

System otwierania drzwi przez gości hotelu kwarantannowego wykorzystujący technologię RFID

Przedmiot: **Podstawy Internetu Rzeczy
Laboratorium**

Imiona, nazwiska i numery indeksów autorów:

Bogusława Tiołka 254505

Aleksandra Stecka 254584

Paweł Walkowiak 254513

Joanna Wdziękońska 254515

Semestr studiów: 5

Data ukończenia pracy: 31.01.2022

Prowadzący laboratorium: dr inż Krzysztof Chudzik

Spis treści

Wymagania projektowe	3
Cel / Sformułowanie problemu	3
Narzędzia	3
Wymagania funkcjonalne i нефункционалне	4
Mockupy interfejsu	5
Opis architektury systemu	9
Opis implementacji i zastosowanych rozwiązań	10
Docker	10
Baza danych	11
Struktura logiczna relacyjnej bazy danych	11
Skrypt generujący bazę danych	12
Skrypt generujący dane testowe do bazy danych	13
Broker MQTT	15
Podstawowa konfiguracja brokera	15
Szyfrowanie TLS/SSL	16
Uwierzytelnianie TLS/SSL	18
API	20
Opis działania i prezentacja interfejsu	22
Opis sposobu instalacji i uruchomienia	22
Instalacja i uruchomienie aplikacji mobilnej	22
Instalacja i uruchomienie serwera aplikacji	28
Aplikacja mobilna	35
Aplikacja webowa	40
Opis wkładu pracy każdego z autorów	43
Bogusława Tłółka 254505	43
Aleksandra Stecka 254584	43
Paweł Walkowiak 254513	43
Joanna Wdziękońska 254515	44
Podsumowanie	45
Ocena zgodności projektu z wymaganiami	45

Uwagi dotyczące napotkanych trudności	45
Propozycja zmian lub rozbudowy projektu	46
Literatura	46
Aneks	46

Wymagania projektowe

Cel / Sformułowanie problemu

Celem projektu jest stworzenie systemu umożliwiającego umożliwiającego gościom hotelu kwarantannowego automatyczne otwieranie drzwi do ich pokoi oraz głównej bramy hotelu za pomocą kart RFID zamiast kluczy.

Gość melduje się w hotelu za pośrednictwem aplikacji mobilnej, aby ograniczyć kontakt z osobami na recepcji. Po zameldowaniu się w hotelu każdy gość zamiast klucza do swojego pokoju otrzymuje kartę RFID. Ta karta jest przypisana do tego gościa przez cały jego pobyt w hotelu i umożliwia mu otwarcie drzwi do jego pokoju. Pracownik przekazuje gościowi token do wpisania do aplikacji – token służy do powiązania gościa z pokojem, w którym będzie się zatrzymywał, i kartą RFID, której będzie używał do otwarcia drzwi do swojego pokoju. Każdy token jest ważny przez co najwyżej 24h od utworzenia oraz jest jednorazowy – po wykorzystaniu przez jednego gościa przestaje być ważny.

Podczas wymeldowywania się z hotelu gość jest zobowiązany zdać kartę RFID, która następnie może być wykorzystana przez innych gości. Nie zdanie karty RFID jest związane z koniecznością opłacenia kaucji w wysokości 50zł. W przypadku zgubienia karty, gość jest zobowiązany zablokować kartę, żeby nie mogła zostać ona wykorzystana przez osoby nieuprawnione.

Jeżeli gość, który dalej jest na kwarantannie, użyje swojej karty aby otworzyć główną bramę hotelu, nie zostanie on wypuszczony. Jedynie goście, których kwarantanna się skończyła, zostaną przepuszczeni przez bramkę. Za bramką znajduje się recepcja, przy której gość wymeldowuje się z hotelu i zdejmuje swoją kartę pracownikowi.

Narzędzia

A. Baza danych – MariaDB

Dane użytkowników systemu wiążące ich dane osobowe z numerem karty RFID, która umożliwia otwarcie drzwi pokoju, będą przechowywane w bazie danych z wykorzystaniem systemu bazodanowego MariaDB (<https://mariadb.com/>).

Użycie w języku Python z wykorzystaniem modułu mariadb (<https://pypi.org/project/mariadb/>).

- B. Maszyna wirtualna symulująca moduł czytnika kart zbliżeniowych RFID
Z uwagi na wprowadzenie zajęć zdalnych niemożliwe jest wykorzystanie rzeczywistego modułu czytnika karty zbliżeniowych. Przed przejściem na zajęcia zdalne zakładano wykorzystanie modułu występującego w zestawach laboratoryjnych Raspberry Pi. Odczyt kart RFID zostanie zasymulowany za pomocą maszyny wirtualnej.
- C. Protokół MQTT
Lekki protokół transmisji danych, oparty na wzorcu publikacja/subskrypcja. Ze względu na swoją charakterystykę, nadaje do komunikacji z urządzeniami internetu rzeczy. Możliwość wykorzystania w języku python udostępnia moduł paho-mqtt (<https://pypi.org/project/paho-mqtt/>).

Wymagania funkcjonalne i нефункционалне

- A. Funkcjonalne
- Gość hotelowy melduje się w hotelu za pomocą aplikacji mobilnej.
 - Gość hotelowy otrzymuje od pracownika hotelu na recepcji kartę RFID. Karta RFID umożliwia gościowi otwarcie jednego konkretnego pokoju. Za pomocą aplikacji mobilnej gość wprowadza token, otrzymany od pracownika hotelu, dzięki któremu tworzone jest powiązanie gościa z pokojem, w którym będzie się zatrzymywał i kartą, którą będzie się posługiwał. Do otworzenia drzwi do tego pokoju gość posługuje się kartą RFID.
 - Gość hotelowy wymeldowuje się z hotelu za pomocą aplikacji mobilnej. Wymeldowanie się z hotelu oznacza zapisanie końcowej daty ważności przypisania karty RFID do gościa. Gość jest zobowiązany wymeldować się z hotelu i zdać kartę RFID przy recepcji i w obecności pracownika hotelu, przed opuszczeniem hotelu.
 - Gość hotelowy blokuje kartę RFID na wypadek zgubienia lub ukradnięcia karty za pomocą aplikacji mobilnej.
 - Pracownik administracyjny ma możliwość przeglądania i wyszukiwania aktualnych gości hotelu za pomocą aplikacji webowej.
 - Pracownik administracyjny dodaje nowe karty RFID za pomocą aplikacji webowej.
 - Pracownik administracyjny tworzy nowy token za pomocą aplikacji webowej.

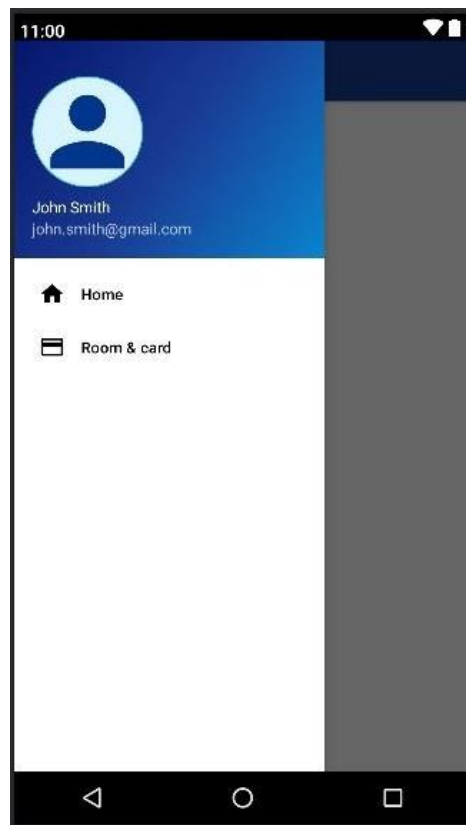
B. Niefunkcjonalne

- a. Połączenie urządzenia z czujnikiem RFID z serwerem aplikacji jest szyfrowane.
- b. Aplikacje są dostępne w systemie 24/7/365.
- c. Aplikacja webowa działa na najpopularniejszych przeglądarkach internetowych takich jak Google Chrome, Mozilla Firefox, Microsoft Edge.
- d. Aplikacja mobilna działa na urządzeniach z systemem Android.
- e. Architektura systemu jest oparta o model warstwowy MVVM.
- f. Aplikacje odpowiadają na zapytania użytkownika w czasie do 2 sekund.
- g. Aplikacje są dostępne w języku angielskim.

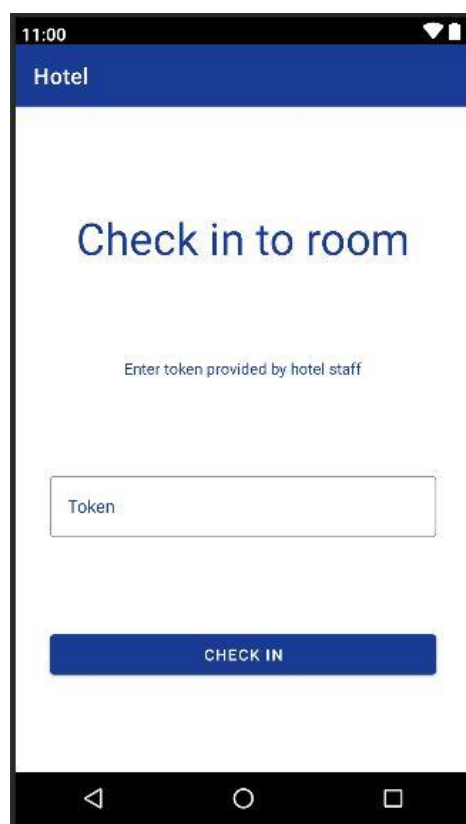
Mockupy interfejsu

A. Interfejs aplikacji mobilnej

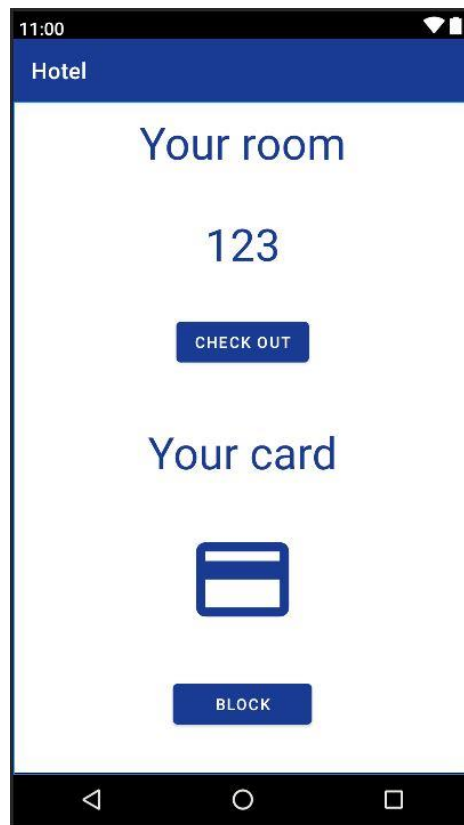
Rysunek 1, 2. Ekran wprowadzania danych użytkownika



Rysunek 3. Ekran nawigacji

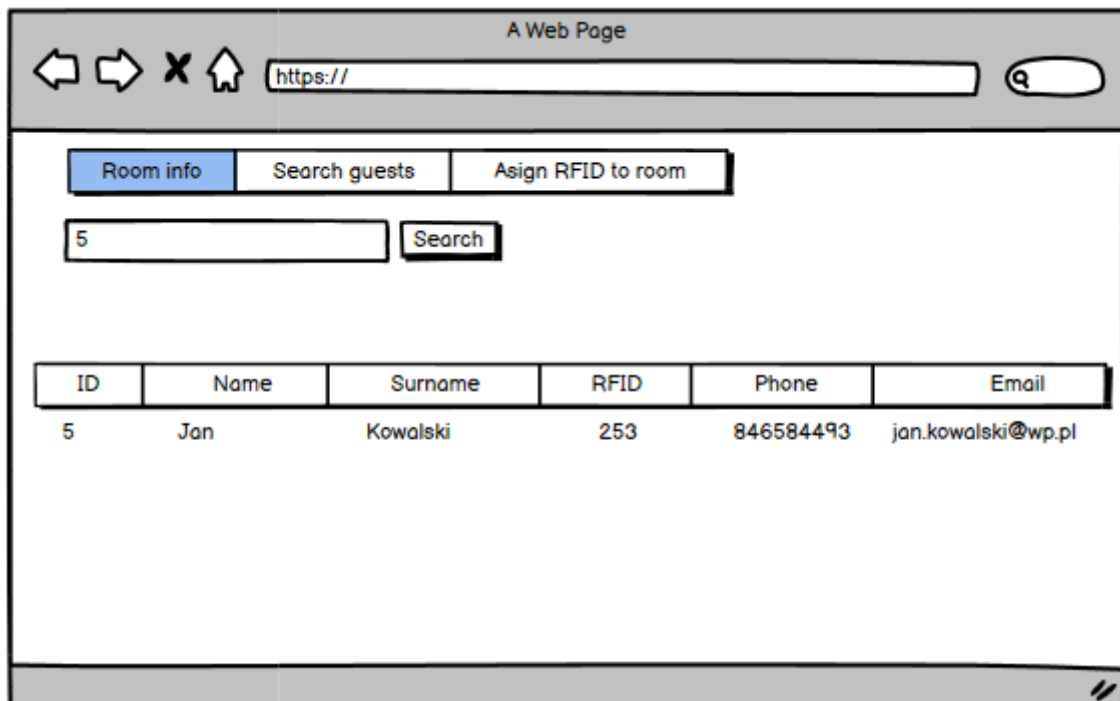


Rysunek 4. Ekran wprowadzenia tokena, którego gość otrzymuje od pracownika hotelu



