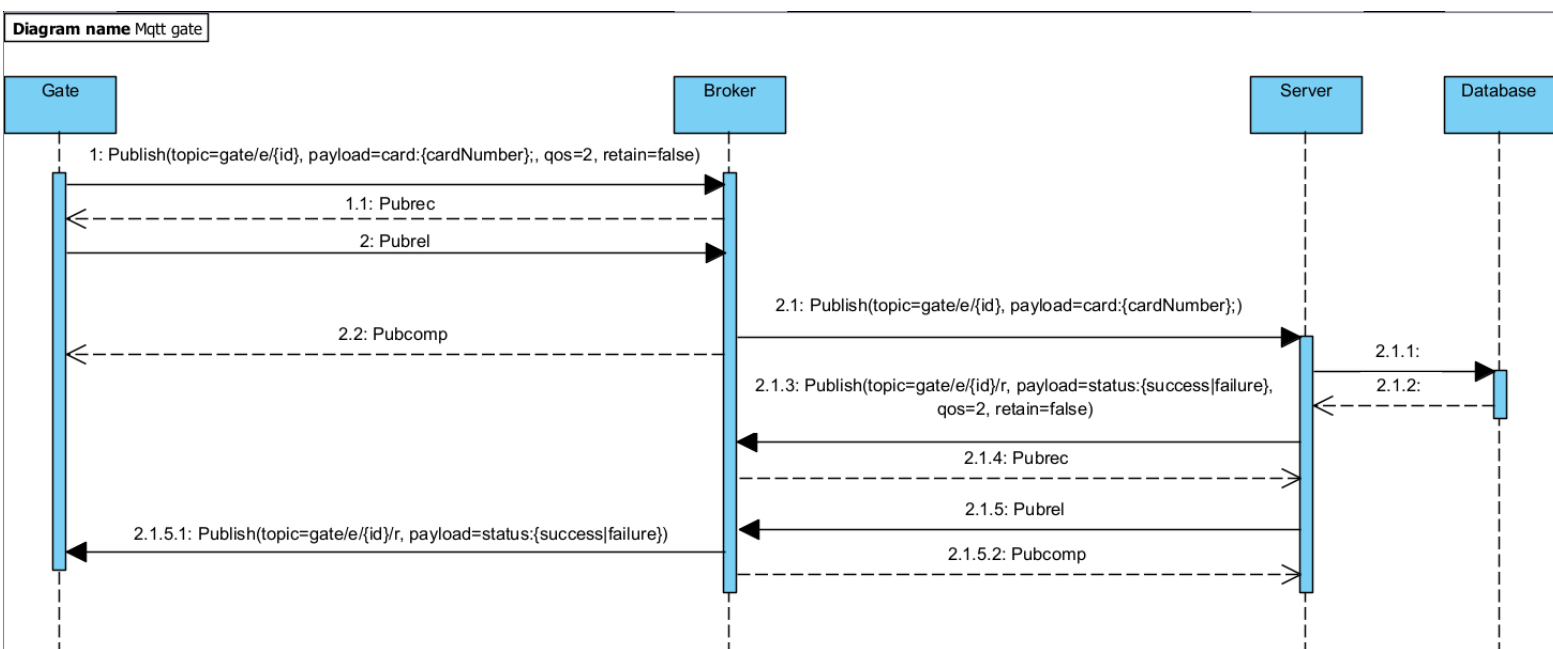
Użyta jest również flaga `retain=false` żeby broker nie przysyłał wiadomości jeśli klienci byli rozłączeni. Np. jeśli szlaban utraci połączenie z siecią i odnowi po 5 minutach, nie powinna do niego przyjść wiadomość sprzed 5 minut nakazująca otwarcie.

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.



3 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 certyfikacji.

