

Projekt Internet of Things

Smart Room



Natalia Pyrzyk, Weronika Opyrchał, Marcin Gądek, Łukasz Kras,
Anna Dziarkowska, Kamil Czernek, Dawid Worek, Liliana Rojas, Kamil Sternal

Spis Treści

Wstęp	2
1. Sprzęt	3
1.1 Wstęp	3
1.2 Zastosowane czujniki	5
1.2.1 Czujniki PIR - czujniki ruchu na podczerwień	5
1.2.2 Czujnik PMS5003ST – jakość powietrza, temperatura i wilgotność na zewnątrz	6
1.2.3 Czujnik APDS9960 – czujnik natężenia światła wewnątrz pokoju	6
1.2.4 Czujnik DHT22 – czujnik temperatury i wilgotności wewnątrz pokoju	7
1.2.5 Czujnik CCS811 - czujnik jakości powietrza wewnątrz pokoju	7
1.2.5 Czujniki dźwięku	7
1.2.6 Sterowanie klimatyzacją, oknami, żaluzjami i oświetleniem	8
1.3 Komunikacja z serwerem	8
1.4 Przypisy	8
2.Serwer	9
3. Aplikacja	12
3.1 Scenariusz	12
4. Komunikacja	21
4.1 Komunikacja REST API (Admin)	21
4.2 Komunikacja REST API (Guest)	24
4.3 Komunikacja MQTT	26

Wstęp

Projekt *Smart Room* powstał z myślą o stworzeniu, zgodnie z nazwą, pokoju tzw. inteligentnego. Jego możliwości to pomiar temperatury zewnętrznej oraz wewnętrznej, smogu, wilgotności oraz liczby osób w pokoju. Dodatkowo umożliwia kontrolę nad temperaturą oraz natężeniem światła. Projekt ten składa się z czujników będących urządzeniami pomiarowymi, aplikacji, serwera oraz bazy danych.

Głównym elementem dla użytkownika do korzystania z naszego projektu jest aplikacja działająca na systemach Android. W niej możliwe jest podejmowanie decyzji oraz odczytywanie pomiarów. Jej komunikacja z serwerem, a także komunikacja serwera z urządzeniami następuje przy wykorzystaniu MQTT oraz RestAPI. Zapewnia nam to stały, płynny przepływ danych pomiędzy wszystkimi elementami projektu. Poniższa dokumentacja opisuje dokładne działanie każdego z nich.

1. Hardware

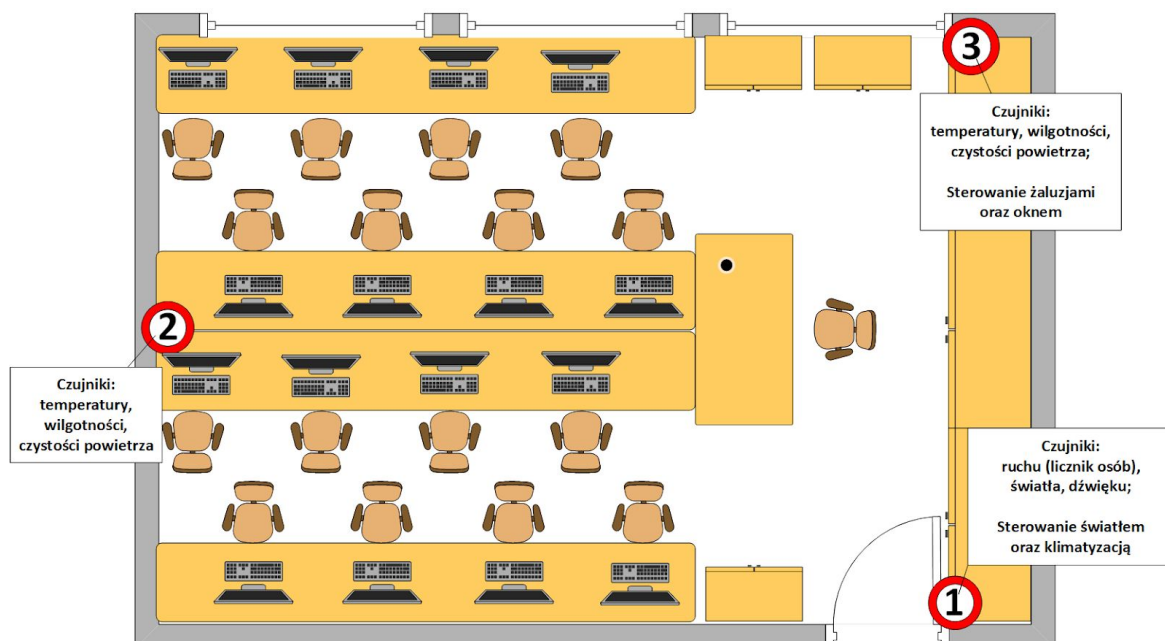
1.1 Wstęp

Część sprzętowa realizowała następujące zadania:

- odczyt danych z czujników parametrów środowiskowych (temperatura, wilgotność, jakość powietrza, a wewnątrz także natężenie światła)
- sterowanie osprzętem zamontowanym w pokoju: żaluzjami, klimatyzacją oraz oświetleniem
- przesyłanie danych z czujników do serwera
- liczenie osób przebywających w pokoju za pomocą czujników ruchu
- odbiór ustawień osprzętu od serwera

Wybrano platformę Raspberry Pi 3B+. Zainstalowano tam system Raspbian – standardową dystrybucję dla tej platformy zawierającą biblioteki przydatne przy obsłudze czujników poprzez takie interfejsy jak I²C czy UART. Szczególnie przydatna była również biblioteka pozwalająca na manipulowanie stanami poszczególnych wyprowadzeń procesora oraz odczyt ich stanów. Dodatkowo wszystkie biblioteki posiadają API do języka Python, które ze względu na szybkość implementacji, bardzo rozbudowaną społeczność oraz ilość przykładów został wybrany jako język, w którym zostało napisane oprogramowanie tej części projektu. Sam język był zresztą od początku powstania platformy Raspberry Pi wybrany jako główny język pisania aplikacji na nią.

Aby spełnić wszystkie wymagania zdecydowano się użyć 3 sztuki Raspberry Pi umiejscowionych w 3 różnych miejscach pokoju.