Rysunek 5. Ekran informujący gościa o numerze jego pokoju oraz pozwalający zablokować kartę

B. Interfejs aplikacji webowej



Rysunek 6. Wyszukiwanie aktualnego gościa w pokoju – wątek główny

A Web Page

https://

Room info Search guests Assign RFID to room

3 Search

ID	Name	Surname	RFID	Phone	Email
No matching records found					

Rysunek 7. Wyszukiwanie aktualnego gościa w pokoju i numeru karty RFID – wątek poboczny (pokój nie jest zamieszkiwany przez żadnego gościa)

A Web Page

https://

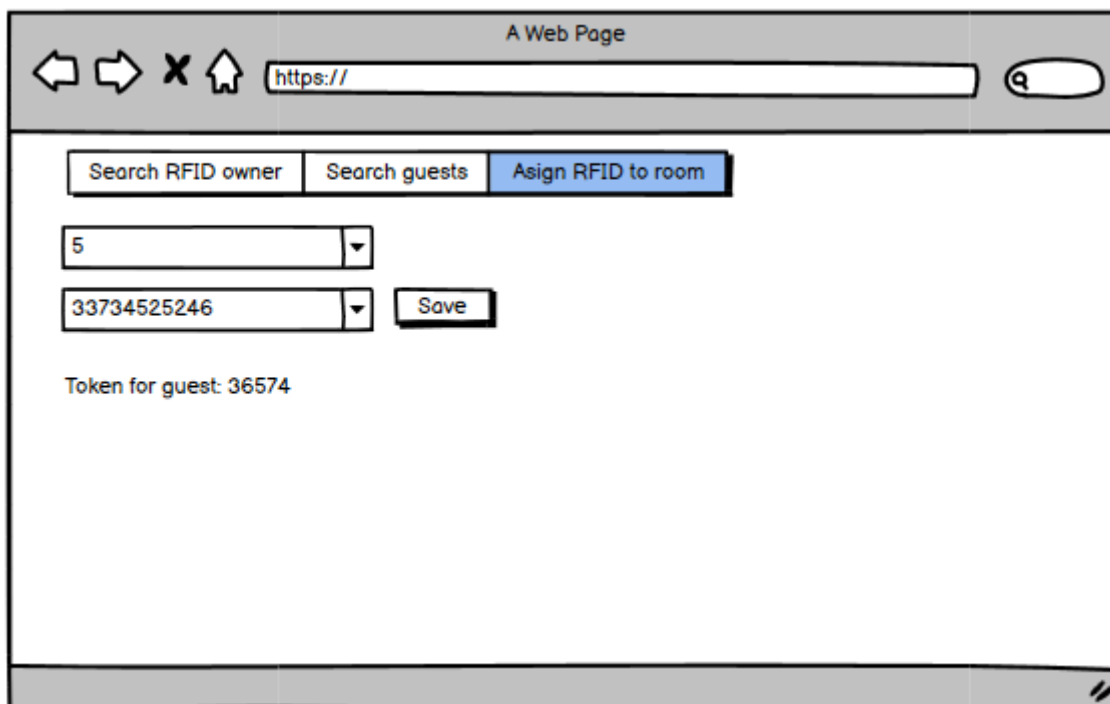
Search RFID owner Search guests Assign RFID to room

Enter guest surname:

Kowalski Search

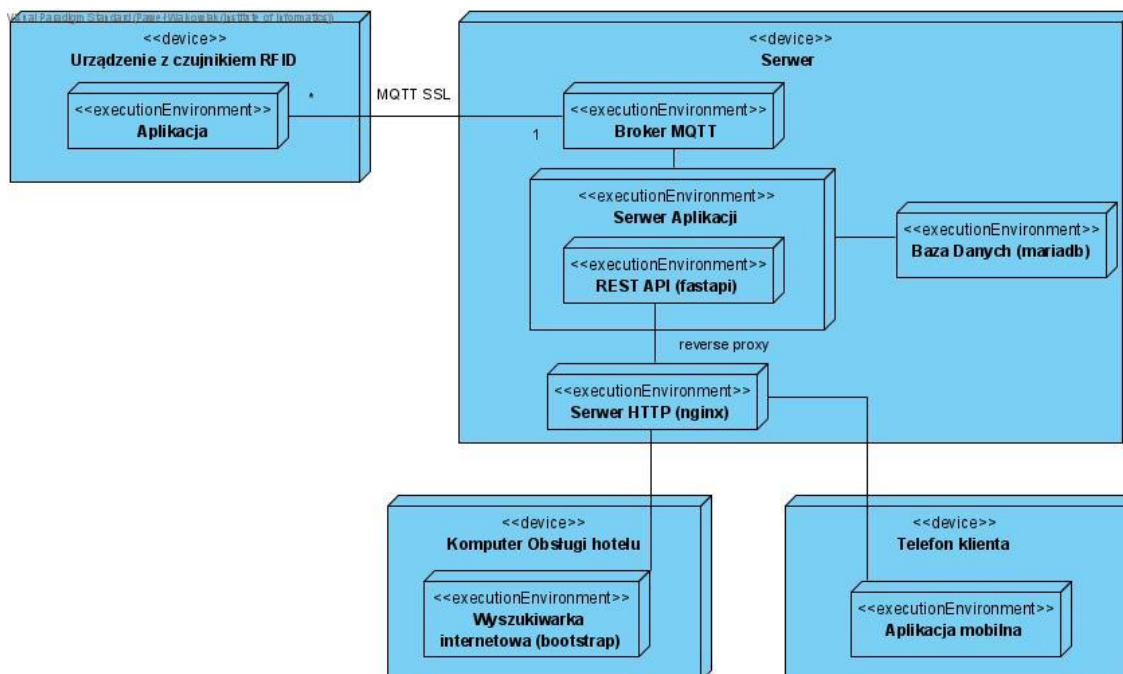
ID	Name	Surname	RFID	Phone	Email
5	Jan	Kowalski	253	846584493	jan.kowalski@wp.pl
8	Roman	Kowalski	523	274538364	r.kowalski@wp.pl

Rysunek 8. Wyszukiwanie aktualnych gości hotelu po nazwisku



Rysunek 9. Przypisanie karty RFID do pokoju hotelowego

Opis architektury systemu



Rysunek 10. Diagram rozmieszczenia przedstawiający architekturę systemu

System składa się z głównej części serwera, którego składowymi są broker mqtt, baza danych, serwer http oraz serwer aplikacji zapewniający komunikację oraz przetwarzanie danych pomiędzy pod węzłami.

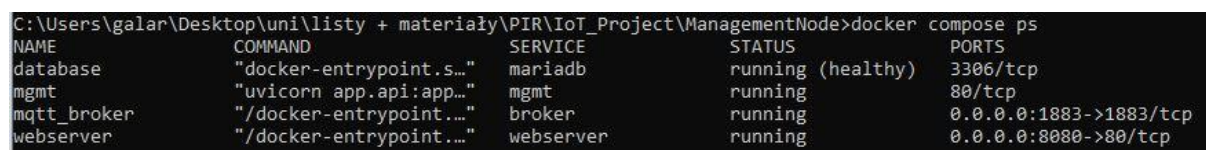
Komunikuje się on z brokerem za pomocą klienta mqtt, z bazą danych za pomocą klienta bazy danych oraz udostępnia REST API do komunikacji aplikacją mobilną i serwerem.

W systemie znajdują się również bramka z czujnikiem RFID, która łączy się z brokerem mqtt, komputer kliencki który korzysta ze stron znajdujących się na serwerze http, oraz aplikacja mobilna komunikująca się z serwerem za pomocą REST API.

Opis implementacji i zastosowanych rozwiązań

Docker

Do postawienia serwisów wykorzystano konteneryzację (Docker) z narzędziem orkiestryzacji docker-compose. Postawiono cztery serwisy – bazę danych (database), serwer aplikacji (mgmt), broker MQTT (mqtt_broker) oraz serwer webowy (webserver).



NAME	COMMAND	SERVICE	STATUS	PORTS
database	"docker-entrypoint.s..."	mariadb	running (healthy)	3306/tcp
mgmt	"uvicorn app.api:app..."	mgmt	running	80/tcp
mqtt_broker	"/docker-entrypoint..."	broker	running	0.0.0.0:1883->1883/tcp
webserver	"/docker-entrypoint..."	webserver	running	0.0.0.0:8080->80/tcp

Screen 1. Widok statusu serwisów Docker

Wykorzystano Docker z uwagi na prostą instalację i konfigurację serwisów, a także możliwość postawienia serwisów lokalnie u każdego z członków zespołu w celu testów swojej części systemu. Komunikacja z zewnątrz odbywa się przez serwer webowy (nginx), udostępniający strony statyczne (widoki administratora z tabelami bootstrap) i mapujący komunikację na adres `http://<adres_serwera>:8080/api/` do kontenera mgmt udostępniającego Rest API.

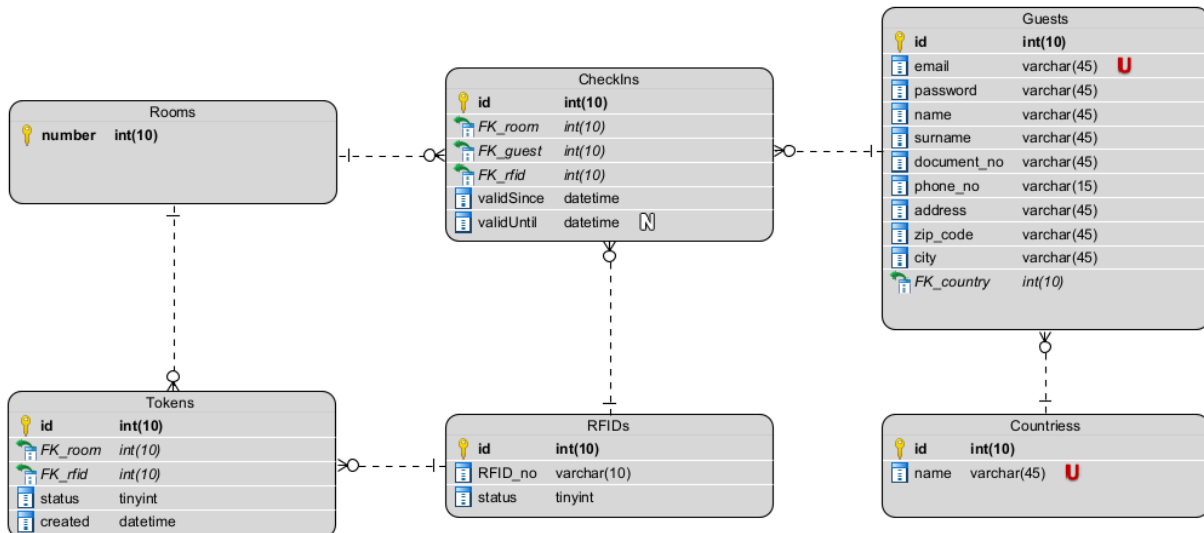
Aby uruchomić wszystkie serwisy należy przejść do folderu, w którym znajduje się `docker-compose.yml` i wywołać komendę **docker compose up -d**

Aby sprawdzić status serwisów należy wywołać komendę **docker compose ps**

Żeby zamknąć wszystkie serwisy należy wywołać komendę **docker compose down**

Baza danych

Struktura logiczna relacyjnej bazy danych



Rysunek 11. Diagram logiczny bazy danych

Tabele:

- Countries – przechowuje informacje na temat krajów pochodzenia gości hotelu. Tabela wprowadzona w celu zminimalizowania literówek i błędów w nazwach krajów,
- Guests – przechowuje informacje na temat gości hotelu,
- CheckIns – przechowuje informacje na temat powiązań gości z pokojami oraz kartami RFID. Gość zatrzymuje się w konkretnym pokoju i używa konkretnej karty do otwarcia drzwi do swojego pokoju. W momencie wymeldowania się gościa z hotelu czas i data wymeldowania zapisywane są w polu validUntil. Przed wymeldowaniem się z hotelu pole validUntil ma wartość NULL, co oznacza, że przypisanie gościa do pokoju jest aktualne,
- RFIDs – przechowuje informacje na temat kart RFID, włącznie z pełnym numerem karty RFID i jej statusem (zgubiona/skradziona lub też nie),
- Rooms – przechowuje informacje na temat pokoi w hotelu. Tabela wprowadzona w celu zminimalizowania literówek i błędów w numeracji pokoi,
- Tokens – przechowuje informacje na temat tokenów, których goście używają, aby utworzyć powiązanie pomiędzy sobą a pokojem i kartą. Każdy token jest ważny przez co najwyżej 24h od utworzenia (datę utworzenia przechowuje pole created) oraz jest jednorazowy – po wykorzystaniu przez jednego gościa przestaje być ważny (informację, czy token został już wykorzystany przechowuje pole status).

Relacje między tabelami:

- Guests – Countries to relacja N – 1 ponieważ wielu gości może pochodzić z tego samego kraju.
- Rooms – Tokens to relacja N – 1 ponieważ dany token dotyczy tylko jednego pokoju, ale może istnieć wiele tokenów dla jednego pokoju,
- RFIDs – Tokens to relacja N – 1 ponieważ dany token dotyczy tylko jednej karty RFID, ale może istnieć wiele tokenów dla jednej karty RFID,
- CheckIns – Guests to relacja N – 1 ponieważ dany gość może wynajmować wiele pokoi w różnym czasie – jeśli kilka osobnych razy będzie wynajmował pokój się w hotelu – ale dany pokój w danym czasie jest wynajęty przez tylko jednego gościa,
- CheckIns – Rooms to relacja N – 1 ponieważ dany pokój może być wynajmowany przez wielu gości w różnym czasie ale dany pokój w danym czasie jest wynajęty przez tylko jednego gościa,
- CheckIns – RFIDs to relacja N – 1 ponieważ dany pokój może być otwierany przez różne karty w różnym czasie, ale dana karta w danym czasie otwiera tylko jeden pokój.