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 poprzez 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 zasadach:

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

```
61 var optionBuilder = new MqttClientOptionsBuilder();
62 optionBuilder
63     .WithCredentials(clientSettings.UserName, clientSettings.Password)
64     .WithClientId(clientSettings.Id)
65     .WithCleanSession(true)
66     .WithKeepAlivePeriod(new System.TimeSpan(0, 0, 30))
67     .WithTcpServer(brokerHostSettings.Host, brokerHostSettings.Port);
68
69 optionBuilder.WithTls(new MqttClientOptionsBuilderTlsParameters()
70 {
71     UseTls = true,
72     SslProtocol = System.Security.Authentication.SslProtocols.Tls12,
73     Certificates = new List<X509Certificate>()
74     {
75         new X509Certificate2(clientSettings.CertFile, clientSettings.CertPassword)
76     },
77 });
```

4 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:

```

 9      "BrokerHostSettings": {
10          "Host": "localhost",
11          "Port": 8883,
12          "CaCertFile": "mqtt_ca.crt"
13      },
14      "ClientSettings": {
15          "Id": "server5eb020f043ba8930506acbddserver",
16          "UserName": "server5434783",
17          "Password": "n(n7F8e75~CQVKg8:3Rt`YLDz}d/HR",
18          "CertFile": "client_certificate.pfx",
19          "CertPassword": "1234"
20      },

```

5 Plik konfiguracyjny serwera

```

96      services.AddSingleton<IMqttClientService, MqttClientService>();
97      services.AddSingleton<IHostedService>(serviceProvider =>
98      {
99          return serviceProvider.GetService<IMqttClientService>();
100      });

```

6 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

```
113 public async Task HandleApplicationMessageReceivedAsync(MqttApplicationMessageReceivedEventArgs eventArgs)
114 {
115     string messageTopic = eventArgs.ApplicationMessage.Topic;
116     Console.WriteLine($"Received message topic: {messageTopic}");
117     string messageType = messageTopic.Substring(0, messageTopic.LastIndexOf('/') + 1);
118     Console.WriteLine(messageType);
119     string messagePayload = Encoding.UTF8.GetString(eventArgs.ApplicationMessage.Payload);
120
121     switch (messageType)
122     {
123         case EntryGatesTopic:
124             await HandleEntryGateMessageReceivedAsync(messageTopic, messagePayload);
125             break;
126         case LeaveGatesTopic:
127             await HandleLeaveGateMessageReceivedAsync(messageTopic, messagePayload);
128             break;
129         case CardReaderTopic:
130             await HandleCardReaderMessageReceivedAsync(messageTopic, messagePayload);
131             break;
132         default:
133             Console.WriteLine($"Unhandled message topic: {messageTopic}");
134             break;
135     }
136 }
```

7 Obsługa przychodzących wiadomości - ogólnie

Metoda odpowiedzialna za obsługę szlabanów wjazdowych odczytuje identyfikator 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

```
57 private string GetCardId(string payload, string parameter = "card")
58 {
59     string[] cardParameter = payload.Split(':', ';');
60
61     return cardParameter[Array.IndexOf(cardParameter, parameter) + 1];
62 }
63
64 private async Task HandleEntryGateMessageReceivedAsync(string messageTopic, string messagePayload)
65 {
66     string clientId = messageTopic.Substring(messageTopic.LastIndexOf('/') + 1);
67     string cardNumber = GetCardId(messagePayload);
68     Console.WriteLine($"Received card RFID: {cardNumber}");
69     using (var scope = _scopeFactory.CreateScope())
70     {
71         var db = scope.ServiceProvider.GetRequiredService<DatabaseContext>();
72         DbResponse message = await db.CheckEntry(clientId, cardNumber);
73         Console.WriteLine($"Open gate: {message}");
74         await SendGateResponse(messageTopic, message);
75     }
```

8 Obsługa wiadomości od szlabanów wjazdowych

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

```
93 client = mqtt.Client(client_id, clean_session=True, protocol=MQTTv311, transport="tcp")
94 client.username_pw_set(username, password)
95 client.tls_set(ca_certs=caCrt, certfile=clientCrt, keyfile=clientKey, tls_version=ssl.PROTOCOL_SSLv23,
96               ciphers=None, keyfile_password=keyPassword, cert_reqs=ssl.CERT_NONE)
97 client.connect(broker, port)
98 client.on_message = process_message
99 client.loop_start()
100 channel_ret = topic + "/r"
101 client.subscribe(channel_ret)
```