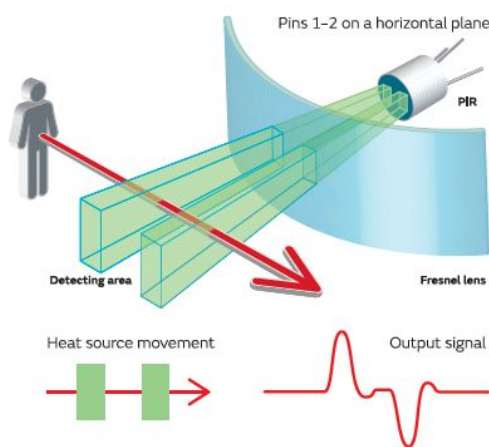
Raspberry Pi oznaczone numerem 1 korzystając z dwóch czujników ruchu PIR szacuje liczbę osób znajdującą się w pokoju. Jako że jest najbliżej “puszki” z przyciskami do światła i klimatyzacji, steruje więc tym osprzętem. Korzystając dodatkowo z czujnika dźwięku pozwala na potwierdzenie obecności jakiejkolwiek osoby w pokoju. Moduł został wyposażony także w czujnik światła zbierający informację o jego natężeniu wewnątrz sali. Numer 2 znajduje się na środku pokoju i do niego podłączone są czujniki mierzące aktualny stan środowiska w pokoju: temperaturę, wilgotność oraz jakość powietrza. Numer 3, umiejscowiony przy oknie, steruje żaluzjami i oknem oraz mierzy stan powietrza, temperaturę oraz wilgotność na zewnątrz.

Wszystkie Raspberry Pi mają połączenie z Internetem, tak aby móc raportować serwerowi dane zebrane z czujników oraz przyjmować od niego wymagania dotyczące ustawienia klimatyzacji, oświetlenia czy stanem okna.

1.2 Zastosowane czujniki

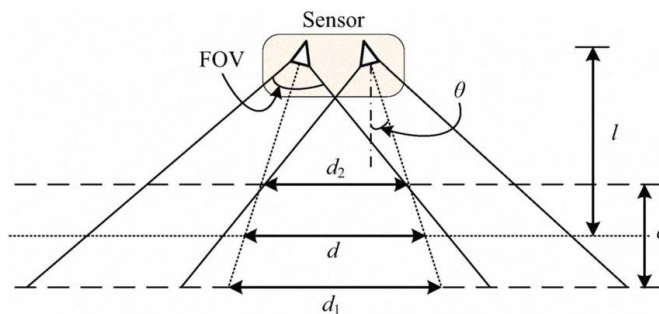
1.2.1 Czujniki PIR - czujniki ruchu na podczerwień

Do obliczenia i wykrywania wejścia i wyjścia ludzi na zewnątrz i do wewnątrz wybrano parę czujników PIR. Pozwalają one na wykrywanie ruchu. Wykrywanie ruchu w urządzeniach tego typu bazuje na zmianie promieniowania podczerwonego, które emitowane jest przez obiekty o temperaturze wyższej od otoczenia. Sensor zasilany jest napięciem z zakresu 4,5 V do 20 V, posiada zasięg do 7 m. Wykrycie obiektu sygnalizowane jest stanem wysokim.



Rys. Działanie czujnika PIR

Na powyższym zdjęciu jest pokazane jak działa PIR. Wiemy, że wykrywa ruch w jednym kierunku (wejście lub wyjście), ale by PIRy wykrywały wejście i wyjście ludzi jest wymagana poniższa konfiguracja, która była zaimplementowana w tym projekcie.



Rys. Fizyczne rozmieszczenie dwóch czujników PIR

Są zastosowane dwa PIRy, które “patrzą” na dwie przeciwne strony. Dzięki kolejności w jakiej oba czujniki się aktywują wiadomo czy ktoś wszedł czy wyszedł z pokoju.

1.2.2 Czujnik PMS5003ST – jakość powietrza, temperatura i wilgotność na zewnątrz

Do pomiaru parametrów środowiskowych na zewnątrz wybrano czujnik PMS5003ST. Jest on w stanie pracować w zakresie temperatur od -10 do +60 stopni, i w wilgotności od 0 do 99 procent. Czujnik po włączeniu automatycznie zaczyna przysyłać co sekundę przez port szeregowy pomiary. Jakość powietrza jest oceniana na podstawie stężenia pyłów PM2.5. Według europejskiej normy nie powinno ich być więcej niż 25 μg na m^3 [1] (aczkolwiek od roku 2020 ma to być 20 μg na m^3 [2]). Następnie jest to przeliczane na procenty za pomocą funkcji:

```
def _pm25_to_air_quality(self, pm25_level):  
    level_in_percent = 1 - (pm25_level / 25)  
    if level_in_percent < 0:  
        level_in_percent = 0  
    return level_in_percent
```

Dzięki temu użytkownikowi końcowemu łatwiej jest ocenić jakość powietrza na zewnątrz niż patrząc na bezwzględną wartość stężenia pyłów.

1.2.3 Czujnik APDS9960 – czujnik natężenia światła wewnątrz pokoju

Układ APDS9960 integruje w sobie kilka funkcjonalności: czujnik natężenia światła, czujnik zbliżeniowy (na podstawie pomiarów natężenia światła), oraz detekcja gestów (przesunięcie palcami w powietrzu wzdłuż albo w szerz czujnika). W naszym projekcie tylko ta pierwsza była wykorzystywana, czyli monitorowanie natężenia światła. Była ona dla nas priorytetowa, a jak się okazało w trakcie testów, tylko jedna funkcjonalność może być używana w jednym czasie, zaś przełączanie pomiędzy różnymi funkcjonalnościami bardzo komplikowało oprogramowanie oraz, co również sprawdzono podczas testów, nie działało do końca poprawnie.

Przy odczycie pomiarów z czujnika wykorzystano gotową bibliotekę [3]. Ustawienia pozostawiono na domyślne – pomiar natężenia odczytywano z 10 bitową rozdzielczością.

1.2.4 Czujnik DHT22 – czujnik temperatury i wilgotności wewnątrz pokoju

Czujnik DHT22 łączy w sobie dwie funkcjonalności - pomiar temperatury oraz wilgotności powietrza. Posiada szeroki zakres pomiarowy, który określany jest przez producenta jako od -40 do +80 stopni Celsjusza z dokładnością do 0,5 stopnia. Zakres pomiaru wilgotności względnej powietrza mieści się pomiędzy 0, a 100%. Wilgotność względna określa stosunek rzeczywistej wilgoci w powietrzu do jej maksymalnej ilości, którą może utrzymać powietrze w danej temperaturze. Do oprogramowania czujnika wykorzystano bibliotekę dla tego czujnika [5], napisaną w oparciu o język Python. Używając wyżej wymienionej

biblioteki oraz własny funkcji dane pomiarowe zbierane są co najmniejszy możliwy okres czasu 2 sekund.