Skrypt generujący bazę danych

```
DROP DATABASE IF EXISTS hotel;
```

```
CREATE DATABASE hotel;
```

```
USE hotel;
```

```
CREATE TABLE Guests (  
    id            int(10) NOT NULL AUTO_INCREMENT,  
    email         varchar(45) NOT NULL UNIQUE,  
    password      varchar(45) NOT NULL,  
    name          varchar(45) NOT NULL,  
    surname       varchar(45) NOT NULL,  
    document_no   varchar(45) NOT NULL,  
    phone_no      varchar(15) NOT NULL,  
    address       varchar(45) NOT NULL,  
    zip_code      varchar(45) NOT NULL,  
    city          varchar(45) NOT NULL,  
    FK_country    int(10) NOT NULL,  
    PRIMARY KEY (id));
```

```
CREATE TABLE Rooms (  
    number int(10) NOT NULL AUTO_INCREMENT,  
    PRIMARY KEY (number));
```

```
CREATE TABLE RFIDs (  
    id            int(10) NOT NULL AUTO_INCREMENT,  
    RFID_no       varchar(10) NOT NULL,  
    status        tinyint NOT NULL,
```

```

        PRIMARY KEY (id));

CREATE TABLE Countries (
    id    int(10) NOT NULL AUTO_INCREMENT,
    name  varchar(45) NOT NULL UNIQUE,
    PRIMARY KEY (id));

CREATE TABLE CheckIns (
    id            int(10) NOT NULL AUTO_INCREMENT,
    FK_room       int(10) NOT NULL,
    FK_guest      int(10) NOT NULL,
    FK_rfid       int(10) NOT NULL,
    validSince    datetime NOT NULL,
    validUntil    datetime NULL,
    PRIMARY KEY (id));

CREATE TABLE Tokens (
    id            int(10) NOT NULL AUTO_INCREMENT,
    FK_room       int(10) NOT NULL,
    FK_rfid       int(10) NOT NULL,
    status        tinyint DEFAULT 1 NOT NULL,
    created       datetime NOT NULL,
    PRIMARY KEY (id));

ALTER TABLE Guests ADD CONSTRAINT
    FKGuests400793 FOREIGN KEY (FK_country) REFERENCES Countries (id);

ALTER TABLE CheckIns ADD CONSTRAINT
    FKCheckIns811411 FOREIGN KEY (FK_guest) REFERENCES Guests (id);

ALTER TABLE CheckIns ADD CONSTRAINT
    FKCheckIns469541 FOREIGN KEY (FK_room) REFERENCES Rooms (number);

ALTER TABLE CheckIns ADD CONSTRAINT
    FKCheckIns569952 FOREIGN KEY (FK_rfid) REFERENCES RFIDs (id);

ALTER TABLE Tokens ADD CONSTRAINT
    FKTokens606278 FOREIGN KEY (FK_room) REFERENCES Rooms (number);

ALTER TABLE Tokens ADD CONSTRAINT
    FKTokens264901 FOREIGN KEY (FK_rfid) REFERENCES RFIDs (id);

```

Skrypt generujący dane testowe do bazy danych

```

USE hotel;

INSERT INTO Countries (name) VALUES
    ('Afghanistan'),
    ('Australia'),
    ('Austria'),
    ('Belgium'),
    ('Colombia'),
    ('Croatia'),

```

```

('Cuba'),
('Czech Republic'),
('Denmark'),
('Faroe Islands'),
('Germany'),
('Greenland'),
('Italy'),
('Norway'),
('Poland'),
('Slovakia'),
('Slovenia'),
('Switzerland'),
('Ukraine'),
('United Kingdom'),
('United States');

```

```

INSERT INTO Guests (email, password, name, surname, document_no, phone_no,
address, zip_code, city, FK_country) VALUES
('john.doe@gmail.com', 'p@ssw0rd', 'John', 'Doe', 'ABC123456',
'123456789', '55 Gresham St', 'E1 7AY', 'London', 20),
('jan.kowalski@cybernet.com.pl', 'p@ssw0rd', 'Jan', 'Kowalski',
'DEF346293', '948374348', 'Strumykowa 15', '51-348', 'Wroclaw', 15),
('k.karolina@gmail.com', 'p@ssw0rd', 'Karolina', 'Kowalska',
'WEC473028', '452838457', 'Powstancow Slaskich 78/2', '34-324', 'Warszawa',
15),
('harmon.beth@gmail.com', 'p@ssw0rd', 'Beth', 'Harmon', 'DFC354282',
'679123644', '445 Park Ave', '10022', 'New York', 21),
('cedrone.e@gmail.com', 'p@ssw0rd', 'Enzo', 'Cedrone', 'VND232134',
'594243484', 'Via dei Girasoli', '00172', 'Roma', 13);

```

```

INSERT INTO Rooms VALUES
(),
(),
(),
(),
();

```

```

INSERT INTO RFIDs (RFID_no, status) VALUES
('2344832389', 1),
('3820433859', 1),
('3326382912', 1),
('0093023738', 0);

```

```

INSERT INTO Tokens (FK_room, FK_rfid, status, created) VALUES
(1, 1, 1, '2022-01-10 18:30:56'),
(2, 4, 1, '2022-01-07 09:17:13'),
(3, 2, 1, '2022-01-07 12:10:18'),
(4, 3, 0, '2022-01-06 17:51:43');

```

```

INSERT INTO CheckIns (FK_room, FK_guest, FK_rfid, validSince, validUntil)
VALUES
(1, 1, 1, '2022-01-01 18:25:34', '2022-01-10 19:45:38'),
(1, 2, 3, '2021-12-10 06:17:23', '2021-12-20 10:05:45');

```

```
INSERT INTO CheckIns (FK_room, FK_guest, FK_rfid, validSince) VALUES
(2, 1, 3, '2022-01-07 21:56:19'),
(4, 5, 2, '2022-01-06 17:18:59'),
(3, 4, 4, '2022-01-04 18:12:20');
```

Broker MQTT

Podstawowa konfiguracja brokera

Wykorzystano broker Mosquitto – lekki broker obsługujący protokół MQTT. Konfiguracja brokera odbywa się w pliku mosquitto.conf.

Jako port wykorzystywany przez broker wykorzystano port 1883 – w tym celu w pliku konfiguracji mosquitto.conf należy wpisać linię:

listener 1883

W celu zasymulowania odczytu karty RFID przez moduł czytnika kart zbliżeniowych RFID znajdujący się w zestawach laboratoryjnych Raspberry Pi napisano skrypt w Pythonie. Wykorzystano bibliotekę paho.mqtt w celu nawiązania połączenia z brokerem Mosquitto.

Kody 1 oraz 2 przedstawiają fragmenty pliku mqtt_conn.py znajdującego się w katalogu /management_node/api/code/

Kod 3 przedstawia fragment pliku mgmt.py znajdującego się w katalogu /management_node/api/code/

```
def connect_to_broker():
    client.connect(sys.argv[1], 8883, 600)
    client.on_message = process_message
    client.loop_start()
    topic = "room/{0}/listen".format(sys.argv[2])
    client.subscribe(topic, 1)
    print("{0} listening on topic {1}".format(sys.argv[2], topic))
```

Kod 1. Łączenie z brokerem Mosquitto i konfiguracja subskrypcji dla klienta – w parametrze sys.argv[1] przekazywany jest adres ip brokera mqtt

```
def disconnect_from_broker():
    client.loop_stop()
    client.disconnect()
```

Kod 2. Odłączenie klienta od brokera Mosquitto

```
def check_rfid(topic, rfid_nbr):  
    client.publish(topic, rfid_nbr)  
    print("Publishing {1} on topic: {0}".format(topic, rfid_nbr), 1)
```

Kod 3. Przesłanie wiadomości przez klienta

Szyfrowanie TLS/SSL

Zaimplementowano szyfrowanie komunikacji za pomocą TLS w wersji 1.2. Protokół TLS/SSL wykorzystuje port 8883, dlatego w ustawieniach brokera Mosquitto oraz dockera symulującego klienta należało zmienić numer portu na 8883.

W celu zaimplementowania szyfrowania komunikacji za pomocą TLS wygenerowano odpowiednie certyfikaty przy użyciu komendy openssl. Certyfikaty wygenerowano na lokalnym Windowsie, a następnie przeniesiono na maszyny na dockerze. Dokładny opis wykorzystanych komend można znaleźć w dokumentacji OpenSSL [\[1\]](#)

Wygenerowanie certyfikatu CA razem z kluczem prywatnym:

```
openssl req -x509 -nodes -sha256 -new -keyout ca.key -out ca.crt
```

Zdecydowano nie zabezpieczać klucza prywatnego hasłem.

Wygenerowanie klucza prywatnego dla brokera Mosquitto:

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

Ponownie zdecydowano nie zabezpieczać klucza prywatnego hasłem.

Wygenerowanie żądania o podpisanie certyfikatu dla brokera Mosquitto:

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

Podpisanie certyfikatu dla brokera Mosquitto:

```
openssl x509 -req -in broker.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out broker.crt
```

Analogicznie do certyfikatu dla brokera Mosquitto wygenerowano certyfikat dla klientów. Wszyscy klienci używają tego samego certyfikatu w celu uproszczenia zarządzania certyfikatami.

Bardzo istotne jest dodanie na brokerze certyfikatu CA jako certyfikat zaufany. Ponieważ broker jest postawiony na Linuxie Alpine wykonuje się to za pomocą komendy:

```
cat /mosquitto/config/ca.crt >> /etc/ssl/certs/ca-certificates.crt
```

Ta komenda została wpisana do pliku dockerfile brokera, dzięki czemu przy każdym tworzeniu obrazu certyfikat CA będzie wpisywany do spisu certyfikatów zaufanych.

Dzięki temu broker jest w stanie zweryfikować certyfikaty klientów, którzy się do niego podłączają.

W konfiguracji brokera Mosquitto ustawiono:

listener 8883

allow_anonymous true

cafile /mosquitto/config/ca.crt

certfile /mosquitto/config/broker.crt

keyfile /mosquitto/config/broker.key

tls_version tlsv1.2

Ustawiono, aby broker nasłuchiwał na porcie 8883, pozwalał na ruch anonimowy oraz korzystał z odpowiednich certyfikatów i kluczy do szyfrowania. Ustawiono też wersję protokołu TLS.

Przy takich ustawieniach komunikacja jest szyfrowana, ale klienci nie są w żaden sposób uwierzytelniani i w dalszym ciągu osoby z zewnątrz mogą podsłuchać komunikację.

Kod 4 przedstawia fragment pliku `mqtt_connect.py` znajdującego się w katalogu `/management_node/api/code/`

```
def connect_to_broker(broker):  
    client.tls_set(ca_certs="/code/app/ca.crt", certfile="/code/app/client.crt", keyfile="/code/app/client.key",  
                  tls_version=ssl.PROTOCOL_SSLv23, ciphers=None, cert_reqs=ssl.CERT_NONE)
```

Kod 4. W skrypcie w pythonie należy ustawić klientowi certyfikaty, z których ma korzystać przy komunikacji z brokerem przy użyciu metody `tls_set(...)`

Uwierzytelnianie TLS/SSL

W celu wprowadzenia uwierzytelniania TLS/SSL dodano następujące ustawienia na brokerze Mosquitto:

password_file /mosquitto/config/passwords.pwd

allow_anonymous false

require_certificate true

W pliku passwords.pwd znajdują się nazwy użytkowników i zahashowane hasła tych użytkowników. Hasła wygenerowano na lokalnym Windowsie przy użyciu komendy:

mosquitto_passwd -c <password file> <username>

mosquitto_passwd <password file> <username>

Opcja -c powoduje utworzenie (lub nadpisanie, jeśli już istniał) pliku. Plik z hasłami przekopiowano na broker Mosquitto. Broker nie pozwala na anonimowy dostęp, a także wymaga od klientów posiadania ważnego certyfikatu, podpisanego certyfikatem CA.

Oprócz ustawienia certyfikatów klientowi przed połączeniem z brokerem należy ustawić nazwę użytkownika i hasło, którym klient będzie się posługiwał.

Kod 5 przedstawia fragment pliku mqtt_connect.py znajdującego się w katalogu /management_node/api/code/

```
def connect_to_broker(broker):
    client.tls_set(ca_certs="/code/app/ca.crt", certfile="/code/app/client.crt", keyfile="/code/app/client.key",
                  tls_version=ssl.PROTOCOL_SSLv23, ciphers=None, cert_reqs=ssl.CERT_NONE)
    client.username_pw_set("<name>", "<password>")
    client.connect(broker, 8883, 600)
```

Kod 5. W skrypcie w pythonie należy ustawić klientowi nazwę użytkownika i hasło przy użyciu metody username_pw_set(...)

