Dokumentacja projektu Doorime

Osoby zaangażowane:

1. Natalia Skawińska
2. Konrad Gluza
3. Marcin Dobrucki
4. Mateusz Morawiec
5. Mikołaj Badura
6. Paweł Czaja
7. Szymon Kolber
8. Szymon Lenart

Dokumentacja jest podzielona na cztery części

- Hardware
- Serwerowa
- Baza Danych
- Aplikacji Moblinej

Dokumentacja Hardware

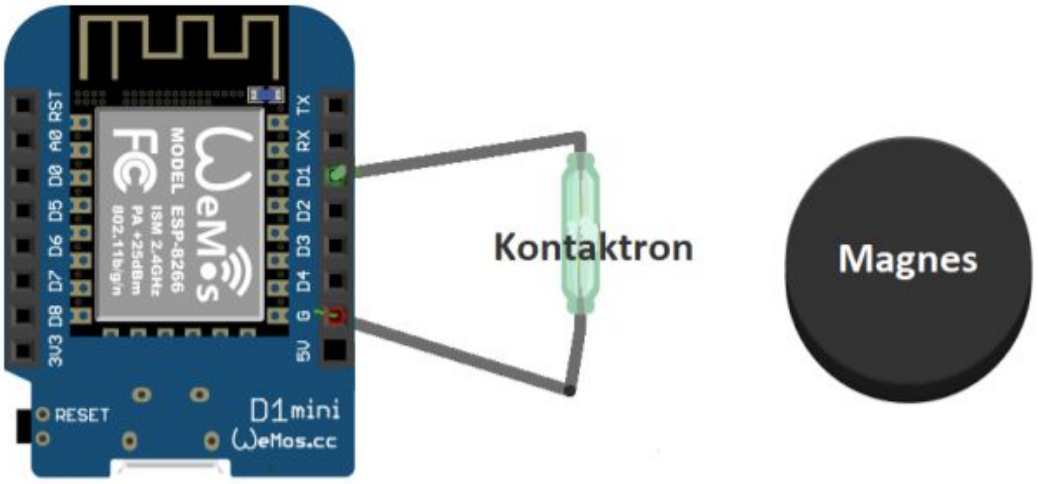
Opis:

1. Sprzęt stanowi źródło informacji wyświetlanych w aplikacji.
2. Zmiana stanu na sprzęcie stanowi trigger do poinformowania użytkownika.
3. Sprzęt powinien się automatycznie łączyć z siecią wifi tworzoną na Raspberry Pi.

Wymagania Techniczne:

1. W przypadku kiedy chcemy na sprzęt zainstalować oprogramowanie - MicroPython, oraz środowisko WebREPL, ze względu na kompatybilność z różnymi systemami operacyjnymi.
2. Urządzenie, z którego korzystamy jest zaprogramowane i od razu gotowe do użycia, nie jest wymagany podpunkt 1 tej sekcji.

Implementacja:



Rys. 1. Obwód elektryczny kontaktronu

Rozwiązanie składa się z modułu Wi-Fi Wemos D1 mini z ESP8266, czujnika otwarcia/zamknięcia (kontaktronu) z gotową obudową, przewodów które dolutowaliśmy oraz z wbudowanego rezystora 100kΩ. W celu dostarczenia zasilania została wykorzystana możliwość zasilania programowo przez pin D1 (GPIO 5). Do tak przygotowanego urządzenia ESP wgrane zostały dwa skrypty odpowiedzialne za jego działanie boot.py, main.py oraz biblioteka mqtt.py niezbędna do komunikacji z serwerem. Wszystkie pliki są dostępne pod tym linkiem - https://github.com/kolberszymon/smartHome.

Dokumentacja części serwerowej

1. **Przygotowanie Centosa 7 na VM node07 do dalszych prac**
    a. Stworzenie kont użytkowników; przypisanie ich do grupy cloudadmin:

```
adduser pczaja
adduser mdobruck
adduser kgluza
chfn -f "Pawel Czaja" pczaja
chfn -f "Marcin Dobrucki" mdobruck
chfn -f "Konrad Gluza" kgluza
groupadd cloudadmin
usermod -g cloudadmin pczaja
groupdel pczaja
usermod -g cloudadmin mdobruck
groupdel mdobruck
usermod -g cloudadmin kgluza
groupdel kgluza
```

    b. Przyznanie wszystkim użytkownikom należącym do grupy cloudadmin praw roota

```
echo "%cloudadmin ALL=(ALL) NOPASSWD: ALL" >> \ /etc/sudoers

UWAGA: lepiej modyfikować za pomocą visudo
```

    c. Instalacja dodatkowych narzędzi

```
yum update && yum upgrade
yum -y install bash-completion net-tools \
wget git unzip tcpdump lsof host nslookup tmux \
bind-utils python3 epel-release htop fuse-sshfs \
autossh groupinstall "Development Tools" python3-devel \
postgresql-libs postgresql-devel python-pip \
python-devel
```

    d. Wygenerowanie i wgranie kluczy publicznych
        i. na plutonie:

```
ssh-keygen
chmod g-r,o-r ~/.ssh/id_rsa.pub
cat ~/.ssh/id_rsa.pub
# wyswietlony klucz publiczny nalezy skopiowac do schowka
```

ii. na node07:

```
su - [username]
mkdir .ssh
chmod 700 .ssh
cd .ssh
echo "[klucz publiczny]" > authorized_keys
chmod 600 authorized_keys
```

Teraz po wpisaniu na plutonie polecenia:
ssh [user]@node07.iot.kt.agh.edu.pl
przeniesie nas automatycznie (uwierzytelniając za pomocą klucza prywatnego) na VM node07

2. Połączenie Raspberry Pi z VM node07

a. Ustawiamy hostname na edge07 (w dalszej części dokumentacji używam na przemian “raspberry” oraz “edge07”)

```
hostnamectl set-hostname edge07
```

b. Instalacja niezbędnych narzędzi:

Należy powtórzyć krok “a” z poprzedniego punktu.
Uwaga: na debianie odpowiednikiem “yum” jest “apt-get”

c. Stworzenie tunelów ssh, umożliwiających zalogowanie się na raspberry od strony node07:

i. Na raspberry:

1. Z poziomu roota uruchamiamy ‘crontab -e’ za pomocą wybranego edytora (np.: vim) i dopisujemy na końcu:

```
@reboot sleep 30s; autossh -M 60007 \
-o "UserKnownHostsFile=/dev/null" \
-o "StrictHostKeyChecking=no" -f -N -R \
localhost:54321:localhost:22 \
pczaja@pluton.kt.agh.edu.pl
```

Umożliwia to połączenie się z plutonem po uruchomieniu, a następnie utworzy na nim gniazdo sieciowe na porcie 54321, które będzie wskazywało na jego port lokany 22. Zatem wywołanie ‘ssh -p 54321 root@localhost’ z poziomu plutona połączy nas po ssh z raspberry. Dodatkowo wykorzystanie modułu autossh sprawi, że raspberry będzie automatycznie otwierało tunel w przypadku zamknięcia socketa na plutonie

ii. Na node07:

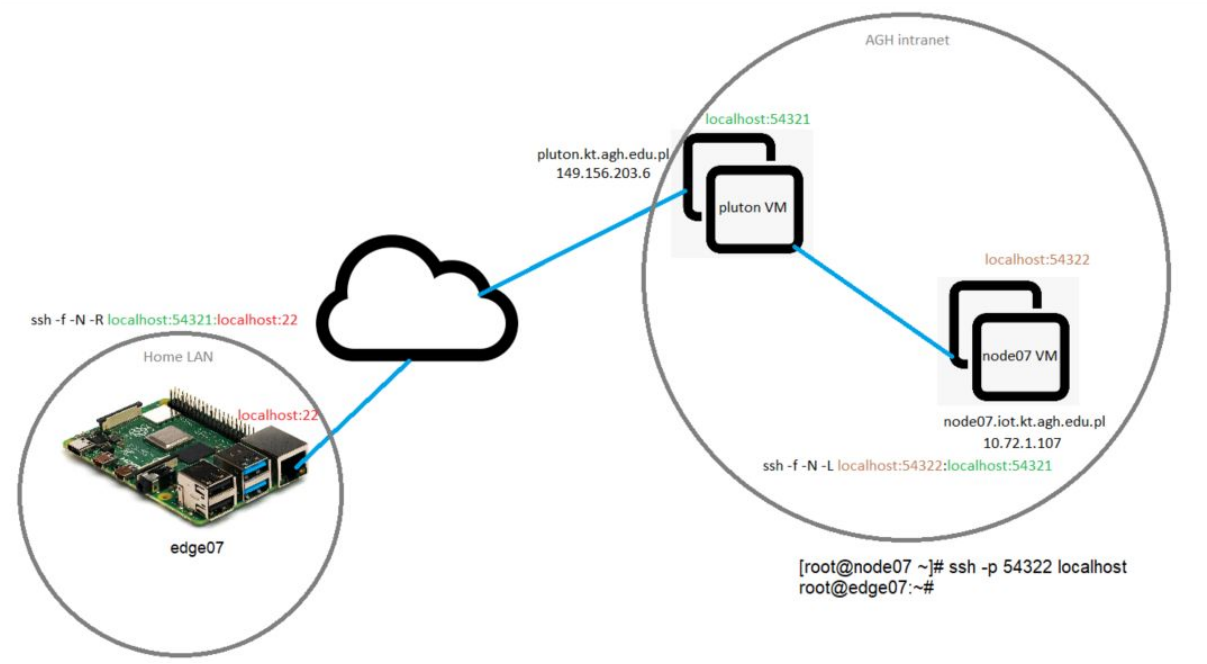
1. Analogicznie jak na raspberry modyfikujemy crontab, dodając na końcu:

```
autossh -M 60009 -o "UserKnownHostsFile=/dev/null"
\ -o "StrictHostKeyChecking=no" -f -N -L \
localhost:54322:localhost:54321 \
pczaja@pluton.kt.agh.edu.pl
```

Dzięki temu na node07 został stworzony socket, wskazujący na gniazdo localhost:54321 na plutonie. Tym samym uzyskaliśmy przekierowanie na port 22 raspberry, za pośrednictwem plutona.

Wykorzystane parametry autossh:
-f ⟶ działanie w tle
-N ⟶ odcięcie dostępu do shella zdalnej maszyny
-L ⟶ local forward
-R ⟶ remote forward

Schemat połączenia można zobaczyć poniżej:



e. Aby tunele opisane w poprzednim kroku zawiązywały się bez konieczności wpisywania hasła, należy umożliwić logowanie z wykorzystaniem klucza prywatnego pomiędzy:
   i. edge07 (root) → pluton (pczaja)
   ii. node07 (root) → pluton (pczaja)
   iii. node07 (root) → edge07 (root)

   Aby tego dokonać należy na urządzeniu, z którego chcemy się zalogować, wygenerować parę kluczy: publiczny i prywatny, za pomocą polecenia 'ssh-keygen'. Następnie należy wysłać klucz publiczny do urządzenia docelowego z użyciem polecenia 'ssh-copy-id'.