1.2.5 Czujnik CCS811 - czujnik jakości powietrza wewnątrz pokoju

Czujnik CCS811 został wykorzystany przez nas do określania stopnia czystości powietrza wewnątrz sali. Umożliwia on pomiar stężenia dwutlenku węgla wewnątrz pokoju oraz sumarycznego stężenia lotnych związków organicznych, mogących mieć negatywny wpływ na nasze samopoczucie oraz zdrowie. Wartości pomiarów z czujnika zbierane są z wykorzystaniem magistrali komunikacyjnej I2C, w która wyposażona jest wykorzystywana przez nas platforma Raspberry Pi. Do oprogramowania czujnika wykorzystano stworzoną dla niego bibliotekę [6], w której niektóre funkcje zostały zmodyfikowane na potrzeby naszego projektu. Między innymi zostały stworzone dodatkowe funkcje pozwalające na przetworzenie konkretnych wartości liczbowych pomiarów na punkty procentowe obrazujący w prosty sposób zanieczyszczenie powietrza wewnątrz pomieszczenia.

1.2.5 Czujniki dźwięku

W celu usprawnienia detekcji obecności osób w pokoju zastosowano dodatkowy czujnik dźwięku. Posiadał on wyjście cyfrowe, którego stan logiczny zależał od natężenia dźwięku. Poziom wyzwalania jest ustawiany potencjometrem. Tutaj też pojawił się największy problem: dość trudno było znaleźć takie ustawienie potencjometru, aby czujnik dźwięku się odpowiednio wyzwał, tzn. był na wyjściu stan wysoki jeśli ktoś z pobliżu rozmawiał, a stan niski gdy było cicho. Było to spowodowane prawdopodobnie mechaniczną niedokładnością ustawienia potencjometru. Rozwiązaniem byłoby wyprowadzenie z czujnika sygnału analogowego i jego późniejsza analiza po spróbkowaniu za pomocą układu ADC. Wymagałoby to jednak dużej ingerencji z gotową płytkę (a więc i możliwość jej zepsucia) oraz użycia innego sprzętu niż Raspberry Pi, które nie posiada układu ADC.

1.2.6 Sterowanie klimatyzacją, oknami, żaluzjami i oświetleniem

W powyższym projekcie nie uwzględniamy i wykreślamy szczegółowych połączeń naszych modułów z urządzeniami zewnętrznymi. Na potrzeby testów, zmiana konfiguracji stanu poszczególnych urządzeń takich jak klimatyzacja, oświetlenie czy stan okna, została zasymulowana poprzez zmianę stanu (poziomu napięcia) na przypisanych do tego celu portach wyjściowych platformy.

1.3 Komunikacja z serwerem

Do komunikacji z serwerem wybrano protokół MQTT. Został on zaprojektowany przez firmę IBM specjalnie dla zastosowań Internetu Rzeczy. Zapewnia komunikację w obie strony – klient może wysyłać dane do serwera, a serwer zawiadamia klienta o nowych danych i je przesyła. W naszym projekcie MQTT działa na warstwie transportowej TCP, co zwiększa niezawodność transmisji. Dodatkowo jest prosty w użyciu oraz dostępne są dobrze udokumentowane biblioteki. W projekcie po stronie sprzętu użyto portu biblioteki Paho [4] w języku Python.

1.4 Źródła

- [1] <https://en.wikipedia.org/wiki/Particulates#Regulation>
- [2] <http://powietrze.krakow.pl/pyly-zawieszone-pm10-i-pm2-5/>
- [3] <https://github.com/liske/python-apds9960>
- [4] <https://github.com/eclipse/paho.mqtt.python>
- [5] https://github.com/adafruit/Adafruit_Python_DHT
- [6] https://github.com/adafruit/Adafruit_CCS811_python
- [7] <https://cdn-learn.adafruit.com/downloads/pdf/pir-passive-infrared-proximity-motion-sensor.pdf>

2. Serwer

Platforma serwera jaka została użyta w projekcie to *Amazon Web Services* (AWS).

AWS udostępnia szereg usług w chmurze m.in. moc obliczeniową, hosting baz danych czy usługi dostarczania treści.

Użytym serwerem aplikacji jest *Wildfly* w wersji 14.0.1 wspierającym Java EE8.

Do obsługi protokołu MQTT wybrano oprogramowanie *Eclipse Mosquitto* w wersji 1.5.5. Jest on open-sourcowym brokerem implementującym MQTT.

Wykorzystanym silnikiem bazy danych jest *MariaDB* w wersji 10.3.15. Jest to jeden z najpopularniejszych serwerów bazy danych na świecie. Stanowi ulepszony zamiennik dla MySQL. Używana jest tak chętnie, ponieważ jest szybka, skalowalna i rozrosła. Zawiera w sobie również wiele wtyczek i narzędzi, co czyni ją bardzo uniwersalną dla wielu różnych przypadków. MariaDB jest oprogramowaniem typu *open source* i jako relacyjna baza danych zapewnia interfejs SQL dla danych wejściowych.

Na poniższym rysunku pokazane są używane tabele.

```
MariaDB [iot]> show tables;
+-----+
| Tables_in_iot |
+-----+
| inRoom123      |
| outRoom123     |
| rooms          |
| user           |
+-----+
```