Nazwa użytkownika i hasło, którymi posługuje się klient, muszą znajdować się w pliku skonfigurowanym jako password_file na brokerze.

```
C:\Users\galar\Desktop\uni\listy + materiały\IoT_Project\ManagementNode>docker compose logs broker
mqtt_broker | 2022-01-16 08:47:34: mosquitto version 2.0.14 starting
mqtt_broker | 2022-01-16 08:47:34: Config loaded from /mosquitto/config/mosquitto.conf.
mqtt_broker | 2022-01-16 08:47:34: Opening ipv4 listen socket on port 8883.
mqtt_broker | 2022-01-16 08:47:34: Opening ipv6 listen socket on port 8883.
mqtt_broker | 2022-01-16 08:47:34: mosquitto version 2.0.14 running
mqtt_broker | 2022-01-16 08:47:47: New connection from 172.19.0.1:44026 on port 8883.
mqtt_broker | 2022-01-16 08:47:47: New client connected from 172.19.0.1:44026 as mgmt (p2, c1, k600, u'mgmt').
mqtt_broker | 2022-01-16 08:49:38: New connection from 172.19.0.1:44038 on port 8883.
mqtt_broker | 2022-01-16 08:49:38: New client connected from 172.19.0.1:44038 as gate1 (p2, c1, k600, u'gate1').
```

Screen 2. Pomyślne połączenie klientów z brokerem – klienci posiadają identyfikator, a w nawiasie wpisana jest nazwa użytkownika klienta

```
root@194341332dac:/code# bash start_service hotel.domain 1
1 listening on topic room/1/listen
Publishing 2344832389 on topic: room/1 1
Gate 1, Hello John Doe
Publishing 3820433859 on topic: room/1 1
Gate 1, Alarming, Enzo Cedrone you are not allowed to leave quarantine
Publishing 3326382912 on topic: room/1 1
Gate 1, Hello Jan Kowalski
Publishing 0093023738 on topic: room/1 1
Gate 1, keep close, not registered card
Publishing 4248555247 on topic: room/1 1
Gate 1, keep close, not registered card
```

Screen 3. Komunikacja z klienta, symulującego bramkę z czujnikiem RFID poprzez broker do węzła zarządzania który odpowiada, jak zachować się dla danego numeru karty RFID

```
C:\Program Files\mosquitto>mosquitto_sub.exe -h hotel.domain -p 8883 -t "room/#"
Error: A TLS error occurred.

C:\Program Files\mosquitto>mosquitto_pub.exe -h hotel.domain -p 8883 -t "room/1" -m "Not Authorized"
Error: A TLS error occurred.
```

Screen 4. Podsluchanie komunikacji bez autoryzacji nie jest możliwe, broker odrzuca połączenie

API

Do implementacji API wykorzystano bibliotekę Pythona fastapi [\[2\]](#). W celu pozyskania danych z bazy danych klienci komunikują się z API, które następnie komunikuje się z węzłem zarządzania, bezpośrednio komunikującym się z bazą danych.

Kody 6, 8 oraz 9 przedstawiają fragment pliku api.py znajdującego się w katalogu /management_node/api/code/

Kod 7 przedstawia fragment pliku mqtt_connect.py znajdującego się w katalogu /management_node/api/code/

Kod 10 przedstawia fragment pliku mgmt.py znajdującego się w katalogu /management_node/api/code/

W celu zdefiniowania metod dostępnych w API najpierw konieczne jest przede wszystkim utworzenie obiektu typu FastAPI oraz utworzenie obiektu klasy zarządzającej.

```
from mgmt import Mgmt
import mqtt_conn as mqtt

app = FastAPI(root_path="/api")
mn = Mgmt()
mqtt.connect_to_broker("hotel.domain")
```

Kod 6. Utworzenie obiektu typu FastAPI oraz obiektu klasy zarządzającej, która bezpośrednio łączy się z bazą danych

```
client = mqtt.Client("mgmt", protocol=MQTTv311)

def connect_to_broker(broker):
    client.tls_set(ca_certs="/code/app/ca.crt", certfile="/code/app/client.crt", keyfile="/code/app/client.key",
                  tls_version=ssl.PROTOCOL_SSLv23, ciphers=None, cert_reqs=ssl.CERT_NONE)
    client.username_pw_set("mgmt", "1234")
    client.connect(broker, 8883, 600)

    client.on_message = process_message
    client.loop_start()
    client.subscribe("room/+", qos=1)
```

Kod 7. Wykorzystany fragment mqtt_conn.py

Zdefiniowano odpowiednie modele danych w celu lepszej organizacji oraz większej prostoty kodu. Dzięki modelom danych można jako argument metody przekazać element typu modelu i w ciele metody odwołać się do jego pól, zamiast przekazywać wielu argumentów do metody.

```

class RfidToRoom(BaseModel):
    room_id: int
    rfid_id: int

class Room(BaseModel):
    room_id: int

class Guest(BaseModel):
    guest_id: int

class NewGuest(BaseModel):
    email: str
    name: str
    surname: str
    password: str
    doc_no: str
    phone_no: int
    address: str
    zip_code: str
    city: str
    country_code: int

class CheckIn(BaseModel):
    token: int
    guest_id: int

```

Kod 8. Definicja modeli

Metody FastAPI zostaną omówione na przykładzie `get_guest(guest: Guest)`. Metoda ma dla danego id gościa zwrócić dane gościa o takim id.

We wszystkich metodach wykorzystywana jest operacja HTTP post – dzięki temu parametry metody zostaną przesłane na serwer jako ciało wiadomości HTTP i nie zostaną dodane do URL. Wewnątrz metody FastAPI wywoływana jest odpowiednia metoda obiektu klasy zarządzającej – w tym przypadku, jest to metoda zwracająca dane gościa o danym id. W klasie zarządzającej wykonywana jest kwerenda SQL na bazie danych, a następnie wynik kwerendy jest zapisywany jako słownik JSON.

```

@app.post("/getGuest")
def get_guest(guest: Guest):
    return mn.get_guest(guest.guest_id)

```

Kod 9. Metoda `get_guest` w pliku `api.py`

```

def get_guest(self, guest_id: int):
    result = {}
    if guest_id in self.get_one_parameter_list("SELECT id FROM Guests"):
        cmd = "SELECT Guests.name, surname, document_no, phone_no, email, address, zip_code, city, Countries.name " \
            "FROM Guests JOIN Countries ON Guests.FK_country = Countries.id WHERE Guests.id={0}".format(guest_id)
        cur = self._conn.cursor()
        cur.execute(cmd)
        for name, sur, doc, pho, ema, add, code, cit, country in cur:
            result = {"name": name, "surname": sur, "document_no": doc,
                    "phone_no": pho, "email": ema, "address": add,
                    "zip_code": code, "city": cit, "country": country}
        cur.close()
    return result

```

Kod 10. Metoda `get_guest` w pliku `mgmt.py`

Opis działania i prezentacja interfejsu

Opis sposobu instalacji i uruchomienia

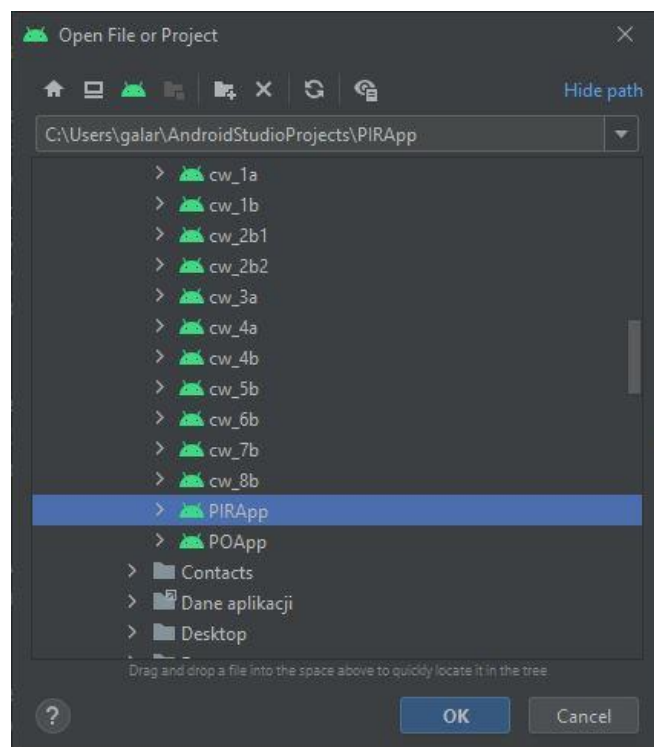
Instalacja i uruchomienie aplikacji mobilnej

Wymagania:

1. Dostęp do internetu.
2. Android Studio w wersji Arctic Fox 2020.3.1 lub wyższej zainstalowany i skonfigurowany na maszynie, na której uruchomiony zostanie emulator urządzenia mobilnego klienta.

Kroki instalacji i uruchomienia:

1. Rozpakuj paczkę z projektem aplikacji, branch “mobile_app”, w docelowym folderze przechowywania projektu aplikacji – może być to folder AndroidStudioProjects, w którym Android Studio domyślnie przechowuje projekty.
2. Otwórz projekt w programie Android Studio.



Przykład 1. Okno wyboru projektu w programie Android Studio – tutaj projekt z aplikacją to PIRApp

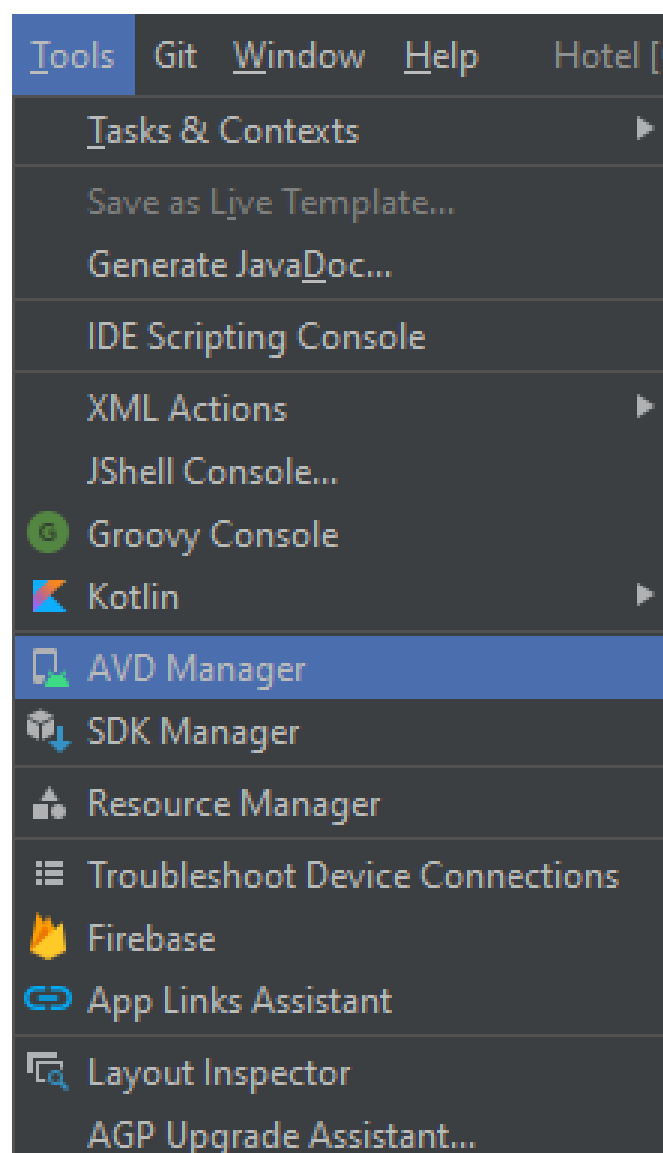
3. Konfiguracja URL do API w kodzie aplikacji.

Wymagane jest wprowadzenie zmian w trzech plikach: CheckInFragment.kt, RoomCardFragment.kt oraz UserFragment.kt. Należy dostosować wartość zmiennej BASEURL – zmienić adres IP w adresie URL na adres serwera aplikacji według wzoru *http://<adres_serwera>:8080/api/*

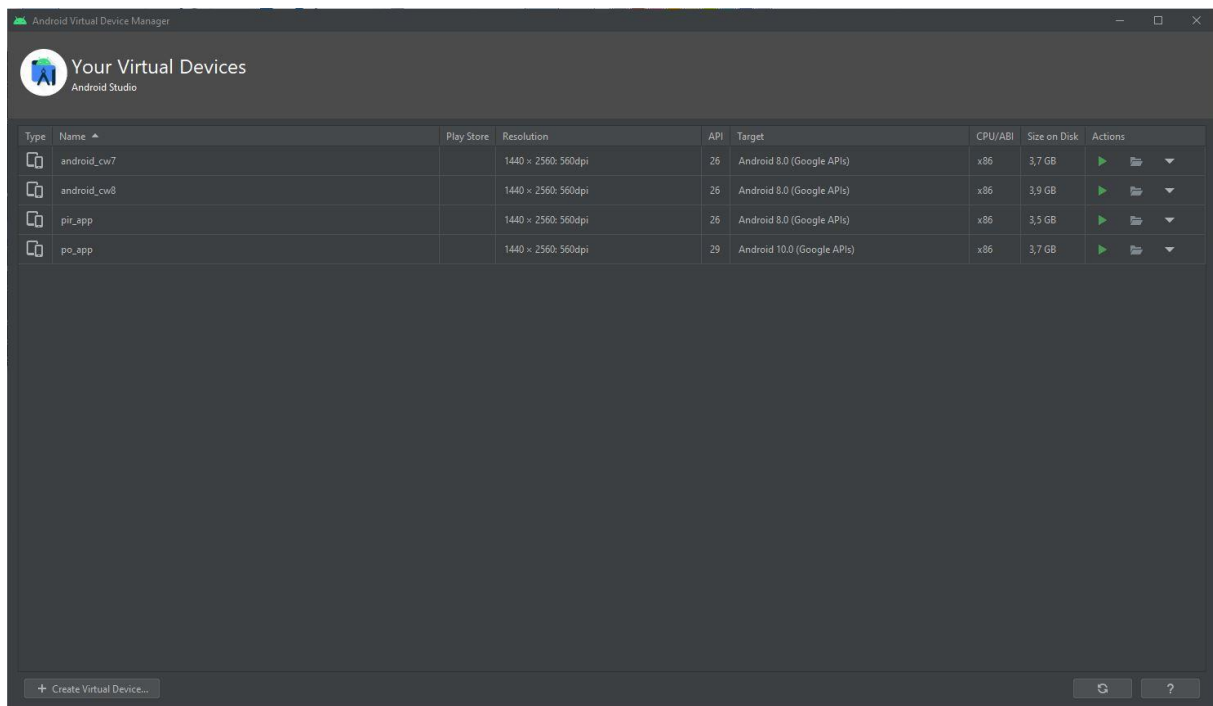
```
private val BASEURL = "http://192.168.0.101:8080/api/"
```