   Alternatywnie klucz publiczny można umieścić na urządzeniu docelowym ręcznie, wklejając go w odpowiedniej formie do pliku ~/.ssh/authorized_keys

f. Operacje, które można przeprowadzać na stworzonych tunelach
   i. połączenie po ssh node07 → raspberry:
      ssh -p 54322 raspberry@localhost
   ii. Aby zamknąć tunele należy wykonać (po stronie serwera oraz na raspberry):
      kill -9 $(lsof -t -c ssh)
   iii. zamontowanie konkretnego katalogu z rasppbery na node07:
      sshfs -p 54322 raspberry@localhost:\
      /path/to/some/raspberry/directory /mnt
   iv. Aby odmontować katalog, należy po stronie node07:
      umount /mnt

g. Uwagi dotyczące omawianych tuneli
   i. Tunele powstały wyłącznie w celach umożliwienia zdalnej konfiguracji raspberry. Gdyby raspberry ostatecznie trafiło na produkcję, zostałyby usunięte
   ii. W dalszych punktach dokumentacji opisuję jak w analogiczny sposób zestawiam kolejne tunele, umożliwiające połączenie się raspberry z bazą danych node07
   iii. Zestawione tunele działają bez konieczności ingerencji w firewall urządzeń końcowych (node07, edge07, pluton)
   iv. Aby zestawić połączenie nie są wymagane prawa roota na plutonie
   v. W przypadku utraty połączenia między node07 a edge07, spowodowanym zamknięciem gniazda localhost:54321 na plutonie, konieczne może być zresetowanie raspberry. Dzięki temu tunel zawiąże się na nowo.
   vi. Od strony node07 również mogą wystąpić problemy z tunelami.

Można je rozwiązać restartując tunel za pomocą skryptu /root/open_tunnel_to_edge07.sh.

```
[root@node07 ~]# cat open_tunnel_to_edge07.sh
/root/close_tunnel_to_edge07.sh
autossh -M 0 -o "UserKnownHostsFile=/dev/null" -o \
"StrictHostKeyChecking=no" -f -N -L \
localhost:54322:localhost:54321 pczaja@pluton.kt.agh.edu.pl

[root@node07 ~]# cat close_tunnel_to_edge07.sh
kill -9 $(lsof -i :ssh | grep pluton.kt.agh.edu.pl:ssh  \
|awk '{print  $2}') 2> /dev/null
kill -9 $(ps -ef | grep localhost:54322:localhost:54321 | awk \
'{print $2}') 2> /dev/null
```

3. Konfiguracja Access Point'a na raspberry w celu połączenia z ESP

(poniższą konfigurację należy przeprowadzić w całości ne edge07):

a. Konfiguracja adresu statycznego na interfejsie wlan0 - dodanie wpisu do pliku /etc/dhcpcd.conf:

```
tail -3 /etc/dhcpcd.conf
        interface wlan0
        static ip_address=192.168.240.1/28
        nohook wpa_supplicant
```

b. Konfiguracja DHCP (dnsmasq) na interfejsie wlan0:

i. Dobór adresacji podsieci:

```
Address:   192.168.240.1
Netmask:   255.255.255.240 = 28
Wildcard:  0.0.0.15
Network:   192.168.240.0/28  (Class C)
Broadcast: 192.168.240.15
HostMin:   192.168.240.1
HostMax:   192.168.240.14
Hosts/Net: 14
```

ii. Konfiguracja DNS - dodatnie wpisu do pliku /etc/dnsmasq.conf

```
cat /etc/dnsmasq.conf
interface=wlan0
dhcp-range=192.168.240.2,192.168.240.14, \
255.255.255.240,24h
```

iii. Konfiguracja serwisu hostapd

```
cat /etc/hostapd/hostapd.conf
interface=wlan0
#bridge=br0
driver=nl80211
ssid=IoT-Doorime
hw_mode=g
channel=7
wmm_enabled=0
# WMM-PS Unscheduled Automatic Power Save Delivery [U-APSD]
macaddr_acl=0
# Optionally, WPA passphrase can be received from \
RADIUS authentication server
auth_algs=1
# bit 1 = Shared Key Authentication
ignore_broadcast_ssid=2
# default: disabled (0)
# 1 = send empty (length=0) SSID in beacon and ignore \
probe request for broadcast SSID
# 2 = clear SSID (ASCII 0), but keep the original \
length and ignore probe requests for broadcast SSID \
wpa=2
wpa_passphrase=IoT_2020$D00r1m3
wpa_key_mgmt=WPA-PSK
# WPA-PSK = WPA-Personal / WPA2-Personal
wpa_pairwise=TKIP
# Pairwise cipher for WPA (v1)
rsn_pairwise=CCMP
# Pairwise cipher for RSN/WPA2
```

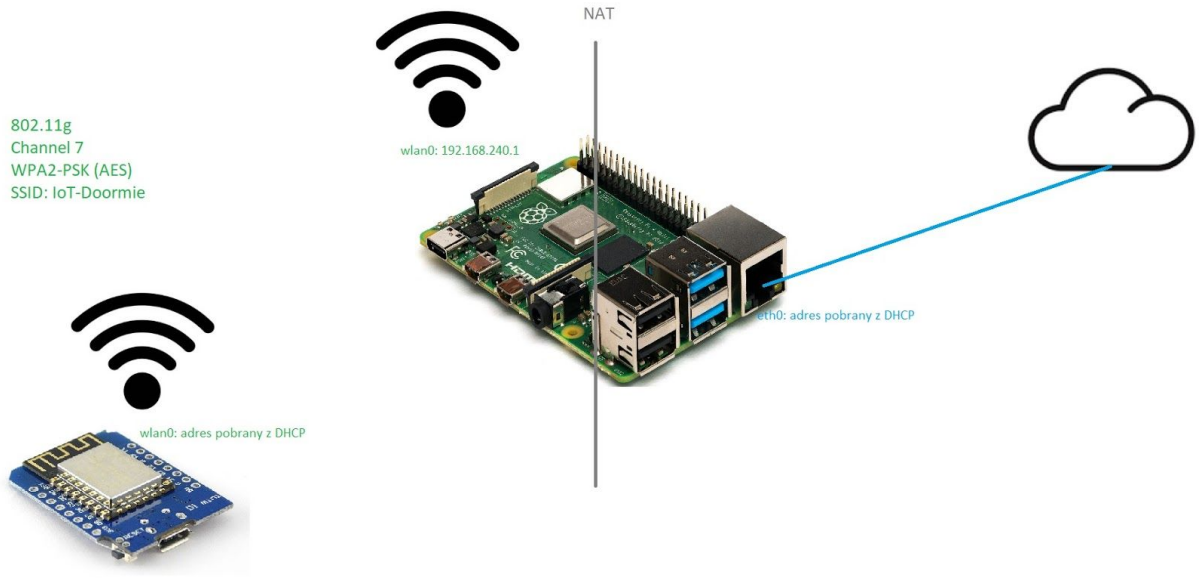
Podsumowanie powyższej konfiguracji:

- Wykorzystany standard wifi: 802.11g (2.4 GHz).
- Kanał 7
- Brak rozgłaszania ssid w beacon (pole ssid wyzerowane) + ignorowanie probe requestów
- Standard szyfrowania WPA2-PSK (AES)
- Urządzenie zostało skonfigurowane w taki sposób, że AP aktywowany jest zaraz po starcie

iv. Konfiguracja NAT z wykorzystanie *iptables:*

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

c. Każde urządzenie łączące się z raspberry za pomocą wifi będzie miało pełny dostęp do sieci. Warunkiem jest podłączenie raspberry do sieci z aktywnym DHCP za pomocą interfejsu eth0. Schemat obrazuje poniższa ilustracja:



4. Stworzenie tunelu ssh edge07→ node07, umożliwiającego raspberry połączenie bezpośrednio z bazą danych postgresql na node07:

a. Na maszynie wirtualnej node07 istnieje kontener zawierający bazę danych postgresql. Można się z nią połączyć lokalnie z poziomu VM, za pomocą gniazda localhost:5432. Aby możliwe było połączenie się z tym gniazdem bezpośrednio od strony raspberry, należy:

i. Na node07 uruchomić skrypt /root/share_db.sh.
Zawiera on odpowiednio przygotowane polecenie ‘autossh’:

```
[root@node07 ~]# cat share_db.sh
#!/bin/bash
autossh -f -M 20000 \
-o "UserKnownHostsFile=/dev/null" \
 -o "StrictHostKeyChecking=no" \
-f -N -R localhost:65432:localhost:5432 \
pczaja@pluton.kt.agh.edu.pl
```

Dzięki temu możliwe jest połączenie się z bazą z poziomu plutona (gniazdo: localhost:65432)

iii. Na edge07 należy dodać do crontaba (polecenie crontab -e) wpis:

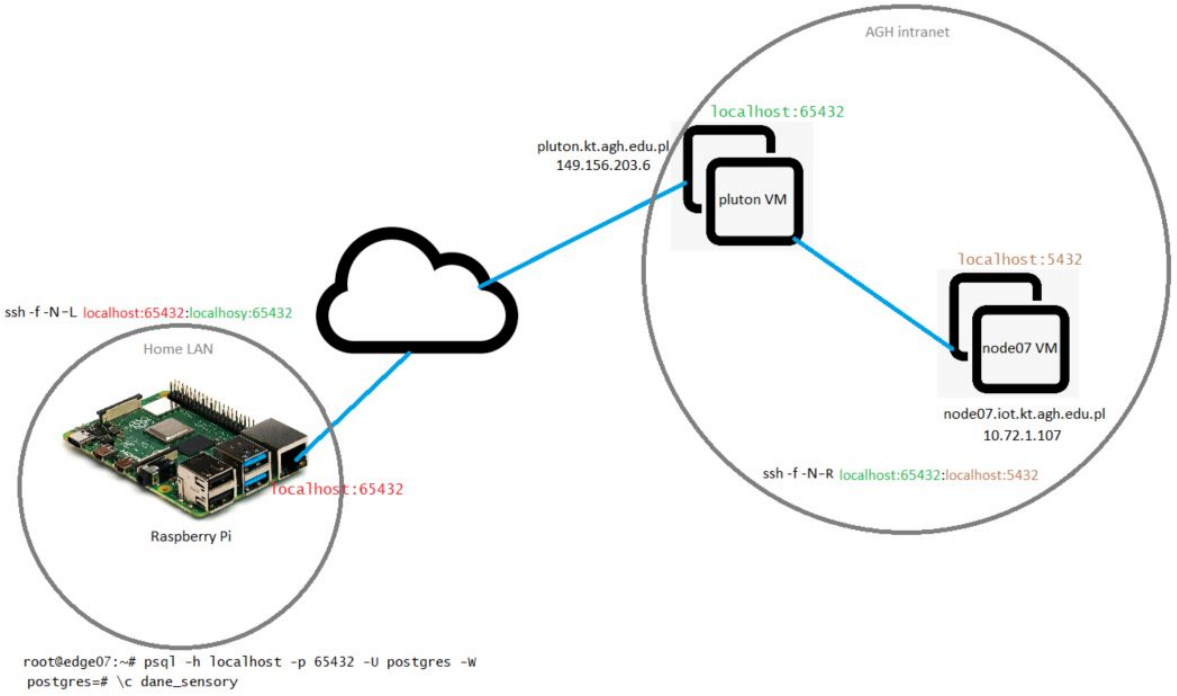
```
@reboot sleep 60s; exec autossh -f -M 62020 \
-o "UserKnownHostsFile=/dev/null" \
-o "StrictHostKeyChecking=no" -N -L \
localhost:65432:localhost:65432 pczaja@pluton.kt.agh.edu.pl
```

Teraz po uruchomieniu raspberry, jego port lokalny localhost:65432 wskazuje na port localhost:65432 na plutonie.
Natomiast localhost:65432 na plutonie prowadzi do localhost:5432 na node07 (do bazy danych pgsql).

b. Z poziomu raspberry można się połączyć z bazą danych, za pomocą poleceń:

```
[root@node07 ~]# docker exec -it postgres-database psql \
 -U postgres -W
postgres=# \c dane_sensory
dane_sensory=# \! pg_dump dane_sensory -U postgres -W
```

c. Schemat połączenia:



5. Stworzenie RESTfull API do komunikacji pomiędzy aplikacją klienta a bazą danych

a. Instalacja niezbędnych modułów python 3:

```
python3 -m pip install flask
python3 -m pip install flask_sqlalchemy
```

python3 -m pip install pyjwt

b. **UWAGA: Kod źródłowy oraz pełna dokumentacja API, znajduje się pod linkiem:** https://github.com/kolberszymon/smartHome/tree/master/api

c. Stworzenie bazy danych sqlite, przechowującej dane do uwierzytelniania:

```
yum install -m sqlite
cd /root/api/api_final
python
    from api import db
    db.create_all()
    exit()
```

d. Uruchomienie API:

```
python3 /root/api/api_final/api.py
```

6. Zestawienie tuneli między urządzeniem końcowym (np.: aplikacją) a maszyną wirtualną node07:

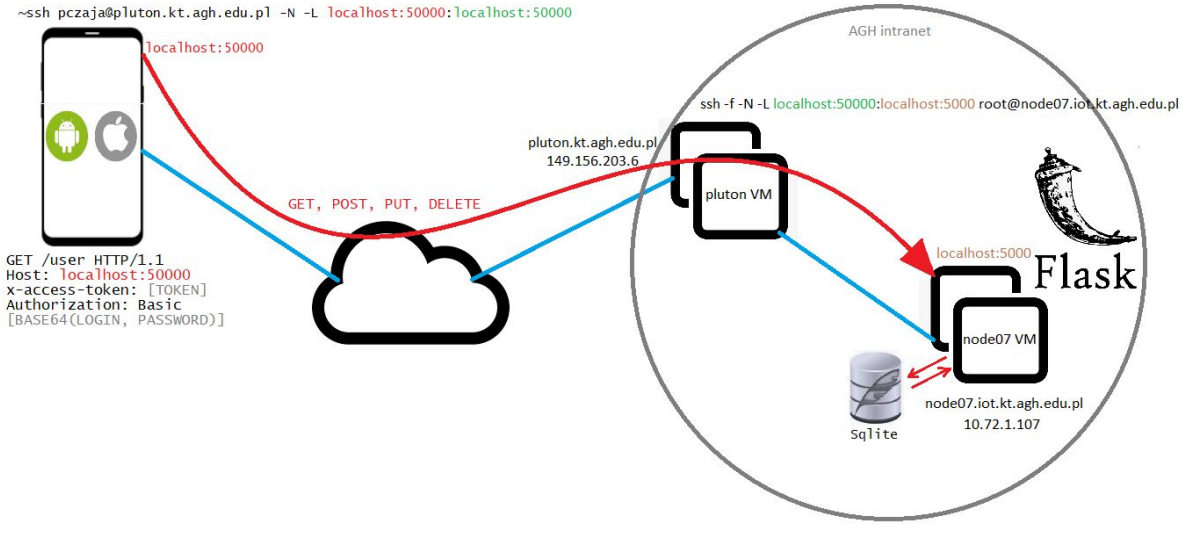
a. Na pluton.kt.agh.edu.pl:

```
ssh -o "UserKnownHostsFile=/dev/null" \
-o  "StrictHostKeyChecking=no" -f -N -L \
localhost:50000:localhost:5000 \ root@node07.iot.kt.agh.edu.pl
```

b. Na urządzeniu końcowym:

```
ssh pczaja@pluton.kt.agh.edu.pl -f -N -L \
localhost:50000:node07.iot.kt.agh.edu.pl:5000
```

Schemat połączenia:



7. Uwagi dotyczące API

c. Podczas tworzenia projektu używaliśmy debuggera Flask, który umożliwia podglądanie zapytać w czasie rzeczywistym. Aby go wyświetlić, należy na node07 wykonać polecenie: ‘tmux a -t api’.
d. W podobny sposób możliwe jest wyświetlenie stanu bazy danych sqlite, za pomocą: ‘tmux a -t sqlite’
e. W wersji produkcyjnej najprawdopodobniej obie opcję zostałyby wyłączone ze względów bezpieczeństwa
f. Baza danych sqlite zawiera jedynie informację o id użytkowników, ich nazwę użytkownika oraz hash ich hasła (sha256)
g. Poufność wysyłanych zapytań do api zapewniają tunele ssh, które szyfrują połączenie
h. Uwierzytelnianie do api odbywa się z wykorzystaniem tokena: po walidacji loginu oraz hasła użytkownik dostaje token, który upoważnia go do dalszych operacji
i. Token to zaszyfrowany symetrycznie json, zawierający id użytkownika oraz datę wygaśnięcia
j. Możliwe jest wykorzystywanie API niezależnie od aplikacji. Podczas testów wykorzystywałem aplikację ‘Postman’: https://www.postman.com/downloads/
k. Więcej informacji na temat api, można znaleźć w jego dokumentacji: https://github.com/kolberszymon/smartHome/blob/master/api/api_v3-dokumentacja.pdf

7. Dane Techniczne:
   - Protokół komunikacyjny: MQTT (prostota implementacji, skalowalność, mała zajętość łącza).
   - Zainstalowanie brokera Mosquitto implementującego protokół MQTT.

Dokumentacja Bazy Danych

Opis:

1. Baza Danych jest odporna na nieprawidłowe formaty danych.
2. Baza Danych triggeruje odpowiednie zachowania w zależności od przychodzących danych.
3. Baza Danych posiada odpowiednią strukturę danych.

Wymagania Techniczne:

1. Zainstalowane środowisko Docker.
2. Docker Engine.
3. PostgreSQL.

Implementacja:

Dzięki Docker'owi mamy możliwość odseparowania infrastruktury od aplikacji. Ponadto dostarcza nam możliwość na działanie aplikacji w wirtualnych kontenerach, które nie wymagają dodatkowego obciążenia nadzorcy oraz na działanie bezpośrednio w obrębie jądra gospodarza.