Rys. 2.1 Tabele zawarte w bazie danych

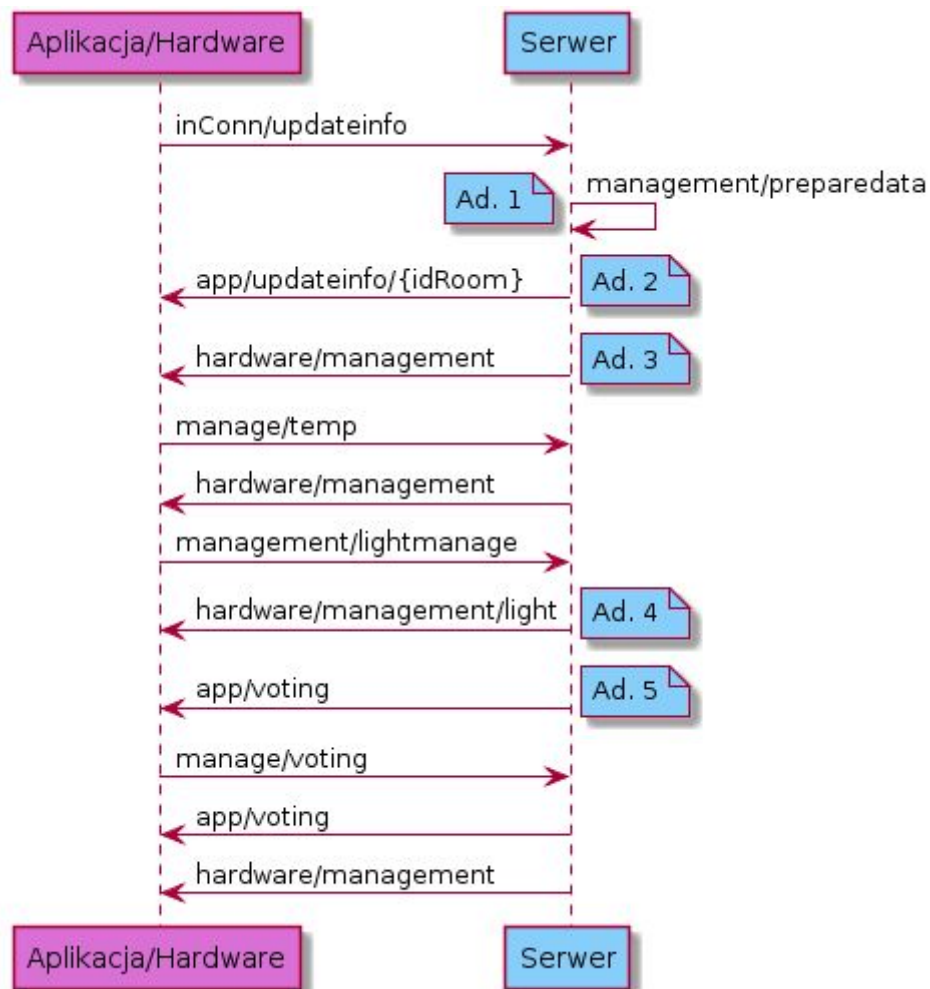
Tabela *inRoom123* zawiera dane otrzymane od hardware. Znajduje się w nich: temperatura i jakość powietrza wewnątrz oraz na zewnątrz, jasność światła, ilość ludzi w środku, datę wpisu i informacje: czy klimatyzacja jest włączona, czy okno jest otwarte oraz czy wykryto dźwięk.

Tabela *outRoom123* ma taki sam rodzaj danych jak *inRoom123*, ale po uśrednieniu kilku rekordów otrzymanych od hardware w pewnym przedziale czasowym. Tak przerobione dane są wysyłane do aplikacji.

Tabela *rooms* posiada informacje o id pokoiów i hasłach do nich. Wykorzystywana przy wyświetlaniu listy dostępnych pokoi.

Tabela *user* przechowuje dane o nazwie, hasle i tokenie użytkowników będących administratorami. Używana podczas logowania się jako administrator. Wpisany login i hasło porównywane są z danymi zawartymi w tej tabeli.

Poniżej znajduje się rysunek obrazujący ogólne *flow* serwera.



Rys. 2.2 Flow serwera

1. [Ad. 1] Serwer po otrzymaniu danych w formie *jsona* konwertuje je oraz przygotowuje wpis do bazy danych. Dane zostają wpisane do zazy danych oraz zostaje wysłany request z id pokoju na wewnętrzny topic o przygotowanie danych.

2. [Ad. 2] Pobrany zostaje z bazy ostatni bądź ostatnie niezerowe wpisy. Dane te zostają uśrednione i zapisane do tabeli aplikacji dla pokoju oraz przekazanie tych danych do aplikacji.
3. [Ad. 3] Dla danych z poprzedniego punktu zostaje sprawdzony stan klimatyzacji (czy jest włączona), okna (czy jest otwarte) oraz temperatury (ile jest stopni).
4. [Ad. 4] Sprawdzany jest stan światła (czy jest włączone i z jaką intensywnością świeci).
5. [Ad. 5] Podczas głosowania, głosy zostają zapisane oraz podliczone, a następnie odsyłane odpowiednie polecenia do sprzętu zgodnie z wynikami głosowania.

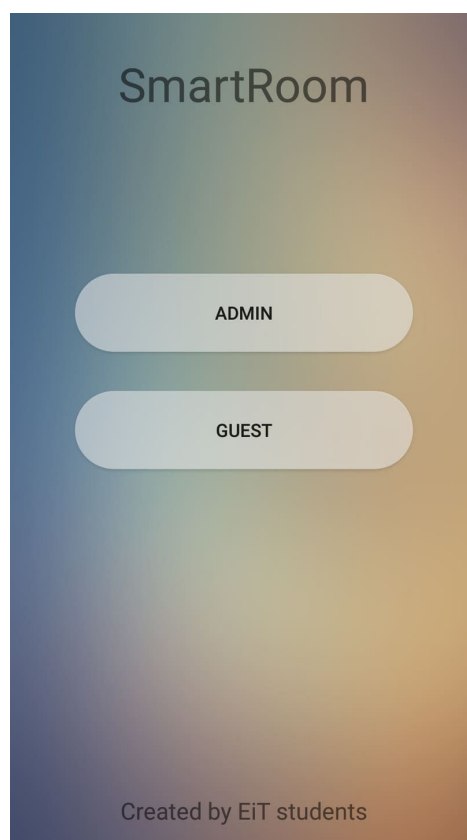
3. Aplikacja

Celem jaki został postawiony, było stworzenie takiej aplikacji, która jest prosta i jasna w obsłudze, a dodatkowo wygląda estetycznie i jest intuicyjna.

Zaimplementowano serię otwierających się kolejno okienek, tworząc odpowiedni scenariusz działania, zgodny z założeniami projektu.