Przykład 2. Zmienna, której wartość należy zmienić

4. Utworzenie i konfiguracja emulatora urządzenia mobilnego z systemem Android.
 - a. wybierz z menu Tools opcję AVD Manager

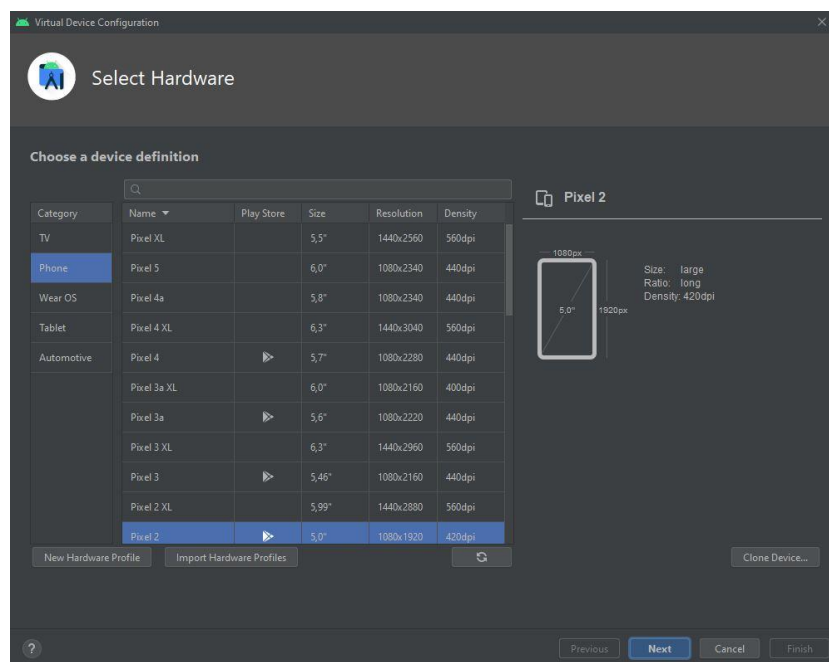


Przykład 3. Wybór opcji AVD Manager



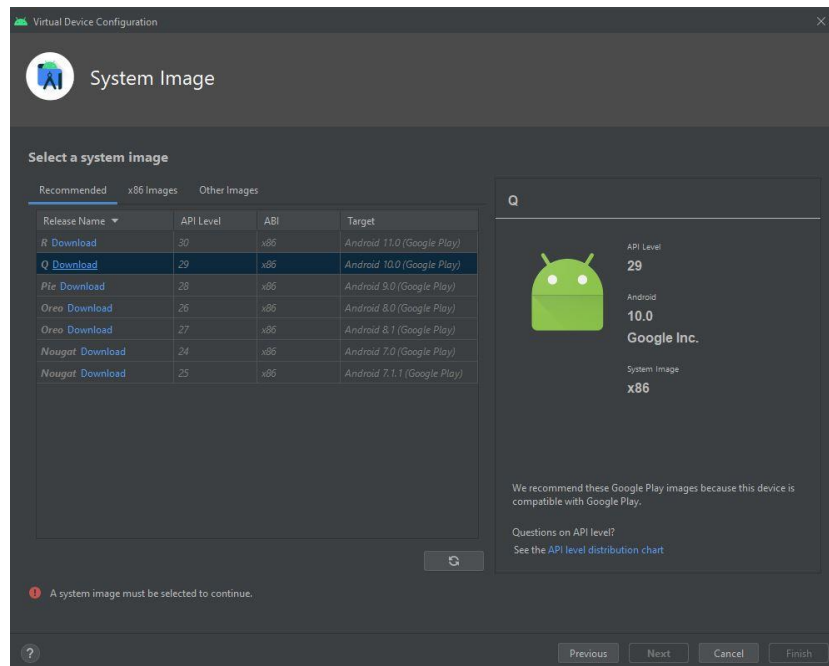
Przykład 4. Oczekiwany rezultat: pojawi się okno zarządzania maszynami wirtualnymi

b. Naciśnij przycisk Create Virtual Device...



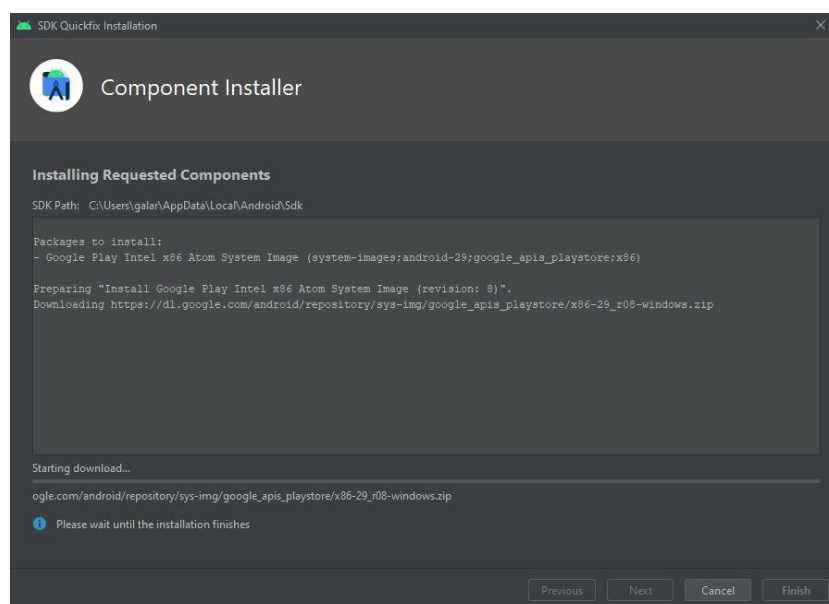
Przykład 5. Oczekiwany rezultat: pojawi się okno tworzenia nowej maszyny wirtualnej

- c. Wybierz hardware wirtualnego urządzenia – konkretny wybór nie ma znaczenia. Po wybraniu hardware naciśnij Next.



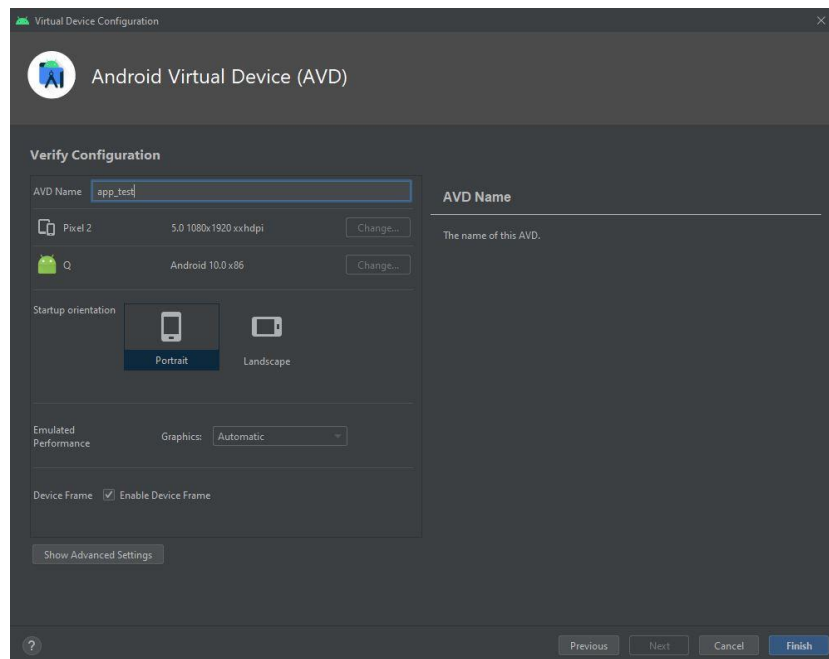
Przykład 6. Oczekiwany rezultat: pojawi się okno wyboru systemu operacyjnego urządzenia

- d. Do uruchomienia aplikacji wymagane jest urządzenie z Androidem 10. Pobierz Android 10 (Android Q). Ten proces może zająć kilka, nawet kilkanaście minut.



Przykład 7. Oczekiwany rezultat: rozpocznie się pobieranie obrazu systemu Android

- e. Wybierz pobrany obraz systemu Android i naciśnij Next.



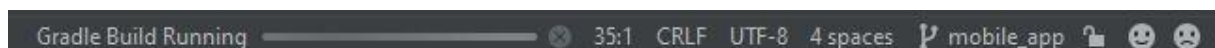
Przykład 8. Oczekiwany rezultat: pojawi się ostatni ekran tworzenia wirtualnego urządzenia

- f. Nazwij swoje urządzenie i naciśnij Finish.
- g. Na górnym pasku po prawej stronie wybierz swoje urządzenie i naciśnij przycisk Run. Rozpocznie się proces budowania aplikacji, który może zająć kilka minut. Uruchomiony zostanie emulator urządzenia mobilnego.

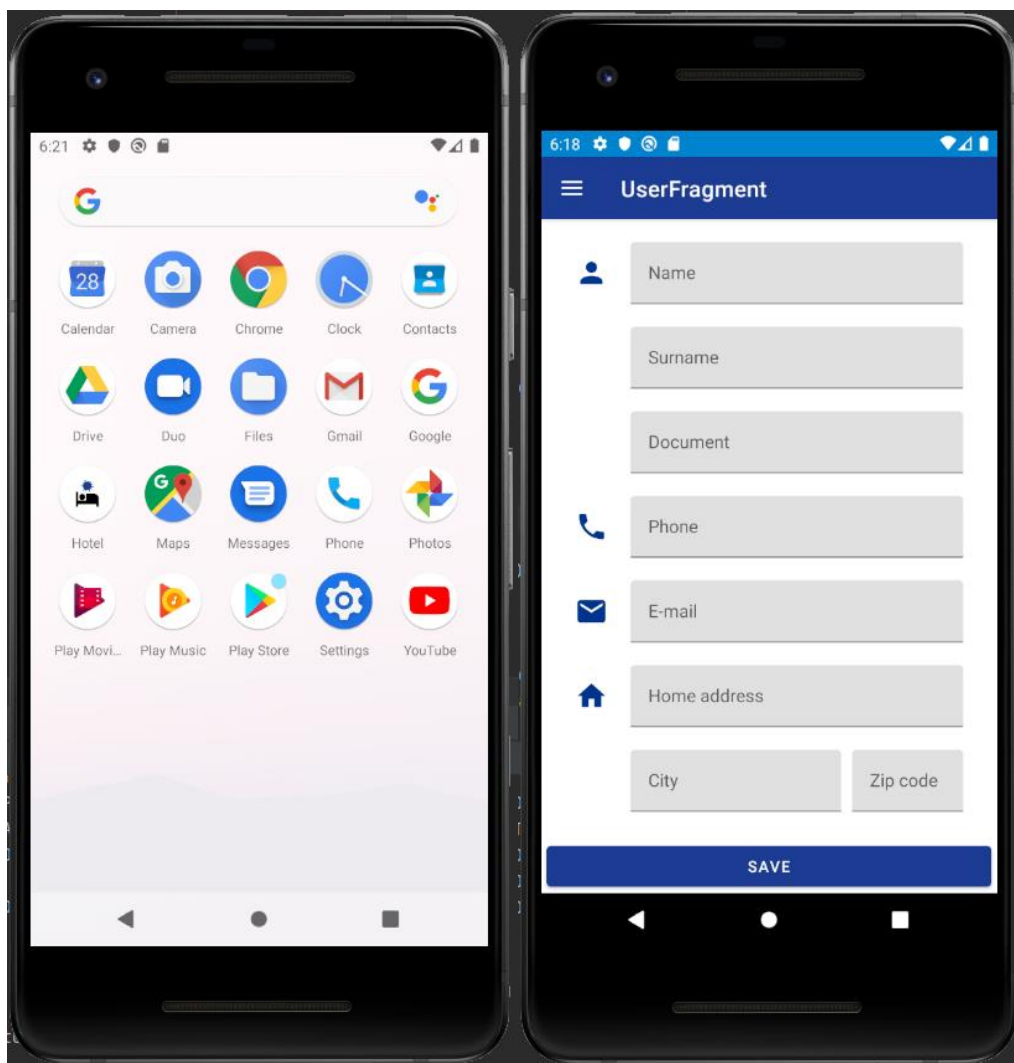


Przykład 10. Widok górnego paska z wybranym utworzonym urządzeniem

- h. Po zakończeniu procesu budowania aplikacji automatycznie powinien rozpocząć się proces instalowania aplikacji na emulatorze urządzenia mobilnego. Jeśli to nie nastąpi, naciśnij ponownie przycisk Run.