Baza danych zawiera trzy tabele:

```
dane_sensory=# \d+
                    List of relations
 Schema |    Name     | Type  |  Owner   | Size  | Description
--------+-------------+-------+----------+-------+-------------
 public | informacje  | table | postgres | 48 kB |
 public | uzytkownicy | table | postgres | 48 kB |
 public | wifi        | table | postgres | 16 kB |
(3 rows)
```

1. "Informacje" zawiera: datę, identyfikator urządzenia, nazwę stan oraz identyfikator kolekcji.

```
                                        Table "public.informacje"
   Column    |          Type          | Collation | Nullable |              Default               | Storage  | Stats target | Description
-------------+------------------------+-----------+----------+------------------------------------+----------+--------------+-------------
 data        | character varying(10000) |           |          |                                    | extended |              |
 id_urzadz   | character varying(10000) |           |          |                                    | extended |              |
 nazwa_urzadz | character varying(10000) |           |          | 'domyslna nazwa'::character varying | extended |              |
 stan        | boolean                |           |          |                                    | plain    |              |
 id_kolekcja | character varying(10000) |           |          |                                    | extended |              |
Indexes:
    "informacje_id_urzadz_key" UNIQUE CONSTRAINT, btree (id_urzadz)
Access method: heap
```

2. “Użytkownicy” zawiera: identyfikator klienta, identyfikator kolekcji, nazwę

```
dane_sensory=# \d+ uzytkownicy
                                                Table "public.uzytkownicy"
     Column     |          Type          | Collation | Nullable |              Default               | Storage  | Stats target | Description
----------------+------------------------+-----------+----------+------------------------------------+----------+--------------+-------------
 id_klient      | character varying(1000)  |           |          |                                    | extended |              |
 id_kolekcja    | character varying(1000)  |           |          |                                    | extended |              |
 nazwa_kolekcji | character varying(10000) |           |          | 'domyslna nazwa'::character varying | extended |              |
Indexes:
    "klient_kolekcja" UNIQUE CONSTRAINT, btree (id_klient, id_kolekcja)
Access method: heap
```

3. “Wifi” zawiera: identyfikator klienta, identyfikator sieci

```
dane_sensory=# \d+ wifi
                                        Table "public.wifi"
  Column   |          Type          | Collation | Nullable | Default | Storage  | Stats target | Description
-----------+------------------------+-----------+----------+---------+----------+--------------+-------------
 id_klient | character varying(1000) |           |          |         | extended |              |
 ssid      | character varying(1000) |           |          |         | extended |              |
Access method: heap
```

Co więcej, aby zapewnić integralność danych w bazie został dodany trigger wywoływany przy czynności usuwania kolekcji z tabeli “użytkownicy”, który ma na celu usunięcie wszystkich urządzeń z tej kolekcji w tabeli “informacje”.

```
--
-- Name: del_devices(); Type: FUNCTION; Schema: public; Owner: postgres
--

CREATE FUNCTION public.del_devices() RETURNS trigger
    LANGUAGE plpgsql
    AS $$
begin
delete from informacje where id_kolekcja=old.id_kolekcja;return null;
end;
$$;
```

Przydatne Komendy:

Instalacja Docker:

$yum install -y yum-utils \ device-mapper-persistent-data \ lvm2
$yum-config-manager \ --add-repo \https://download.docker.com/linux/centos/docker-ce.repo

Instalacja silnika docker oraz containerd (container runtime):

$yum install docker-ce docker-ce-cli containerd.io

Pobieranie obrazu z DockerHub'a:

$docker pull postgres

Uruchomienie:
Uruchomienie bazy danych postgres w kontenerze działającym w tle, powiązany z wybranym portem, który pozwala na łączność do aplikacji spoza kontenera

$docker run --name postgres-database -e POSTGRES_PASSWORD=postgres -d -p 5432:5432 -v /home/kgluza/docker/volumes/postgres:/var/lib/postgresql/data postgres

W celu zwiększenia bezpieczeństwa, można dodatkowo stworzyć wcześniej katalog w którym będzie przechowywany backup (/home/kgluza/docker/volumes/postgres)

Dokumentacja aplikacji mobilnej

Opis projektu

1. Aplikacja stanowi warstwę prezentacji kompletnego systemu, monitorującego okna i drzwi w domu.
2. Po skonfigurowaniu fizycznych urządzeń, użytkownik może przypisać konkretne czujniki do swojego konta w aplikacji, aby śledzić ich stan.
3. Użytkownik może w każdym momencie sprawdzić stan przypisanych do siebie czujników.
4. Możliwy jest podział czujników w oddzielne zbiory (kolekcje).
5. Użytkownik przypisuje do swojego konta identyfikatory domowych sieci Wi-Fi, dzięki czemu po wyjściu z domu (w momencie rozłączenia z jedną z tych sieci) otrzyma powiadomienie o wszystkich otwartych drzwiach i oknach.

Pojęcia

1. Urządzenie - fizyczny czujnik, którego stan będziemy monitorować.
2. Stan urządzenia - informacja, czy konkretne drzwi bądź okna są otwarte, czy zamknięte.
3. Kolekcja - zbiór kilku urządzeń.

Wymagania techniczne

1. Aplikacja dedykowana jest dla systemu Android w wersji 4.1 bądź wyższej.
2. Do działania aplikacji konieczne jest połączenie z internetem.
3. W obecnej chwili, ze względu na trwający proces implementacji, do uruchomienia aplikacji konieczne jest posiadanie zewnętrznej platformy Expo (dostępnej w Sklepie Play). Po zakończeniu procesu implementacji będzie możliwa instalacja aplikacji bezpośrednio w systemie.
4. Ze względu na warunki projektu (serwer w prywatnej domenie AGH), przed uruchomieniem aplikacji konieczne jest zestawienie tunelu SSH pomiędzy smartfonem, a domeną *pluton.kt.agh.edu.pl*.