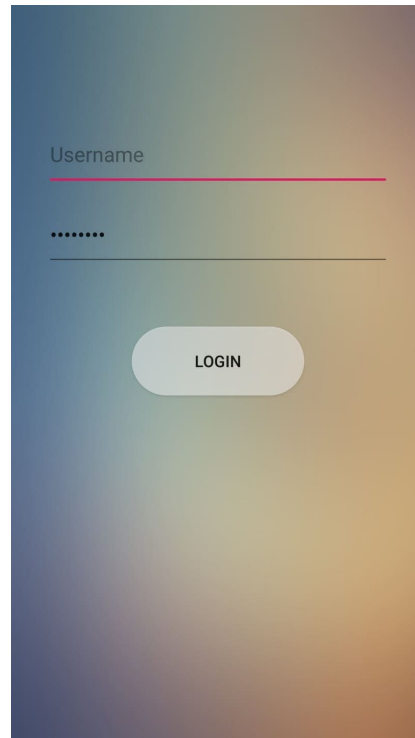
3.1 Scenariusz

Po uruchomieniu, pierwszym oknem, które widzimy jest okno pozwalające na wybór czy będziemy korzystać z aplikacji jako gość, czy jako administrator. Odpowiedni plik xml w kodzie nosi nazwę "**activity_first_window.xml**", a odpowiadająca mu aktywność to plik "**FirstWindow.java**"



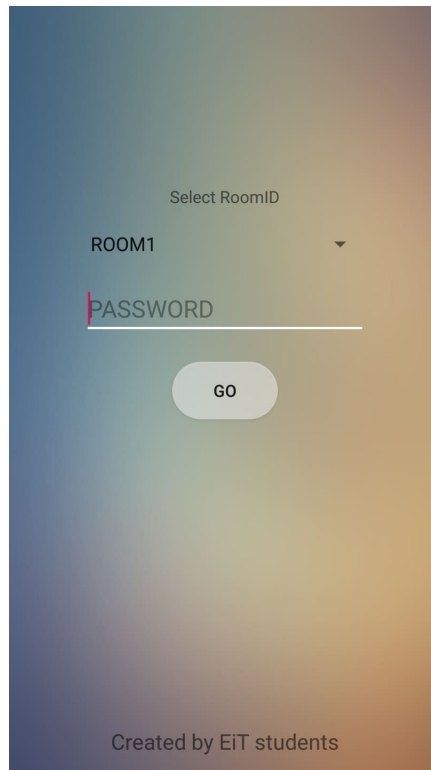
Rys.3.1: Pierwsze okno aplikacji z wyborem trybu

Jeśli wybierzemy tryb administratora, to przechodzimy do ekranu logowania (odpowiednio “**activity_login.xml**” oraz “**LoginActivity.java**”). Wybrane okno prezentuje się jak poniżej:



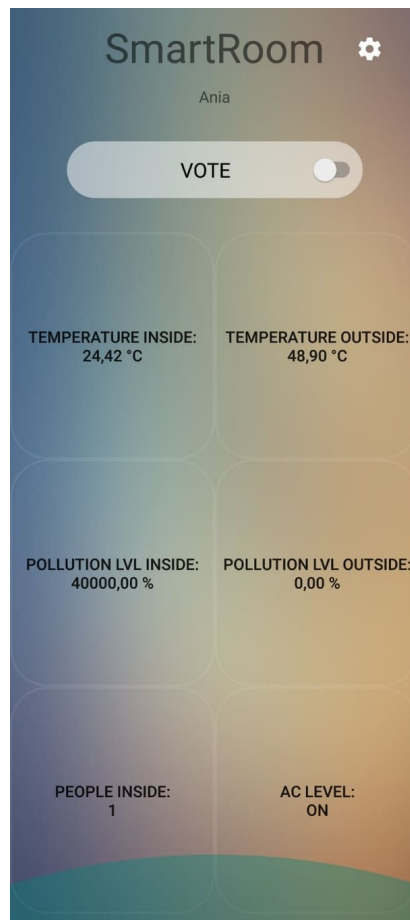
Rys.3.2: Okno logowania dla administratora

Jeśli logowanie przebiegnie pomyślnie, to z serwera pobierana zostaje lista dostępnych pokoi. Następnie otwiera się okno, które umożliwia wybór odpowiedniego pokoju i ustawienie mu hasła dostępu. Odpowiadające tej aktywności pliki w projekcie to “**room_key_activity_for_admin.xml**” oraz “**RoomKeyForAdmin.java**”, a omawiane okno wygląda następująco:



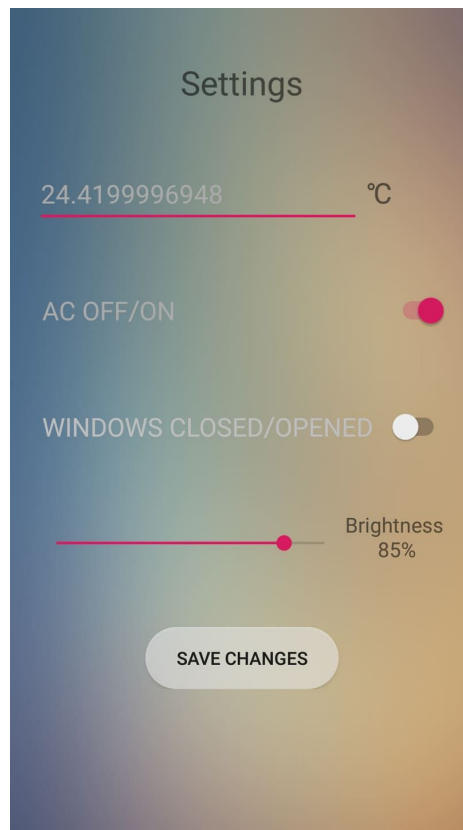
Rys.3.3: Wybór i ustawienie hasła do pokoju, przez administratora

Jeśli proces ten przebiegnie pomyślnie, to administrator ustawi wybrane przez siebie hasło dostępu dla zwykłych użytkowników i przejdzie do ekranu głównego, któremu odpowiadają pliki "**activity_main_screen.xml**" oraz "**MainScreenActivity.java**". Podczas ustawiania hasła pobierane są aktualne dane z serwera, które przenoszone zostają do kolejnego widoku, dzięki czemu administrator nie musi czekać na ich uzyskanie podczas wyświetlania ekranu głównego dla administratora, który przedstawiono na rysunku 4.4.