Przykład 11. Widok dolnego paska – obserwuj, czy rozpoczął się proces Install. Powinien automatycznie rozpocząć się po zakończeniu budowania aplikacji, ale może tak się nie zdarzyć. Wtedy naciśnij ponownie przycisk Run.



Przykład 12, 13. Aplikacja zainstalowana i działająca na emulatorze

5. Aby uruchomić aplikację po zainstalowaniu wystarczy uruchomić emulator urządzenia mobilnego i uruchomić aplikację. Aplikację można również odinstalować i zainstalować ponownie, jeśli zajdzie taka potrzeba.
6. Mogą występować problemy z emulatorem urządzenia mobilnego. W takiej sytuacji może pomóc wyczyszczenie danych z emulatora. W tym celu należy otworzyć AVD manager – otworzyć menu emulatora i wybrać opcję “Wipe data”.

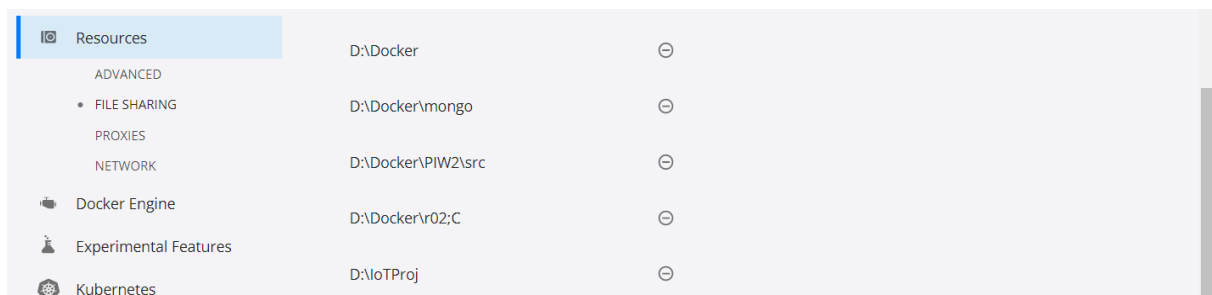
Instalacja i uruchomienie serwera aplikacji

Wymagania:

1. Dostęp do internetu
2. Docker desktop zainstalowany i skonfigurowany na maszynie serwera, wersją docker engine v20.10.12 lub wyższą oraz włączonym mechanizmem orkiestracji docker compose V2 (Settings->General->checkbox "Use Docker Compose V2")
3. Dostęp administratora do maszyny serwera

Kroki instalacji i uruchomienia:

1. Rozpakuj paczkę z kodem serwera, folder "ManagementNode", w docelowym folderze przechowywania kodu i bazy danych
2. Sprawdź czy docker engine ma dostęp do folderu w aplikacji docker desktop, Settings->Resources->File Sharing. Jeśli nie widnieje wpis ze ścieżką do folderu "ManagementNode" dodaj wpis.



Przykład 14. Ustawienie udostępniania systemu plików, wpis D:/lotProj

3. Wykonaj wpis w pliku /etc/hosts, na systemie linux /etc/hosts, na windowsie C:\Windows\System32\drivers\etc\hosts, po otwarciu pliku w dowolnym edytorze z uprawnieniami administratora, dodać wpis, i zapisać plik:
<adres_ip_serwera> hotel.domain

```
# localhost name resolution is handled within DNS itself.
# 127.0.0.1 localhost
# ::1 localhost
192.168.1.107 hotel.domain
```

Przykład 15. Utworzenie domeny na danym adresie

4. Włączyć docker engine, poprzez uruchomienie Docker Desktop
5. Korzystając z konsoli komend przejść do folderu "ManagementNode", wykonać komendę *docker compose up -d --build*. Nastąpi pobieranie pakietów oraz budowanie obrazów, proces może potrwać kilka minut.

```
D:\IoTProj\IoT_Project\ManagementNode>docker compose up -d --build
[+] Building 371.8s (20/20) FINISHED
=> [managementnode_broker internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 32B 0.0s
=> [managementnode_broker internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [managementnode_mgmt internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 32B 0.0s
=> [managementnode_broker internal] load metadata for docker.io/library/eclipse-mosquitto:latest 11.6s
=> [managementnode_mgmt internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [managementnode_mgmt internal] load metadata for docker.io/library/python:3.9 12.5s
=> [managementnode_broker 1/7] FROM docker.io/library/eclipse-mosquitto@sha256:64b7c1729f7d1ffff46b5e884fc389071686b2f 0.0s
=> [managementnode_broker internal] load build context 0.4s
=> => transferring context: 46.01kB 0.4s
=> CACHED [managementnode_broker 2/7] COPY ./mosquitto.conf /mosquitto/config/mosquitto.conf 0.0s
=> CACHED [managementnode_broker 3/7] COPY ./ca.crt /mosquitto/config/ca.crt 0.0s
=> CACHED [managementnode_broker 4/7] COPY ./broker.crt /mosquitto/config/broker.crt 0.0s
=> CACHED [managementnode_broker 5/7] COPY ./broker.key /mosquitto/config/broker.key 0.0s
```

Przykład 16. Komenda docker compose up -d --build

6. Zweryfikuj stan serwisów – wydaj komendę: *docker compose ps*. Serwisy mariadb, broker, webserver powinny być w stanie running, serwis mgmt może być w stanie restarting z powodu nie stworzonej jeszcze bazy danych.

```
D:\IoTProj\IoT_Project\ManagementNode>docker compose ps
NAME                COMMAND                                SERVICE    STATUS          PORTS
database            "docker-entrypoint.s..."            mariadb    running (healthy) 3306/tcp
mgmt                 "uvicorn app.api:app..."            mgmt       restarting
mqtt_broker         "/docker-entrypoint..."            broker     running          0.0.0.0:8883->8883/tcp
webserver            "/docker-entrypoint..."            webserver  running          0.0.0.0:8080->80/tcp
```

Przykład 17. Oczekiwany rezultat *docker compose ps*

7. Stwórz bazę danych i zasiej danymi,
Wydaj komendę: *docker exec -it database /bin/bash*
Powinien się pojawić znak zachęty kontenera *root@<id>:/#*

```
D:\IoTProj\IoT_Project\ManagementNode>docker exec -it database /bin/bash
root@f8c483b729d2:/#
```

Przykład 18. Pojawienie się znaku zachęty kontenera

- Wydaj komendę: *mysql --user=admin --password=p@ss*
Powinien się pojawić znak zachęty bazy mariadb.

```
root@f8c483b729d2:/# mysql --user=admin --password=p@ss
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 203
Server version: 10.2.41-MariaDB-1:10.2.41+maria~bionic mariadb.org binary distribution
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MariaDB [(none)]> _
```

Przykład 19. Pojawienie się znaku zachęty mariadb

- Stwórz bazę danych używając skryptu: *source /home/src/create_database*

```
MariaDB [(none)]> source /home/src/create_database
Query OK, 0 rows affected, 1 warning (0.01 sec)

Query OK, 1 row affected (0.06 sec)

Database changed
Query OK, 0 rows affected (0.07 sec)

Query OK, 0 rows affected (0.15 sec)

Query OK, 0 rows affected (0.12 sec)

Query OK, 0 rows affected (0.13 sec)

Query OK, 0 rows affected (0.12 sec)

Query OK, 0 rows affected (0.11 sec)

Query OK, 0 rows affected (0.43 sec)
Records: 0 Duplicates: 0 Warnings: 0

Query OK, 0 rows affected (0.29 sec)
Records: 0 Duplicates: 0 Warnings: 0

Query OK, 0 rows affected (0.37 sec)
Records: 0 Duplicates: 0 Warnings: 0

Query OK, 0 rows affected (0.30 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Przykład 20. Oczekiwany rezultat skryptu `source /home/src/create_database`

Zasiej bazę danych danymi: `source /home/src/insert_data`

```

MariaDB [hotel]> source /home/src/insert_data
Database changed
Query OK, 21 rows affected (0.04 sec)
Records: 21 Duplicates: 0 Warnings: 0

Query OK, 5 rows affected (0.04 sec)
Records: 5 Duplicates: 0 Warnings: 0

Query OK, 5 rows affected (0.03 sec)
Records: 5 Duplicates: 0 Warnings: 0

Query OK, 4 rows affected (0.04 sec)
Records: 4 Duplicates: 0 Warnings: 0

Query OK, 4 rows affected (0.04 sec)
Records: 4 Duplicates: 0 Warnings: 0

Query OK, 2 rows affected (0.04 sec)
Records: 2 Duplicates: 0 Warnings: 0

Query OK, 3 rows affected (0.01 sec)
Records: 3 Duplicates: 0 Warnings: 0

MariaDB [hotel]>

```

Przykład 21. Oczekiwany rezultat skryptu `source /home/src/insert_data`

Wyjdź z kontenera database.

```

MariaDB [hotel]> exit
Bye
root@f8c483b729d2:/# exit
exit

D:\IoTProj\IoT_Project\ManagementNode>

```

Przykład 22. Wyjście z kontenera database

8. Zrestartuj serwis mgmt:

Wykonaj komendę: `docker compose restart mgmt`

Zweryfikuj stan serwisu: `docker compose ps`

```

D:\IoTProj\IoT_Project\ManagementNode>docker compose restart mgmt
[+] Running 1/1
 - Container mgmt Started

D:\IoTProj\IoT_Project\ManagementNode>docker compose ps

```

NAME	COMMAND	SERVICE	STATUS	PORTS
database	"docker-entrypoint.s..."	mariadb	running (healthy)	3306/tcp
mgmt	"uvicorn app.api:app..."	mgmt	running	80/tcp
mqtt_broker	"/docker-entrypoint..."	broker	running	0.0.0.0:8883->8883/tcp
webserver	"/docker-entrypoint..."	webserver	running	0.0.0.0:8080->80/tcp

```

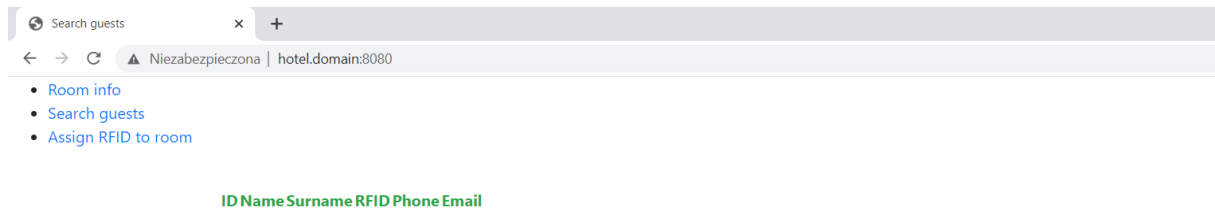
D:\IoTProj\IoT_Project\ManagementNode>

```

Przykład 23. Oczekiwany rezultat komendy `docker compose ps`

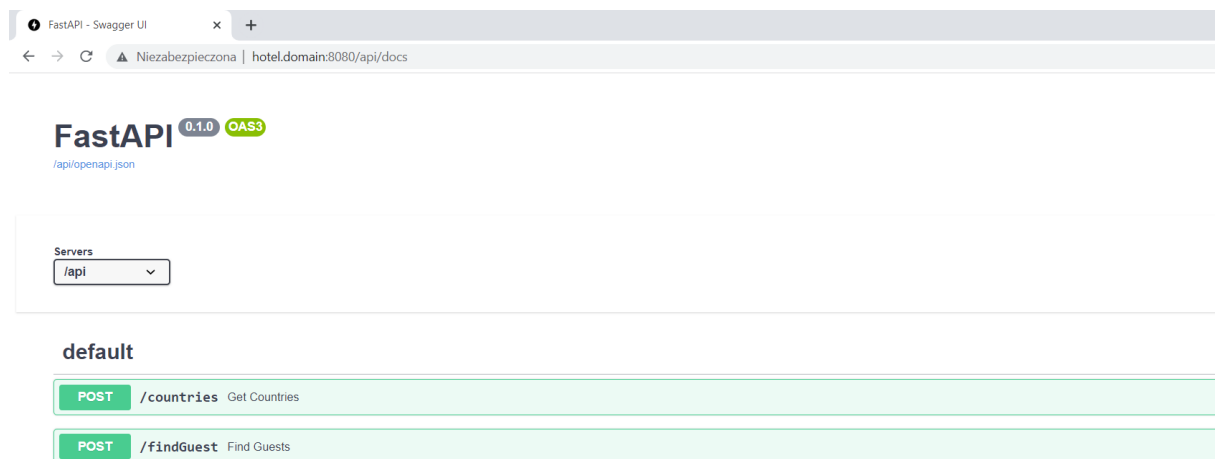
9. Zweryfikuj działanie serwera webowego:

W przeglądarce internetowej wpisz adres: <http://hotel.domain:8080/>
Strona hotelu powinna być dostępna.



Przykład 24. Strona hotelu jest dostępna

W przeglądarce internetowej wpisz adres: <http://hotel.domain:8080/api/docs>
Strona API powinna być dostępna.



Przykład 25. Strona API jest dostępna

W razie problemów z połączeniem sprawdź poprawność wpisu w `/etc/hosts` i zrestartuj serwis mgmt poleceniem `docker compose restart mgmt`

10. Uruchomienie programu bramki i weryfikacja działania brokera mqtt
 - a. Rozpakuj folder Client i przejdź do niego w konsoli komend
 - b. Uruchom kontener klienta, poleceniem *docker compose up -d --build*