Funkcjonalności aplikacji

Konta użytkowników

1. Każdy użytkownik tworzy własne konto, do którego będzie przypisywał urządzenia. Przy pierwszym uruchomieniu aplikacji wyświetli się ekran, na którym można utworzyć konto i zalogować się na nie.
2. Użytkownik identyfikowany jest poprzez unikalną nazwę użytkownika.
3. Logowanie do konta następuje poprzez podanie nazwy użytkownika i hasła.
4. Po zalogowaniu się, sesja użytkownika zostaje utrzymana - użytkownik nie musi podawać loginu i hasła przy każdym uruchomieniu aplikacji (dopóki sam się nie wyloguje).
5. Użytkownik może się wylogować poprzez wciśnięcie przycisku “LOGOUT” w ekranie głównym aplikacji (na samym dole).
6. Użytkownik może usunąć swoje konto poprzez naciśnięcie przycisku “DELETE ACCOUNT” w ekranie głównym aplikacji. Konto może zostać usunięte tylko wtedy, gdy użytkownik nie posiada żadnych kolekcji (należy je wcześniej usunąć).

Kolekcje i urządzenia

1. Przed dodaniem urządzenia, konieczne jest najpierw utworzenie kolekcji, do której zostanie ono przypisane.
2. Aby utworzyć kolekcję, należy nacisnąć przycisk “ADD COLLECTION” w ekranie głównym aplikacji. Zostanie otwarty formularz, przy pomocy którego można utworzyć kolekcję o wybranej przez siebie nazwie.
3. Lista kolekcji użytkownika wyświetlana jest w ekranie głównym aplikacji.
4. Po kliknięciu w konkretną kolekcję aplikacja przechodzi do ekranu z listą urządzeń przypisanych do niej. Przy każdym urządzeniu zawarte są informacje o jego aktualnym stanie (zielona kłódka - zamknięty, czerwona kłódka - otwarty) oraz data ostatniej zmiany stanu.
5. Aby dodać nowe urządzenie do kolekcji należy nacisnąć przycisk “ADD DEVICE”. Zostanie otwarty formularz, w którym należy podać wybraną przez siebie nazwę urządzenia oraz identyfikator fizycznego czujnika (adres MAC).
6. Istnieje możliwość zmiany nazw bądź usuwania kolekcji i urządzeń. Po dłuższym naciśnięciu kolekcji/urządzenia otworzy się menu z opcjami “RENAME” oraz “DELETE”. Kolekcję można usunąć tylko wtedy, gdy nie posiada ona urządzeń.

Udostępnianie kolekcji

1. Użytkownik może udostępnić swoją kolekcję innemu użytkownikowi.
2. Po udostępnieniu kolekcji innemu użytkownikowi ma on prawo do edycji tej kolekcji (zmiany jej nazwy, dodawania i usuwania urządzeń).

3. Jeśli kolekcja jest przypisana do kilku użytkowników, usunięcie jej przez jednego z nich nie spowoduje jej definitywnego skasowania, ale jedynie usunięcia przypisania do niej tego użytkownika. Całkowite usunięcie kolekcji ma miejsce, gdy usuwa ją ostatni przypisany do niej użytkownik.
4. Aby udostępnić kolekcję innemu użytkownikowi należy nacisnąć przycisk “SHARE COLLECTION” wewnątrz tej kolekcji. Otworzy się formularz, w którym należy wpisać nazwę użytkownika, któremu chcemy ją udostępnić.

Wykrycie momentu opuszczenia domu

1. Każdy użytkownik może przypisać do swojego konta listę domowych sieci Wi-Fi.
2. Listę aktualnie przypisanych użytkownikowi sieci można otworzyć po wciśnięciu przycisku “MY NETWORKS”. W tym miejscu znajduje się też formularz, za pomocą którego użytkownik może przypisać sobie nową sieć.
3. Użytkownik może usunąć przypisaną do siebie sieć przez jej dłuższe naciśnięcie.
4. Jeśli nastąpi rozłączenie z jedną z przypisanych do użytkownika sieci i połączenie się z inną siecią (nie przypisaną do użytkownika), to aplikacja uznaje, że opuścił on dom. Jeśli w tym momencie jakieś urządzenia użytkownika są w stanie otwartym, to otrzyma on o tym powiadomienie.
5. Funkcjonalność powiadomień działa, gdy aplikacja jest uruchomiona w tle.

Informacje techniczne

Informacje ogólne

1. Aplikacja została zaimplementowana przy użyciu języka JavaScript, z wykorzystaniem frameworka React Native.
2. Wersje systemu android poniżej 4.1 nie obsługują tej technologii, stąd ograniczenie dotyczące wersji systemu.
3. Aplikacja wykorzystuje zewnętrzną platformę Expo, ułatwiającą proces implementacji i pozwalającą na proste uruchomienie aplikacji na urządzeniu.

Komunikacja z serwerem

1. Aby była możliwa komunikacja z serwerem, należy zestawić tunel SSH pomiędzy smartfonem, a domeną *pluton.kt.agh.edu.pl*.
2. Tunel jest zestawiany za pomocą osobnego programu, napisanego w języku Python (React Native nie posiada bibliotek pozwalających na zestawienie tunelu bezpośrednio z poziomu aplikacji).

3. Po uruchomieniu tunelu SSH, na urządzeniu zostaje otwarty port 54321, z którego ruch zostaje przekierowany na port 55555 serwera.
4. Dane przesyłane między klientem, a serwerem są w formacie JSON.
5. Do wykonywania zapytań HTTP została wykorzystana biblioteka axios (https://github.com/axios/axios).