Rys.3.4: Główny ekran dla administratora

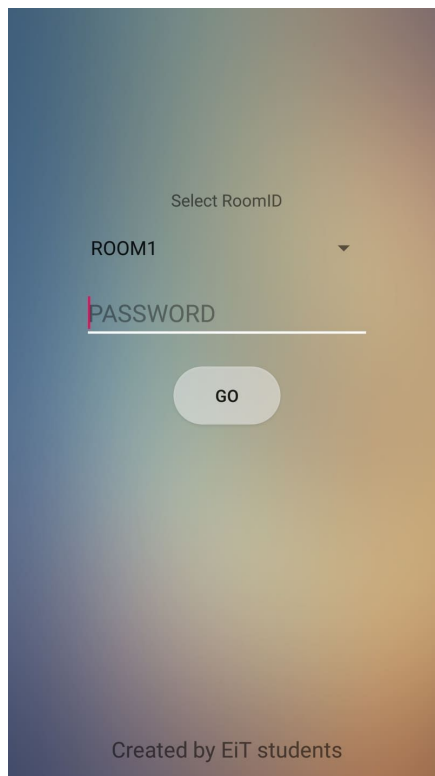
Administrator, podobnie jak gość, może wyświetlać wszystkie mierzone parametry, ustalać głosowanie, oraz zmieniać wybrane ustawienia, takie jak: temperatura, stan klimatyzacji, otwieranie/zamykanie okna, czy jasność regulowana ustawieniem rolet. Funkcjonalności te zawierają się w oknie “Settings”, do którego można trafić klikając ikonę ustawień w prawym górnym rogu. Wyświetlane wartości pobierane są na bieżąco w czasie rzeczywistym z serwera wykorzystując protokół MQTT. Pliki projektu odpowiadające oknie ustawień to: “**activity_settings.xml**” oraz “**SettingsActivity.java**”, a samo okno zaprezentowano na rysunku 4.5.



Rys.3.5: Okno umożliwiające zmianę ustawień w trybie administratora

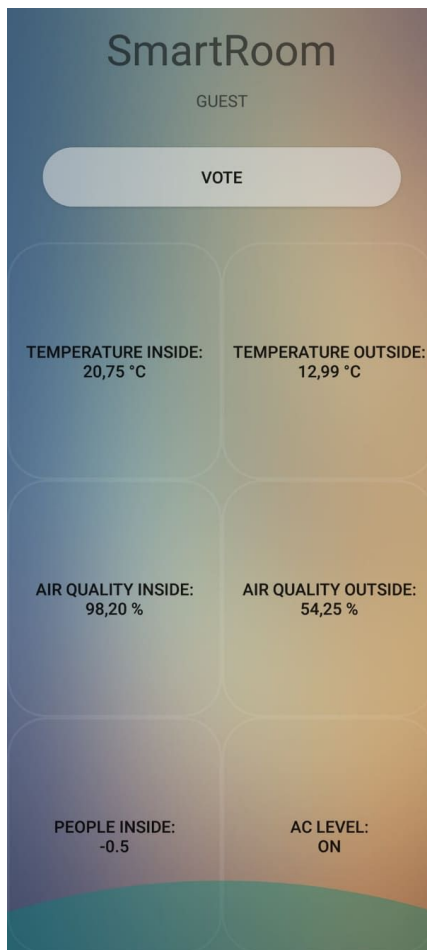
Dodatkowo, gdy otwieramy ustawienia, to wszystkie wartości odpowiadają panującym aktualnie warunkom w pokoju, który wybraliśmy. Wprowadzane dane są sprawdzane pod kątem wartości: temperatura nie może zostać ustawiona poniżej 0 °C lub powyżej 50 °C. Dodatkowo zablokowano możliwość przypadkowego wyboru polegającego na otwarciu okien oraz włączeniu klimatyzacji.

Innym scenariuszem będzie scenariusz gdy w pierwszym oknie wybierzemy tryb zwykłego użytkownika "GUEST". Po dokonaniu takiego wyboru, wyświetla nam się okno umożliwiające uzyskanie dostępu do wybranego pokoju, wpisując odpowiednie hasło. Hasło powinno zostać przekazane użytkownikom przez Administratora, ponieważ to on odpowiada za jego ustawienie. Okno pokazano na rysunku 4.6, a pliki projektu, które mu odpowiadają to "**activity_room_key.xml**" oraz "**RoomKeyActivity.java**"

The image shows a mobile application login screen with a blue-to-orange gradient background. At the top, the text "Select RoomID" is displayed above a dropdown menu showing "ROOM1" with a downward arrow. Below this is a password input field with the placeholder text "PASSWORD" and a red cursor on the left. A white "GO" button is centered below the password field. At the bottom, the text "Created by EiT students" is visible.

Rys.3.6: Logowanie do pokoju dla zwykłego użytkownika

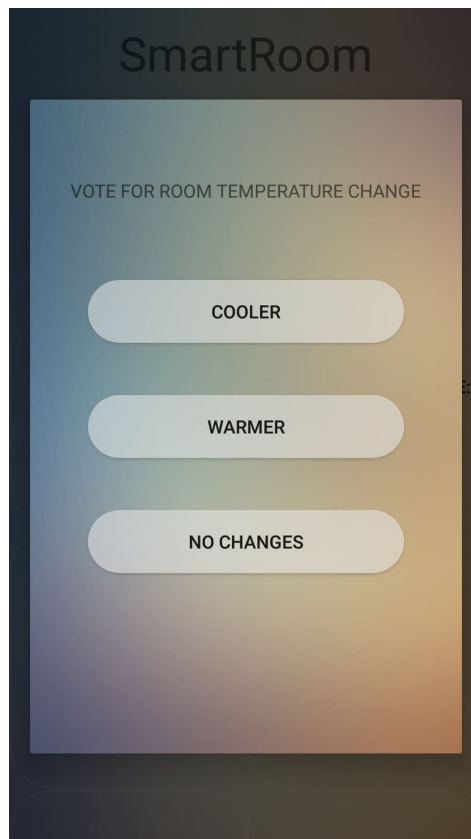
Jeśli logowanie się powiedzie, to przechodzimy do ekranu głównego dla “gościa”. Ekran główny przedstawiono na rysunku 4.7.



Rys.3.7: Główny ekran dla zwykłego użytkownika

Gość ma możliwość oglądania wszystkich dokonywanych w pokoju pomiarów, a także ma możliwość głosowania, w przypadku kiedy nie jest zadowolony z panujących warunków temperaturowych.

Gdy użytkownik zdecyduje się na głosowanie, po kliknięciu przycisku "VOTE" (jeżeli jest dostępny), pojawia się nam okno przeznaczone do głosowania. Do wyboru mamy trzy opcje: "cooler", "warmer", "no changes". Okno przedstawia rysunek 4.8, a odpowiadające mu pliki to "**activity_vote.xml**" oraz "**VoteActicity.java**".