```
D:\IoTProj\IoT_Project\Client>docker compose up -d --build
[+] Building 14.6s (15/15) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 379B                                              0.0s
=> [internal] load .dockerignore                                                  0.0s
=> => transferring context: 2B                                                    0.0s
=> [internal] load metadata for docker.io/library/python:3.9                    7.2s
=> [ 1/10] FROM docker.io/library/python:3.9@sha256:9705b19ced990d8601806d6ce86bd58b2a60143f7be9bf9c88811419a46cb5fc 0.0s
=> [internal] load build context                                                  0.2s
=> => transferring context: 6.51kB                                                0.1s
=> CACHED [ 2/10] WORKDIR /code                                                  0.0s
=> [ 3/10] COPY ./ca.crt /code/ca.crt                                             0.1s
=> [ 4/10] COPY ./client.crt /code/client.crt                                    0.1s
=> [ 5/10] COPY ./client.key /code/client.key                                    0.1s
=> [ 6/10] COPY ./mqtt_connect.py /code/mqtt_connect.py                         0.1s
=> [ 7/10] COPY ./start_service /code/start_service                             0.1s
=> [ 8/10] RUN chmod +x /code/mqtt_connect.py                                   0.6s
=> [ 9/10] RUN chmod +x /code/start_service                                       0.3s
=> [10/10] RUN python -m pip install --user paho-mqtt                            5.0s
=> exporting to image                                                            0.4s
=> => exporting layers                                                            0.4s
=> => writing image sha256:d69678e7832c03ebb9de82050eea7a35e52159ef075f6cc17a2654125de78949 0.0s
=> => naming to docker.io/library/client_client                                0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
[+] Running 2/2
 - Network client_default Created                                                0.1s
 - Container gate1 Started                                                       6.6s

D:\IoTProj\IoT_Project\Client>
```

Przykład 26. Oczekiwany rezultat komendy *docker compose up -d --build*

- c. Zweryfikuj działanie serwisu poleceniem *docker compose ps*

```
D:\IoTProj\IoT_Project\Client>docker compose ps
NAME                COMMAND                SERVICE    STATUS    PORTS
gate1               "sleep 1d"             client     running   8883/tcp

D:\IoTProj\IoT_Project\Client>
```

Przykład 27. Oczekiwany rezultat komendy *docker compose ps*

- d. Wejdź do kontenera gate1, *docker exec -it gate1 /bin/bash*

```
D:\IoTProj\IoT_Project\Client>docker exec -it gate1 /bin/bash
root@5ba220c94b80:/code#
```

Przykład 28. Oczekiwany rezultat komendy *docker exec -it gate1 /bin/bash*

- e. Uruchom skrypt generujący symulowany ruch na bramce:
bash start_service hotel.domain 1

```
root@5ba220c94b80:/code# bash start_service hotel.domain 1
1 listening on topic room/1/listen
Publishing 2344832389 on topic: room/1 1
Gate 1, Hello John Doe
Publishing 3820433859 on topic: room/1 1
Gate 1, Alarming, Enzo Cedrone you are not allowed to leave quarantine
Publishing 3326382912 on topic: room/1 1
Gate 1, Hello Jan Kowalski
Publishing 0093023738 on topic: room/1 1
Gate 1, keep close, not registered card
Publishing 4463711332 on topic: room/1 1
Gate 1, keep close, not registered card
Publishing 3074938999 on topic: room/1 1
Gate 1, keep close, not registered card
Publishing 2942596148 on topic: room/1 1
Gate 1, keep close, not registered card
Publishing 4611969659 on topic: room/1 1
Gate 1, keep close, not registered card
Publishing 4365236715 on topic: room/1 1
Gate 1, keep close, not registered card
Publishing 3533592554 on topic: room/1 1
Gate 1, keep close, not registered card
Publishing 3895314902 on topic: room/1 1
Gate 1, keep close, not registered card
Publishing 3132965607 on topic: room/1 1
Gate 1, keep close, not registered card
Publishing 2949951563 on topic: room/1 1
Gate 1, keep close, not registered card
Publishing 3912627106 on topic: room/1 1
Gate 1, keep close, not registered card
root@5ba220c94b80:/code#
```

Przykład 29. Oczekiwany rezultat komendy *bash start_service hotel.domain 1*

W razie problemów wykonaj restart serwisu mgmt (*docker compose restart mgmt*). Serwisy klienta oraz serwera można wyłączyć poprzez komendę *docker compose down*.

Aplikacja mobilna

Gość hotelowy pobiera aplikację po przybyciu do hotelu. Ekran startowy, przedstawiony na rysunkach 12 oraz 13, umożliwia wprowadzenie danych dotyczących gościa. Te dane są zbierane i przechowywane w bazie danych w celu identyfikacji gości oraz wyświetlania informacji o gościach przez pracowników hotelu. Po naciśnięciu przycisku save dane użytkownika zapisywane są w bazie danych oraz do aplikacji przekazywana jest informacja o id gościa w bazie danych.

The image displays two side-by-side screenshots of a mobile application interface for entering user data. Both screens show a form with fields for Surname, Document, Phone, E-mail, Home address, City, Zip code, and Country. A 'SAVE' button is at the bottom. The left screen has a blue header 'Hotel' and a list of icons on the left. The right screen has a blue header 'Hotel' and a list of icons on the left.

Left Screenshot (Rysunek 12):

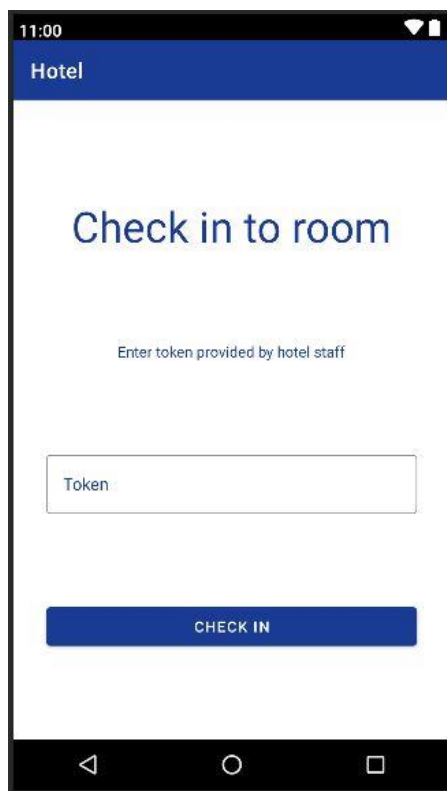
- Header: Hotel
- Fields: Surname, Document, Phone, E-mail, Home address, City, Zip code, Country (dropdown).
- Buttons: SAVE

Right Screenshot (Rysunek 13):

- Header: Hotel
- Fields: Name, Surname, Document, Phone, E-mail, Home address, City, Zip code.
- Buttons: SAVE

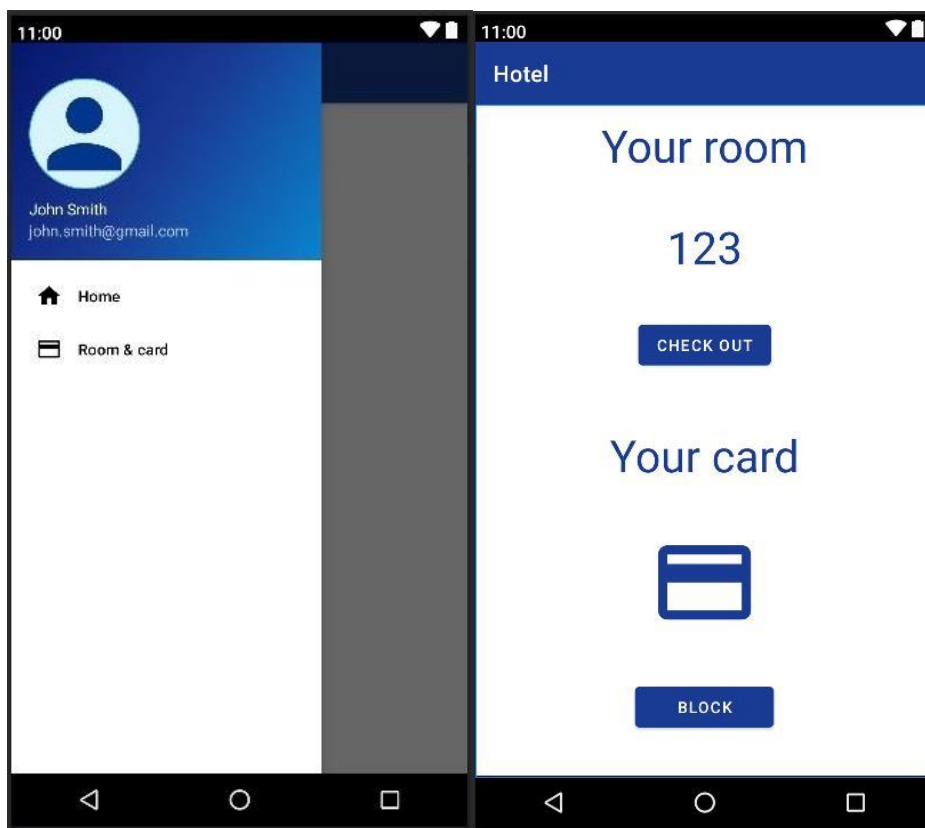
Rysunek 12, 13. Ekran wprowadzania danych użytkownika

Po uzupełnieniu swoich danych w celu zameldowania się do konkretnego pokoju gość podaje w odpowiednim oknie aplikacji token, który otrzymał od pracownika hotelu. Ten token jest jednorazowy i pozwala połączyć gościa z kartą RFID, którą będzie się posługiwał oraz pokojem, w którym będzie się zatrzymywał. Dzięki temu gość będzie w stanie otworzyć swoją kartą RFID swój pokój.



Rysunek 14. Ekran wprowadzenia tokena, którego gość otrzymuje od pracownika hotelu

Po wykonaniu powyższych czynności gość może korzystać z aplikacji w celu przeglądania swoich danych (zakładka Home) oraz zablokowania karty w przypadku jej zgubienia lub wymeldowania się z hotelu po zakończonym pobycie (zakładka Room & card).



Rysunek 15, 16. Ekran nawigacji oraz ekran informujący gościa o numerze jego pokoju oraz pozwalający zablokować kartę

Do komunikacji z fastapi udostępnianego przez serwer aplikacji wykorzystano bibliotekę Volley [\[3\]](#). Pomijając zbędne szczegóły dotyczące implementacji aplikacji natywnych na system Android, aby móc korzystać z biblioteki do pliku build.gradle dla modułu należy dodać zależność:

implementation 'com.android.volley:volley:1.2.1'

Wymagane jest również zdefiniowanie w pliku AndroidManifest.xml zezwoleń na dostęp do informacji o dostępnych sieciach i dostęp do Internetu – są to zezwolenia ACCESS_NETWORK_STATE oraz INTERNET.

Połączenie z API zostanie omówione na przykładzie uzyskania aktualnej listy krajów z bazy danych w celu implementacji samouzupełniania pola tekstowego na wprowadzenie nazwy kraju na interfejsie do wprowadzania danych przez gościa.

W pierwszej kolejności wymagane jest utworzenie obiektu typu `RequestQueue` – jest to kolejka przechowująca zgłoszenia do API i zajmująca się odpowiedziami na nie.

Ponieważ zdefiniowane API zwraca dane w postaci obiektów JSON należy utworzyć zgłoszenie o obiekt JSON – jako argumenty należy podać metodę zgłoszenia do API (post lub get), adres URL API oraz zachowanie w zależności od powodzenia lub niepowodzenia zgłoszenia. Dodatkowo można przekazać w zgłoszeniu inny obiekt JSON jako parametr.

Kody 11, 12 oraz 13 przedstawiają fragment pliku `UserFragment.kt` znajdującego się w katalogu `/mobile_app/app/src/main/java/com/iotproject/hotel/`

```
private fun getJsonDataFromApi(volleyListener: VolleyListener){
    val urlCountries = "http://192.168.0.66:8080/api/countries"
    requestQueue = Volley.newRequestQueue(requireContext())
    val jsonObjectRequest = JsonObjectRequest(
        Request.Method.POST, urlCountries, jsonRequest: null,
        { response -> parseJson(response, volleyListener) },
        { error -> error.printStackTrace() }
    )
    requestQueue?.add(jsonObjectRequest)
}
```

Kod 11. Utworzenie kolejki zgłoszeń i zgłoszenia o otrzymanie listy krajów z bazy danych

Dane do uzupełniania pola tekstowego powinny zostać przekazane jako lista, dlatego konieczna jest konwersja otrzymanego z API obiektu JSON na listę tekstów. Jest to elementarna metoda, która wykonywana jest w momencie powodzenia zgłoszenia do API zgodnie z liniijką:

{ response -> parseJson(response, volleyListener) }

```
private fun parseJson(response: JSONObject, volleyListener: VolleyListener){
    for (i in 1 until response.length()) {
        countriesList.add(response.getString(i.toString()))
    }
    volleyListener.onResponseReceived()
}
```

Kod 12. Przeczytanie listy tekstów z obiektu JSON

Kolejnym krokiem jest zdefiniowanie słuchacza nasłuchującego na powodzenie zgłoszenia do API – po otrzymaniu z API obiektu JSON i uzupełnieniu listy wykorzystywany jest adapter w celu podpięcia listy pod pole tekstowe. Z tej listy pobierane są wartości, które są podpowiadane użytkownikowi.

Lista jest uzupełniana tylko w przypadku powodzenia zgłoszenia do API. Jeśli zgłoszenie zakończy się błędem, nie zostanie wywołana metoda słuchacza.