9 Kod ustawiający komunikację mqtt u klienta

Tworzony jest klient o podanym id, nazwie użytkownika i hasle. 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 - certyfikat 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)

```
1 broker = "localhost"
2 port = 8883
3 topic = "reader"
4 client_id = "2866709a-70da-11ec-90d6-0242ac120003"
5 username = "reader1"
6 password = "VF6v=)N[X8%)BZ>R&FLR[;j&j8)s*"
7 caCrt = "mqtt_ca.crt"
8 clientCrt = "mqtt_reader.crt"
9 clientKey = "mqtt_reader.key"
10 keyPassword = "1234"
```

10 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.

```
903 # Main
904 listener 8883 0.0.0.0
905 # tls_version tlsv1.2 - defaults to 1.3 and 1.2 accordingly
906
907 # Security
908 allow_anonymous false
909 require_certificate true
910
911 acl_file .\acl_list.txt
912 password_file .\password.pwd
913 cafile .\certs\mqtt_ca.crt
914 certfile .\certs\mqtt_server.crt
915 keyfile .\certs\mqtt_server.key
```

11 Ustawienia zabezpieczeń brokera

```
1 pattern write gate/e/%c
2 pattern write gate/l/%c
3 pattern write reader/%c
4 pattern read gate/e/%c/r
5 pattern read gate/l/%c/r
6
7 user server5434783
8 topic read gate/e/+
9 topic read gate/l/+
10 topic read reader/+
11 topic write gate/e+/r
12 topic write gate/l+/r
```

12 Plik acl określający dostępne tematy

```

917 # Logging
918 # log_dest file .\broker.log
919 connection_messages true
920 log_timestamp true
921 log_timestamp_format %Y-%m-%dT%H:%M:%S
922
923 # Other
924 allow_zero_length_clientid false
925 use_identity_as_username false
926 use_subject_as_username false
927 retain_available false
928 persistence false
929 queue_qos0_messages false

```

13 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.

TCP	44	56598 → 8883 [ACK] Seq=1 Ack=1 Win=8192 Len=0
TLSv1.2	215	Client Hello
TCP	44	8883 → 56598 [ACK] Seq=1 Ack=172 Win=2619648 Len=0
TLSv1.2	2303	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
TCP	44	56598 → 8883 [ACK] Seq=172 Ack=2260 Win=5933 Len=0
TLSv1.2	1282	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Message
TCP	44	8883 → 56598 [ACK] Seq=2260 Ack=1410 Win=2618368 Len=0
TLSv1.2	1166	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
TCP	44	56598 → 8883 [ACK] Seq=1410 Ack=3382 Win=4811 Len=0
TLSv1.2	170	Application Data
TCP	44	8883 → 56598 [ACK] Seq=3382 Ack=1536 Win=2618368 Len=0

14 Połączenie serwera z brokerem

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

6955	437.251170	127.0.0.1	127.0.0.1	TLSv1.2	170 Application Data
6956	437.251195	127.0.0.1	127.0.0.1	TCP	44 8883 → 56598 [ACK] Seq=3382 Ack=1536 Win=2618368 Len=0


```

> Frame 6955: 170 bytes on wire (1360 bits), 170 bytes captured (1360 bits) on interface \Device\NPF_{...}, id 0
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 56598, Dst Port: 8883, Seq: 1410, Ack: 3382, Len: 126
▼ Transport Layer Security
  ▼ TLSv1.2 Record Layer: Application Data Protocol: mqtt
    Content Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 121
    Encrypted Application Data: 0000000000000000189ebfee83df2784a236a69cd43f02ef65949d5fecfc3429218b53286...
    [Application Data Protocol: mqtt]