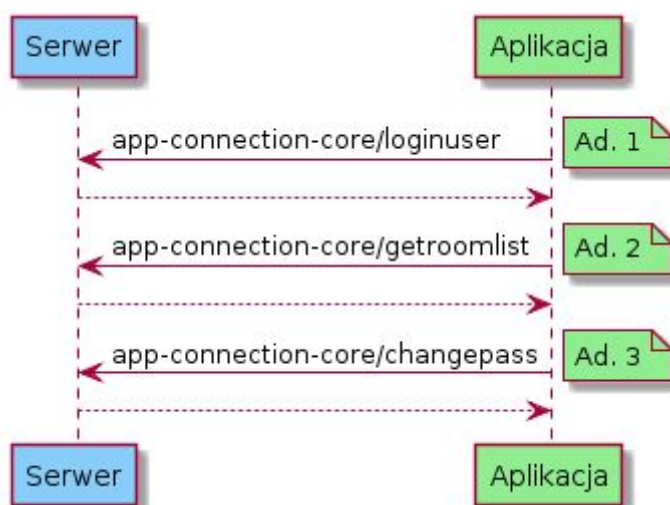


Rys.3.8: Okno odpowiadające za głosowanie na zmianę temperatury

4. Komunikacja

Pierwszym etapem było zaimplementowanie komunikacji w postaci REST API po stronie serwera. Podczas robienia projektu uległa ona zmianie na komunikację poprzez protokół MQTT. W wersji ostatecznej komunikacja Sprzęt - Serwer - Aplikacja wygląda następująco:

4.1 Komunikacja REST API (Admin)



Rys.4.1 Komunikacja Rest API dla administratora

Komunikacja ta odbywa się przy włączaniu aplikacji. Użytkownik aplikacji - w tym wypadku administrator - wywołuje odpowiednie *endpointy*, w celu uzyskania następujących danych:

1. [Ad. 1] Użytkownik wysyła zapytanie ze swoją nazwą oraz hasłem w celu uwierzytelnienia.
Aplikacja wysyła zapytanie w postaci:

```
{
  "user": "Ania",
  "password": "admin123"
}
```

Odpowiedzi serwera na przesłane zapytanie wymienione zostały poniżej:

- a) status kod : 200 - uwierzytelnienie poprawne:

```
{
  "user": "Ania",
  "password": null,
  "token": "przykładowyToken",
  "temp": "24"
}
```

- b) status kod : 401 - brak uwierzytelnienia:

```
{
  "user": "Ania",
  "password": null,
  "token": null,
  "temp": null
}
```

- c) error kod : 404 - czyli problem z konwersją danych
d) error kod : 408 - problem z połączeniem z bazą danych
e) error kod : 500 - inne błędy

2. [Ad. 2] Po poprawnym uwierzytelnieniu, aplikacja pobiera z serwera listę dostępnych pokoi, poprzez wysłanie zapytania na podany adres na rysunku. Serwer może odpowiedzieć w następujący sposób:

- a) status kod 200 - zwraca przykładową listę pokoi:

```
["123","124"]
```

- b) error kod: 500 - brak listy.

3. [Ad. 3] Administrator po wybraniu pokoju może ustawić dla niego nowe hasło. Aplikacja wysyła poniższe zapytanie na podany "endpoint" na rysunku:

```
{  
    "idRoom": "123",  
    "password": "eklektyka"  
}
```

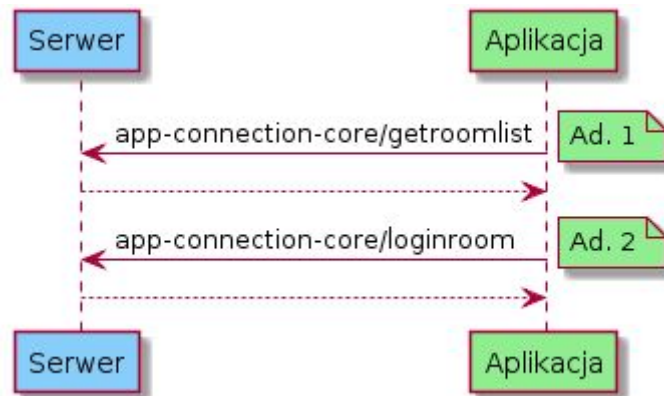
Serwer może odpowiedzieć w następujący sposób:

- a) status 200 - poprawna zmiana hasła, zwraca aktualne dane pomiarowe z danego pokoju:

```
{  
    "id": "123",  
    "timestamp": "2019-06-05 00:51:05",  
    "temp_in": "20.7454",  
    "temp_out": "12.985",  
    "light_in": "0.99",  
    "isClimeOn": "1",  
    "isWindowOpen": "0",  
    "air_qua_in": "0.982",  
    "air_qua_out": "0.54252",  
    "sound_detected": "1",  
    "people_inside": "19"  
}
```

- b) error kod: 500 - brak powodzenia zmiany hasła

4.2 Komunikacja REST API (Guest)



Rys.4.2 Komunikacja Rest API dla gościa

1. Ad. 1

Po zalogowaniu się jako gość, aplikacja pobiera z serwera listę dostępnych pokoi, poprzez wysłanie zapytania na podany adres na rysunku powyżej.

Serwer może odpowiedzieć w następujący sposób:

- a) status kod 200 - zwraca przykładową listę pokoi:

```
[["123","124"]]
```

- b) error kod: 500 - brak listy.

2. Ad. 2

Gość po wybraniu pokoju z listy proszony jest o podanie hasła dostępu.