```

val inputCountry: TextInputLayout = view.findViewById(R.id.countryTextField)

val volleyListener: VolleyListener = object : VolleyListener {
    override fun onResponseReceived() {
        val countryAdapter = ArrayAdapter(requireContext(), R.layout.country_item, countriesList)
        (inputCountry.editText as? AutoCompleteTextView)?.setAdapter(countryAdapter)
    }
}

getJSONDataFromApi(volleyListener)

```

Kod 13. Utworzenie adaptera dla listy i pola tekstowego w metodzie słuchacza

Aby umożliwić dodanie nowego gościa do bazy danych wykorzystano tę samą bibliotekę do komunikacji z API – główną różnicą jest to, że do tworzonego zgłoszenia jako argument należy podać obiekt JSON z danymi gościa przekazywanymi do API.

Rysunek 17. Przykładowe dane nowego gościa

```
MariaDB [hotel]> SELECT * FROM Guests;
```

id	name	surname	document_no	phone_no	email	address	zip_code	city	FK_country
1	John	Doe	ABC123456	+44 123456789	john.doe@gmail.com	55 Gresham St	E1 7AY	London	20
2	Jan	Kowalski	DEF346293	+48 948374348	jan.kowalski@cybernet.com.pl	Strumykowa 15	51-348	Wroclaw	15
3	Karolina	Kowalska	WEC473028	+48 452838457	k.karolina@gmail.com	Powstancow Slaskich 78/2	34-324	Warszawa	15
4	Beth	Harmon	DFC354282	+1 679123644	harmon.beth@gmail.com	445 Park Ave	10022	New York	21
5	Enzo	Cedrone	VND232134	+39 594243484	cedrone.e@gmail.com	Via dei Girasoli	00172	Roma	13
6	Sherlock	Holmes	SH221	+44221221221	sherlock.holmes@gmail.com	221B Baker Street	NW1 6XE	London	20

6 rows in set (0.01 sec)

Screen 5. Nowy gość został dodany do bazy danych – ostatni wiersz w tabeli Guests

Podobnie odbywa się zameldowanie gościa w pokoju za pomocą otrzymanego od obsługi hotelu tokena (w poniższym przypadku o wartości 1). Jeśli operacja przebiegnie pomyślnie w tabeli CheckIns pojawi się nowy rekord.

```
MariaDB [hotel]> SELECT * FROM checkins;
```

id	FK_room	FK_guest	FK_rfid	validSince	validUntil
1	1	1	1	2022-01-01 18:25:34	2022-01-10 19:45:38
2	1	2	3	2021-12-10 06:17:23	2021-12-20 10:05:45
3	2	1	3	2022-01-07 21:56:19	NULL
4	4	5	2	2022-01-06 17:18:59	NULL
5	3	4	4	2022-01-04 18:12:20	NULL
6	1	6	1	2022-01-23 20:14:36	NULL

Screen 6. Nowy rekord w tabeli CheckIns - gość o id = 6 (Sherlock Holmes) został przypisany do pokoju nr 1

Aplikacja webowa

W celu implementacji tabel wykorzystano biblioteki bootstrap [\[4\]](#).

Aplikacja webowa jest wykorzystywana między innymi w celu wyświetlenia danych dotyczących gości w hotelu. Wyszukiwanie informacji na temat gości zostało zaimplementowane jako strona główna aplikacji webowej – rysunek 18.

Home Room info Search guests Assign RFID to room

Rysunek 18. Ekran początkowy

Po wpisaniu przez administratora nazwiska gościa hotelowego i wciśnięciu “Search” pojawiają się dane osoby o podanym nazwisku. Wyszukiwanie gościa po nazwisku zaprezentowano na rysunkach 19 oraz 20.

Home Room info Search guests Assign RFID to room

ID	Name	Surname	RFID	Phone	Email
2	Jan	Kowalski	3326382912	948374348	jan.kowalski@cybernet.com.pl

Rysunek 19. Dane osoby o podanym nazwisku wyświetlane są w tabeli

Home Room info Search guests Assign RFID to room

ID	Name	Surname	RFID	Phone	Email
No matching records found					

Rysunek 20. Gdy wśród gości hotelowych nie ma osoby o podanym nazwisku wyświetlana jest informacja o braku takich rekordów

Możliwe jest również wyszukanie osoby aktualnie mieszkającej w pokoju o podanym numerze. Wtedy zamiast nazwiska należy podać numer pokoju jako daną w polu tekstowym obok przycisku "Search". Wyszukiwanie gościa po numerze pokoju, w którym mieszka, zaprezentowano na rysunkach 21 oraz 22.

Home Room info Search guests Assign RFID to room

ID	Name	Surname	RFID	Phone	Email
1	John	Doe	3326382912	123456789	john.doe@gmail.com

Rysunek 21. Dane osoby mieszkającej w pokoju o podanym numerze wyświetlane są w tabeli

Home Room info Search guests Assign RFID to room

ID	Name	Surname	RFID	Phone	Email
No matching records found					

Rysunek 22. Gdy wśród gości hotelowych nie ma osoby mieszkającej w danym pokoju wyświetlana jest informacja o braku takich rekordów

Do powiązania do powiązania gościa z pokojem, w którym będzie się zatrzymywał, i kartą RFID, której będzie używał do otwarcia drzwi do swojego pokoju służy token, który jest generowany przez pracownika za pomocą aplikacji webowej. Proces generowania tokena zaprezentowano na rysunkach 23 oraz 24.

Home Room info Search guests Assign RFID to room

5

2344832389 Save

Choose RFID

2344832389

3820433859

3326382912

Rysunek 23. Generowanie tokenu, którego gość wykorzysta w celu zameldowania się do konkretnego pokoju

Home Room info Search guests Assign RFID to room

5

2344832389 Save

Token for guest: 32945

Rysunek 24. Po wygenerowaniu tokenu jest on wyświetlany pracownikowi, który następnie może przekazać go gościowi

Opis wkładu pracy każdego z autorów

Bogusława Tłołka 254505

- Wymagania нефункционалне (90%)
- Mockupy interfejsu dla aplikacji mobilnej
- Interfejsy w aplikacji mobilnej
- Nawigacja w aplikacji mobilnej
- Połączenie z API w aplikacji mobilnej – pobieranie listy krajów, dodanie gościa do bazy danych, zameldowanie w pokoju przy pomocy tokena, zmiana statusu karty (zablokowanie), wymeldowanie z hotelu, wyświetlenie numeru pokoju
- Opis działania i prezentacja interfejsu aplikacji mobilnej w dokumentacji (90%)
- Konfiguracja TLS/SSL (skrypt bashowy) (50%)

Aleksandra Stecka 254584

- Cel / Sformułowanie problemu
- Wymagania funkcjonalne
- Projekt relacyjnej bazy danych
- Diagram struktury logicznej bazy danych
- Skrypt generujący bazę danych
- Skrypt generujący dane testowe do bazy danych
- Definicja fragmentów w aplikacji mobilnej
- Konfiguracja szyfrowania i uwierzytelniania TLS/SSL
- Opisy dockera (50%), bazy danych, brokera MQTT, API, działania i prezentacja interfejsu aplikacji mobilnej (10%), działania i prezentacja interfejsu aplikacji webowej (10%), procesu łączenia się z API przez aplikację mobilną w dokumentacji
- Podsumowanie pracy nad projektem
- Instrukcja instalacji i uruchomienia aplikacji mobilnej
- Opracowanie, korekta i formatowanie dokumentacji według szablonu

Paweł Walkowiak 254513

- Narzędzia
- Wymagania нефункционалне (10%)
- Projekt relacyjnej bazy danych
- Konfiguracja serwera http (nginx) z reverse proxy, obsługa CORS
- REST API (fastapi), do komunikacji z aplikacją mobilną i webową
- Komunikacja z bazą danych przez klienta mariadb
- Konfiguracja kontenera bazy danych (mariadb)
- Diagram rozmieszczenia
- Konfiguracja brokera MQTT, nasłuchiwanie na wiadomości od czujników i odpowiadaniu im korzystając z bazy danych
- Stworzenie przykładowego klienta symulującego bramkę z czujnikiem RFID
- Opisy dockera (50%), architektury systemu w dokumentacji

- Instrukcja instalacji i uruchomienia serwera aplikacji

oraz przykładowego klienta (bramki)

Joanna Wdziękońska 254515

- Mockupy interfejsu dla aplikacji webowej
- Utworzenie tabel wynikowych bazujących na bibliotekach bootstrap
- Wykonanie podstron html dla aplikacji webowej z wykorzystaniem bootstrap
- Opis działania i prezentacja interfejsu aplikacji webowej (90%) w dokumentacji
- Połączenie z API w aplikacji webowej
- Konfiguracja TLS/SSL (skrypt bashowy) (50%)
- Logika biznesowa związana z aplikacją webową w javascriptcie (wyświetlenie tokenu dla gościa, informacje o błędach, interakcyjne pojawianie się danych na stronie www)

Podsumowanie

Zrealizowano projekt w terminie oraz w pełnym zakresie. Do wymieniania się kodem wykorzystano system obsługi wersji git GitHub dzięki czemu równoległa praca nad kodem nie sprawiła żadnych problemów.

Wszyscy członkowie zespołu pracowali nad projektem sumiennie i z zaangażowaniem. Zespół był zgrany i nie wystąpiły żadne problemy natury międzyludzkiej (kłótnie, nieprzyjemne sytuacje) ani organizacyjne (nie trzymanie się terminów, brak zaangażowania). Wszyscy członkowie zespołu byli gotowi wprowadzić zmiany do swojej części projektu lub coś uzupełnić, jeśli zostali o to poproszeni, oraz chętnie pomagali sobie nawzajem.

Ocena zgodności projektu z wymaganiami

Zrealizowano wszystkie wymagania funkcjonalne. Trzymano się również podziału funkcjonalności na funkcjonalności aplikacji mobilnej oraz webowej – wszystkie funkcjonalności zostały zaimplementowane na odpowiedniej aplikacji. Gość hotelu jest w stanie dodać swoje dane do bazy danych za pomocą aplikacji mobilnej korzystającej z udostępnianego API. Jest w stanie również podać token otrzymany od pracownika obsługi, wymeldować się z hotelu oraz, w razie potrzeby, zablokować kartę RFID, którą się posługuje. Pracownik administracyjny ma możliwość wykonania wszystkich akcji administracyjnych – wyświetlenia informacji o gościach, dodania nowych kart RFID oraz generowania tokenów dla gości.

Do zabezpieczenia komunikacji MQTT wykorzystano protokół TLS. Aplikacja mobilna działa na urządzeniach z systemem Android oraz, tak jak aplikacja webowa, jest dostępna w języku angielskim. Zrealizowano również pozostałe wymagania niefunkcjonalne.

Uwagi dotyczące napotkanych trudności

Największą trudność sprawiła implementacja uwierzytelniania TLS. Część błędów wynikała z prostych pomyłek – na przykład podpisano certyfikat klienta nie tym certyfikatem CA, co trzeba było, więc broker odrzucał połączenie – i te błędy dość szybko zostały rozwiązane. Większą trudność sprawił fakt, że broker odrzucał połączenie z uwagi na to, że jego certyfikat CA nie był zaufany. Na szczęście z uwagi na to, że broker został zaimplementowany na Linuxie Alpine, dodanie certyfikatu CA do listy certyfikatów zaufanych było bardzo proste – rozwiązanie jest opisane w części dokumentacji dotyczącej implementacji brokera.

Pewne trudności sprawił również Docker – z jednej strony stawianie wszystkich serwisów na Dockerze było bardzo dobrym pomysłem, ponieważ każdy członek zespołu mógł lokalnie postawić wszystkie serwisy i mieć do nich bezpośredni dostęp. Z drugiej jednak strony te same skrypty i pliki dockerfile działały nieco inaczej u różnych członków zespołu. Nie jesteśmy w stanie wytłumaczyć, dlaczego u niektórych członków zespołu po implementacji uwierzytelniania z hasłem i certyfikatem w logach brokera widać było, że klient symulujący bramkę i klient

łączący się z bazą danych się połączyły, ale klient łączący się z bazą danych nie odpowiadał klientowi stymulującemu bramkę. U innych członków zespołu te same serwisy i te same skrypty działały poprawnie.

W początkowej wersji projektu zakładano implementację szyfrowania poufnych danych użytkowników w celu zwiększenia bezpieczeństwa systemu. Zrezygnowano z tego pomysłu głównie z powodu chęci zwiększenia klarowności danych w bazie danych, również na czas prezentacji.

Nie napotkano innych znacznych trudności, których nie udało się rozwiązać.

Propozycja zmian lub rozbudowy projektu

- Dodanie logowania do aplikacji mobilnej w celu ochrony danych gościa oraz ochrony dostępu do akcji blokowania karty i wymeldowania się.
- Dodanie logowania do aplikacji webowej w celu ochrony danych i akcji, które powinny być dostępne tylko dla pracowników administracyjnych hotelu.
- Implementacja funkcjonalności dla gościa również na aplikacji webowej.
- Implementacja autoryzowanego dostępu do API w celu ochrony takich operacji jak dodawanie gościa do bazy danych, blokowanie karty RFID, wymeldowywanie się z hotelu.
- Zabezpieczenie API przed SQL injection oraz innymi atakami poprzez podanie niebezpiecznych danych.

Literatura

- [1] – <https://www.openssl.org/docs/man1.1.1/man1/>
- [2] – <https://fastapi.tiangolo.com/tutorial/>
- [3] – <https://developer.android.com/training/volley>
- [4] – <https://getbootstrap.com/docs/5.1/examples/>

Aneks

https://github.com/PawelWal/IoT_Project.git