```



```

0000 02 00 00 00 45 00 00 a6 ec c1 40 00 80 06 00 00
0010 7f 00 00 01 7f 00 00 01 dd 16 22 b3 ed bd 25 fb
0020 df 32 0b 87 50 18 12 cb 89 bd 00 00 17 03 03 00
0030 79 00 00 00 00 00 00 00 01 89 eb fe e8 3d f2 78
0040 4a 23 6a 69 cd 43 f0 2e f6 59 49 d5 fe cf c3 42
0050 92 18 b5 32 86 dd e9 e7 4d 10 e3 1d 6c 24 8e 33
0060 07 28 2c 06 4a dd 09 17 0c de 07 0c ea fa bc 9a
0070 39 2e e6 90 20 a4 ff b8 1e fb 7e db 94 14 cb 5a
0080 0f 79 60 ed 32 d7 0e 23 5c cb ab 9c f5 4a 72 01
0090 44 39 e3 fb 5e ff 31 00 ce de 7a 8e 50 c1 5a c5
00a0 7c 58 cf 43 64 f5 a6 b3 be e4

```

15 Dane z połączenia serwera z brokerem

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

```

TCP      44 56605 → 8883 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
TLsv1.3 561 Client Hello
TCP      44 8883 → 56605 [ACK] Seq=1 Ack=518 Win=2619648 Len=0
TLsv1.3 2470 Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data
TCP      44 56605 → 8883 [ACK] Seq=518 Ack=2427 Win=2617344 Len=0
TLsv1.3 2245 Change Cipher Spec, Application Data, Application Data, Application Data
TCP      44 8883 → 56605 [ACK] Seq=2427 Ack=2719 Win=2617344 Len=0
TLsv1.3 160 Application Data
TCP      44 8883 → 56605 [ACK] Seq=2427 Ack=2835 Win=2617344 Len=0

```

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”).

7282	454.502026	127.0.0.1	127.0.0.1	TLsv1.3	125 Application Data
7283	454.502038	127.0.0.1	127.0.0.1	TCP	44 8883 → 56605 [ACK] Seq=4718 Ack=2990 Win=2617088 Len=0
7284	454.502804	127.0.0.1	127.0.0.1	TLsv1.3	70 Application Data
7285	454.502826	127.0.0.1	127.0.0.1	TCP	44 56605 → 8883 [ACK] Seq=2990 Ack=4744 Win=2615040 Len=0
7288	454.507974	127.0.0.1	127.0.0.1	TLsv1.3	70 Application Data
7289	454.508005	127.0.0.1	127.0.0.1	TCP	44 8883 → 56605 [ACK] Seq=4744 Ack=3016 Win=2617088 Len=0
7290	454.508481	127.0.0.1	127.0.0.1	TLsv1.2	130 Application Data
7291	454.508508	127.0.0.1	127.0.0.1	TCP	44 56598 → 8883 [ACK] Seq=1668 Ack=3603 Win=4590 Len=0
7292	454.508599	127.0.0.1	127.0.0.1	TLsv1.3	70 Application Data
7293	454.508611	127.0.0.1	127.0.0.1	TCP	44 56605 → 8883 [ACK] Seq=3016 Ack=4770 Win=2614784 Len=0
7446	463.293257	127.0.0.1	127.0.0.1	TLsv1.3	126 Application Data

```

<
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 8883, Dst Port: 56598, Seq: 3517, Ack: 1668, Len: 86
▼ Transport Layer Security
  ▼ TLsv1.2 Record Layer: Application Data Protocol: mqtt
    Content Type: Application Data (23)
    Version: TLS 1.2 (0x0303)
    Length: 81
    Encrypted Application Data: 2f351d292d82fdb6fee53fc33ea0cd35abb73684e70e912783b00f157419ad497434ddc3...
    [Application Data Protocol: mqtt]