Aplikacja wysyła poniższe zapytanie na podany "endpoint" na rysunku:

```
{
  "idRoom": "123",
  "password": "eklektyka"
}
```

Serwer może odpowiedzieć w następujący sposób:

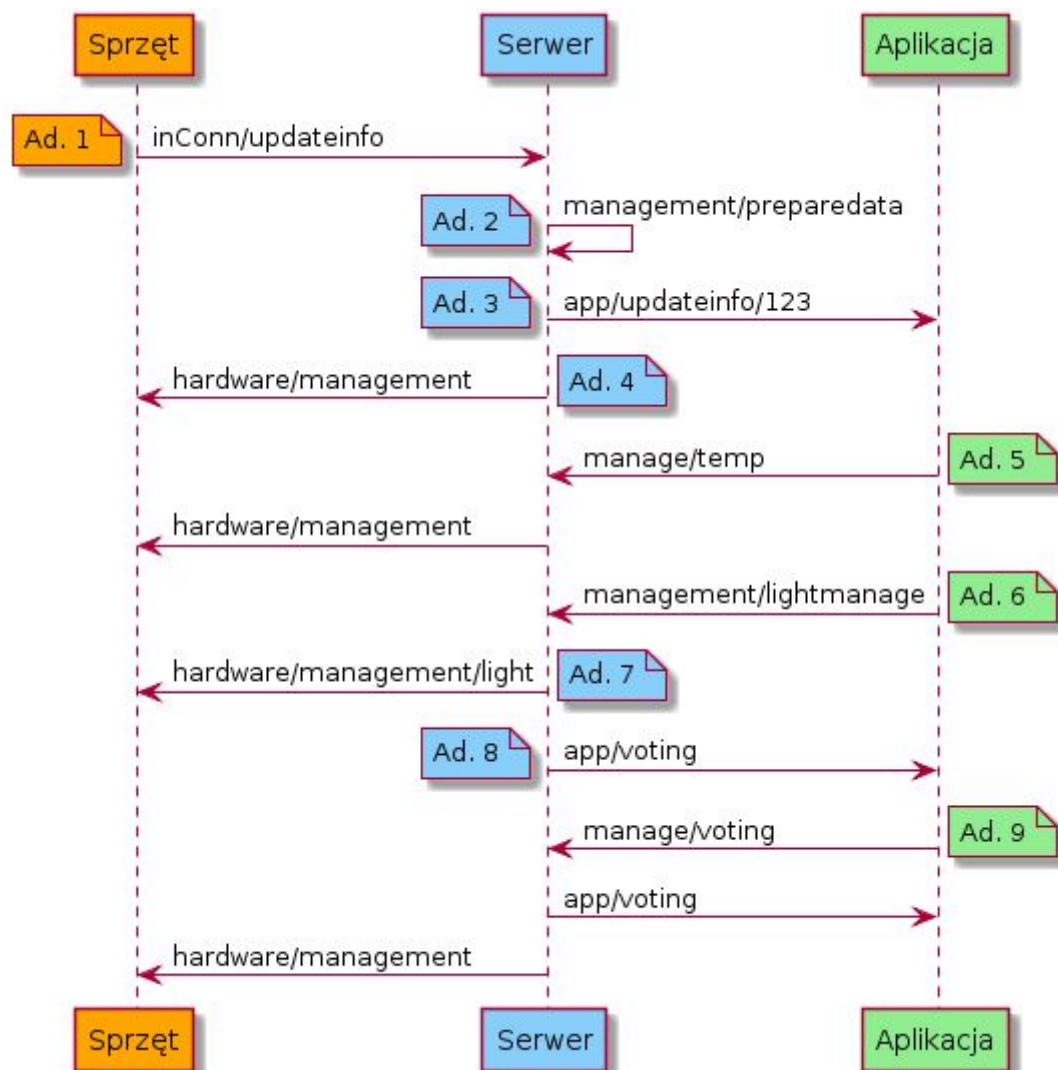
- a) status 200


```
{  
  "id": "123",  
  "timestamp": "2019-06-05 00:51:05",  
  "temp_in": "20.7454",  
  "temp_out": "12.985",  
  "light_in": "0.99",  
  "isClimeOn": "1",  
  "isWindowOpen": "0",  
  "air_qua_in": "0.982",  
  "air_qua_out": "0.54252",  
  "sound_detected": "1",  
  "people_inside": "19"  
}
```

- b) error kod 401 - informacja o błędzie autoryzacji
- c) error kod 500 - brak listy.

4.3 Komunikacja MQTT

Na rysunku przedstawionym poniżej widnieje cała komunikacja przy użyciu protokołu transmisji danych MQTT wraz z nazwami *topic*, na które reaguje serwer, sprzęt i aplikacja.



Rys.4.3 Komunikacja MQTT wraz z nazwami *topiców*.

1. [Ad. 1] Przesyłane zostają aktualne dane od czujników.

```
{
  "id": "123",
  "timestamp": "2019-06-05 00:51:05",
  "temp_in": "20.7454",
  "temp_out": "12.985",
  "light_in": "0.9955453862328321",
  "isClimeOn": "1",
  "isWindowOpen": "0",
  "air_qua_in": "0.982",
  "air_qua_out": "0.54252",
  "sound_detected": "1",
  "people_inside": "19"
}
```

2. [Ad. 2] Odebrane dane są mapowane po stronie serwera oraz zapisywane do bazy.
3. [Ad. 3] Przesyłana zostaje aktualizacja danych do aplikacji.

```
{
  "id": "123",
  "timestamp": "2019-06-05 00:51:05",
  "temp_in": "20.7454",
  "temp_out": "12.985",
  "light_in": "0.9955453862328321",
  "isClimeOn": "1",
  "isWindowOpen": "0",
  "air_qua_in": "0.982",
  "air_qua_out": "0.54252",
  "sound_detected": "1",
  "people_inside": "19"
}
```

4. [Ad.4] Serwer wysyła odpowiedź do sprzętu w postaci takich *jsonów* jak ten przedstawiony poniżej.

```
{
  "id": "123",
  "isClimeOn": "true",
  "isWindowOpen": "false",
  "temp":24
}
```

5. [Ad.5] Od strony aplikacji zostaje wysłane żądanie o zmianę temperatury.

```
{
  "id": "123",
  "isClimeOn": "true",
  "isWindowOpen": "false",
  "temp":24
}
```

6. [Ad.6] Serwer wysła odpowiedź do sprzętu w postaci takich *jsonów* jak ten przedstawiony poniżej z odpowiednimi ustawieniami.

```
{
  "id": "123",
  "light_in": "0.9955"
}
```

7. [Ad. 7] Serwer wysła odpowiedź do sprzętu dotyczącą światła w postaci poniższych *jsonów*:

```
{
  "id": "123",
  "light_in": "0.9955"
}
```

8. [Ad. 8] Do rozpoczęcia lub zakończenia głosowania wysyłane są stringi w postaci "START" lub "STOP".
9. [Ad. 9] Głosujący dokonuje wyboru, po czym aplikacja wysła *json* w poniższej postaci:

```
{
  "id": "123",
  "value": "up"
}
```