Zapytania HTTP

L.p.	Funkcjonalność w aplikacji	Ścieżka	Metoda HTTP
1.	Utworzenie użytkownika	*/create_user*	POST
2.	Usunięcie konta użytkownika	/delete_user	DELETE
3.	Logowanie do konta	*/login*	GET
4.	Weryfikacja poprawności tokenu	/validate	GET
5.	Pobieranie danych użytkownika	*/get_all_data*	GET
6.	Utworzenie nowej kolekcji	*/add_collection*	PUT
7.	Zmiana nazwy kolekcji	/rename_collection	PUT
8.	Usunięcie kolekcji	/delete_collection	DELETE
9.	Dodanie urządzenia	/add_device	PUT
10.	Zmiana nazwy urządzenia	/rename_device	PUT
11.	Usunięcie urządzenia	/delete_device	DELETE
12.	Dodanie sieci wifi	/add_wifi	PUT
13.	Usunięcie sieci wifi	/delete_wifi	DELETE
14.	Udostępnienie kolekcji	/share_collection	POST

Dane przechowywane w pamięci urządzenia

Niżej wymienione zmienne przechowywane są bezpiecznie w trwałej (persistent storage) pamięci telefonu. Zapis i odczyt danych realizowany jest przy pomocy biblioteki SecureStore, szyfrującej przechowywane dane.

1. Token - zwracany po poprawnym uwierzytelnieniu.
2. Dane logowania (zapisane po poprawnym zalogowaniu):

    a. Nazwa użytkownika (username).
    b. Zmienna isAuth (typu boolean), która na starcie aplikacji mówi, czy użytkownik był wcześniej zalogowany, czy nie.
    c. Zmienna basicAuth, będąca haszem loginu użytkownika oraz hasła, wykorzystywana do odświeżania sesji użytkownika.
3. Zmienna atHome (typu boolean), która przetrzymuje informację, czy użytkownik przebywa obecnie w domu. Na jej podstawie aplikacja determinuje, czy użytkownik opuścił dom.

Uwierzytelnianie

1. Podczas logowania, aplikacja przesyła do serwera login oraz hasło użytkownika (w postaci haszu base64). Serwer po zweryfikowaniu poprawności tych danych zwraca token (gdy dane są poprawne).
2. Po poprawnym zalogowaniu, w pamięci urządzenia zostają zapisane zmienne token, username, isAuth i basicAuth. Zmienne username oraz isAuth są także przekazane do globalnej pamięci aplikacji. Na podstawie zmiennej isAuth aplikacja przełącza użytkownika z ekranu logowania do aplikacji.
3. Zaraz po uruchomieniu aplikacji, zostaje sprawdzona aktualność tokenu przechowywanego w pamięci (zapytanie */validate*). Jeśli nie jest on przedawniony, to nie ma potrzeby ponownego logowania.
4. Gdy użytkownik nie wyloguje się ze swojego konta, jego sesja zostanie utrzymana. Została do tego wykorzystana biblioteka BackgroundFetch (https://docs.expo.io/versions/latest/sdk/background-fetch/), oparta na bibliotece Task Manager (https://docs.expo.io/versions/latest/sdk/task-manager/), która co około 15 minut wykonuje zapytanie uwierzytelniające (*/login*), przesyłając hasz basicAuth, dzięki czemu token jest regularnie odświeżany i nie traci swojej ważności.
5. Token jest pobierany z pamięci urządzenia przed każdym zapytaniem HTTP i dołączany do nagłówka. Jeśli nie jest on poprawny lub jest przedawniony, to serwer zwraca odpowiedź ze statusem 401 (Unauthorized).
6. Token pozwala na identyfikację użytkownika przez serwer (na jego podstawie serwer wie, który użytkownik wykonał zapytanie).
7. Wylogowanie z aplikacji polega na usunięciu tokenu i danych logowania z pamięci urządzenia oraz ustawieniu zmiennej isAuth na false (wtedy aplikacja przechodzi do ekranu logowania).

Dane o urządzeniach i kolekcjach użytkownika

1. Wszystkie dane o urządzeniach, kolekcjach i sieciach użytkownika są zwracane przez serwer w jednej wiadomości, w formacie JSON.

2. Pobieranie tych danych zaimplementowane jest w funkcji getUserData. Po wywołaniu tej metody następuje wykonanie zapytania */get_all_data*, a zwrócone informacje zostają zapisane w stanie aplikacji.
3. Metoda getUserData wywoływana jest w kilku przypadkach:
    a. Na starcie aplikacji - zaraz po zalogowaniu bądź po uruchomieniu aplikacji gdy jesteśmy już zalogowani.
    b. Po każdym udanym zapytaniu powodującym zmiany w bazie danych, np. dodanie kolekcji, zmiana nazwy.
    c. Po opuszczeniu domu przez użytkownika (przed wysłaniem powiadomienia).
    d. Przy ręcznym wykonaniu odświeżenia (przesunięcie w dół w ekranie głównym aplikacji).

Wykrycie momentu opuszczenia domu

1. Do pobierania informacji o stanie sieci została wykorzystana biblioteka NetInfo (https://github.com/react-native-community/react-native-netinfo).
2. Wykorzystana została funkcja addEventListener(), która ciągle nasłuchuje stanu sieci. W momencie jego zmiany, uruchamia ona tzw. funkcję zwrotną (callback). W przypadku naszej aplikacji jest to sprawdzenie trzech przypadków:
    a. Jesteśmy w domu (gdy typ sieci to wifi, a ssid należy do listy sieci przypisanych do użytkownika) - wtedy następuje ustawienie zmiennej atHome na wartość true.
    b. Jesteśmy w innej sieci niż domowa - zanim zmienna atHome zostanie ustawiona na wartość false, następuje sprawdzenie jej poprzedniej wartości. Jeśli jej poprzednia wartość wynosiła true, oznacza to, że opuściliśmy dom. Wtedy następuje pobranie aktualnych danych o urządzeniach użytkownika z serwera i wywołanie funkcji, która sprawdza urządzenia w stanie otwartym i wysyła o nich powiadomienia.
    c. Brak połączenia z żadną siecią - wtedy nie dzieje się nic, bo aplikacja i tak nie byłaby w stanie pobrać danych z serwera i wysłać powiadomienia. Jest to także pewien sposób zabezpieczenia aplikacji, np. przed chwilowym rozłączeniem z siecią domową, aby nie dostać później powiadomienia.
3. Powiadomienia są determinowane przez aplikację mobilną (nie przez część serwerową). Używany jest do tego zewnętrzny serwer Expo.