```

```





0000 02 00 00 00 45 00 00 7e ee 03 40 00 80 06 00 00
0010 7f 00 00 01 7f 00 00 01 22 b3 dd 16 df 32 0c 0e
0020 ed hd 76 fd 50 18 27 f3 0e 0b 00 00 17 03 03 00
0030 51 2f 35 1d 29 2d 82 fd b6 fe e5 3f c3 3e a0 cd
0040 35 ah b7 36 84 e7 0e 91 27 83 b0 0f 15 74 19 ad
0050 49 74 34 dd c3 3f 16 9f 29 53 43 79 2c 7b 29 5h
0060 82 a3 da d7 7b be a5 ef ef 81 e7 4c 46 40 dc e2
0070 ah ha 98 b6 eb 76 29 ac 97 0d 1d 0b 49 e9 d4 c8
0080 e7 8e

```

17 Dane przesłane między klientem, a serwerem

Opis implementacji bazy danych

Baza danych została wygenerowana za pomocą Entity Framework Core na podstawie modeli utworzonych w ramach wzorca MVC. W celu poprawnego działania wykorzystanego frameworku konieczne jest dołączenie do projektu wymaganych pakietów NuGet.

	Microsoft.EntityFrameworkCore.SqlServer by Microsoft	5.0.12
	Microsoft SQL Server database provider for Entity Framework Core.	6.0.1
	Microsoft.EntityFrameworkCore.Tools by Microsoft	5.0.12
	Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.	6.0.1
	Microsoft.VisualStudio.Web.CodeGeneration.Design by Microsoft	5.0.2
	Code Generation tool for ASP.NET Core. Contains the dotnet-aspnet-codegenerator command used for generating controllers and views.	6.0.1
	Pomelo.EntityFrameworkCore.MySql by Laurents Meyer, Caleb Lloyd, Yuko Zheng	5.0.3
	Pomelo's MySQL database provider for Entity Framework Core.	6.0.0

18 Manager pakietów NuGet

Konfiguracja tego frameworku, przede wszystkim wskazanie połączenia do bazy danych, znajduje się w pliku Startup.cs

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<DatabaseContext>(options =>
        options.UseMySQL(
            Configuration.GetConnectionString("DatabaseContext"),
            ServerVersion.AutoDetect(Configuration.GetConnectionString("DatabaseContext"))));
}
```

19 Fragment metody ConfigureServices odpowiedzialny za skonfigurowanie połączenia z bazą danych

ConnectionString znajduje się w pliku konfiguracyjnym appsettings.json

```
{
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "DatabaseContext": "Server=iot.mysql.database.azure.com; User ID=parking; Password=uD13wj?<aEA]nsZJyY#UFS@[Xc[q"; Database=iot; SslMode=Required"
  }
}
```

20 ConnectionString w pliku appsettings.json

Komunikacja z bazą danych jest odbywa się z pomocą klasy DatabaseContext. Odpowiada ona także za zadeklarowanie kolekcji DbSet<TEntity>, które są używane w celu pobrania danych wybranych tabel z bazy danych. Każda encja z bazy danych ma odpowiadający model. W bazie danych wykorzystywanej przez aplikację znajdują się tabele:

- RFIDCards
- CardOwners
- Parkings
- ScannedCards
- Terminals

```

public class DatabaseContext : DbContext
{
    0 references
    public DatabaseContext(DbContextOptions<DatabaseContext> options) : base(options)
    {
    }

    0 references
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<RFIDCard>()
            .HasOne(s => s.CardOwner)
            .WithOne(s => s.RFIDCard)
            .HasForeignKey<CardOwner>(s => s.CardId);
    }

    14 references
    public DbSet<RFIDCard> RFIDCards { get; set; }

    0 references
    public DbSet<CardOwner> CardOwners { get; set; }

    7 references
    public DbSet<Parking> Parkings { get; set; }

    3 references
    public DbSet<ScannedCard> ScannedCards { get; set; }

    9 references
    public DbSet<Terminal> Terminals { get; set; }
}

```

21 Klasa DatabaseContext

6. Opis działania i prezentacja interfejsu

System kontroli parkingu Karty Wjazdy i wyjazdy Terminale

Karty

Dodaj nową

Wyszukaj

x

Nr karty	Aktywna	Imię	Nazwisko	Email	Data wydania	Data ważności	
122134	Tak	Maciek	Kopinski	kopin@gmail.com	09.01.2022 11:01:00	12.01.2022 01:11:00	Edytuj Detale Usuń
7688	Tak	Piotr	Kołpa	kolpaaa@gmail.com	10.01.2022 19:11:26	09.02.2022 19:11:26	Edytuj Detale Usuń
445322	Tak	Bartek	Jagiello	barti@wp.pl	10.01.2022 19:11:43	09.02.2022 19:11:43	Edytuj Detale Usuń
2	Tak						Edytuj Detale Usuń
696969	Nie						Edytuj Detale Usuń
99898	Tak						Edytuj Detale Usuń
0000	Nie						Edytuj Detale Usuń
666	Tak						Edytuj Detale Usuń

© 2022 - System kontroli parkingu

22 Ekran główny kart

System kontroli parkingu Karty Wjazdy i wyjazdy Terminale

Dodawanie karty

Numer karty

Zeskanowane karty

☐ Aktywna

☒ Ma właściciela

Imię

Nazwisko

Email

Data wydania

17.01.2022, 14:58:25.315

x

Data ważności

16.02.2022, 14:58:25.315

x

Utwórz

[Powrót](#)

© 2022 - System kontroli parkingu

23 Dodawanie nowej karty

Detale karty

Numer karty	122134
Aktywna	Tak
Imię	Maciek
Nazwisko	Kopinski
Email	kopin@gmail.com
Data wydania	09.01.2022 11:01:00
Data ważności	12.01.2022 01:11:00

[Edytuj](#) | [Powrót](#)

24 Wyświetlanie detali karty

Edycja karty

Numer karty

122134

[Zeskanowane karty](#)

☒ Aktywna

☒ Ma właściciela

Imię

Maciek

Nazwisko

Kopinski

Email

kopin@gmail.com

Data wydania

09.01.2022, 11:01

Data ważności

12.01.2022, 01:11

[Edytuj](#)

[Powrót](#)

25 Edycja danych karty

Wjazdy i wyjazdy

EntryDate	ExitDate	RFIDCard	
08.01.2022 19:53:58	08.01.2022 19:56:43	2	Edit Details Delete
10.01.2022 15:16:10		2	Edit Details Delete
10.01.2022 19:35:50		666	Edit Details Delete

26 Ewidencja wjazdów i wyjazdów

Interfejs użytkownika służy do zarządzania kartami, ich właścicielami oraz daje możliwość podglądu logów wjazdów i wyjazdów. Główny ekran kart (rys 18) pozwala na podgląd wszystkich kart oraz ich użytkowników. Posiada on również możliwość wyszukiwania osób i kart. Z tego widoku użytkownik ma możliwość przejścia do ekranów dodawania nowych kart, edycji, detali lub usuwania karty.

Drugim ekranem jest widok Wjazdów i wyjazdów (rys 22), ekran ten umożliwia analizę logów wjazdów i wyjazdów z parkingu.

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:

- projekt i implementacja interfejsu użytkownika w technologii MVC
- utworzenie, podłączenie i skonfigurowanie szyfrowania połączenia do aplikacji serwera bazy danych
- dokumentacja opisu działania i prezentacji interfejsu
- utworzenie i zarządzanie repozytorium z kodem projektu na platformie github

Piotr Kołpa:

- projekt bazy danych
- dokumentacja diagramu ERD bazy danych w Visual Paradigm
- implementacja modeli MVC
- implementacja bazy danych za pomocą Entity Framework
- 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

8. Podsumowanie

9. Literatura

1. [Dokumentacja mqtt w .net](#)
2. [Zabezpieczanie protokołu mqtt](#)
3. [Ustawianie ACL dla mqtt](#)
4. [Dokumentacja mosquitto dla .conf](#)

10. Aneks