

Kraków, 18.06.2020

Dokumentacja na koniec projektu  
z przedmiotu Bazy Danych

## ***SmartHome Server***

Twórcy:

Adrian Beściak

Marcin Malcher

<https://github.com/AdrianBesciak/SmartHome>

# 1. Zarys ogólny projektu

Projekt zakładał stworzenie serwera typu SmartHome, gromadzącego dane z czujników rozsianych po domu oraz pozwalającego na automatyczne i ręczne sterowanie urządzeniami w domu (np. podlewanie roślin, symulowanie obecności mieszkańców podczas wyjazdu).

Do projektu zdecydowaliśmy się użyć języka programowania Python oraz bazy MongoDB, która dzięki swoim właściwościom pozwoliła nam uniezależnić się od ścisłego schematu relacyjnej bazy danych. Dzięki temu każde urządzenie w naszym systemie może posiadać inne właściwości bez karkołomnej modyfikacji schematu przy każdym nowym typie urządzenia.

System został uruchomiony na minikomputerze RaspberryPi, dzięki czemu małe, energooszczędne urządzenie z Linuxem jest w stanie zarządzać wszystkimi peryferiami i hostować serwer WWW w sieci lokalnej.

W ramach prezentacji możliwości systemu stworzyliśmy dodatkowo 2 urządzenia komunikujące się z serwerem przy użyciu interfejsu UART, które zostały dodane do bazy danych.

## 2. Zastosowane kolekcje:

- **devices** - gromadzi zarejestrowane urządzenia
- **logs** - zbiera logi z operacji wykonanych przez urządzenia
- **tasks** - przechowuje informacje o zadaniach, które mają wykonać się o zaplanowanej porze
- **users** - gromadzi dane o użytkownikach

## 3. Kod obsługujący bazę na najniższym poziomie

Za bezpośrednią obsługę bazy odpowiada nasza klasa MongoCollection, będąca naszym wrapperem dla klienta z modułu PyMongo. Dzięki takiemu rozwiązaniu w razie potrzeby moglibyśmy bardzo łatwo zmienić bazę danych lub bibliotekę do jej obsługi

```
class MongoCollection:
    def __init__(self, collection_name):
        self.__client = MongoClient()
        self.__db = self.__client.SmartHomeServer
        self.__collection = self.__db[collection_name]

    def send(self, message):
        self.__collection.insert_one(message)

    def get(self, field, value):
        return self.__collection.find_one({field: value})
```

```
def remove(self, field, value):
    return self.__collection.delete_one({field: value})
```

```
def getAll(self):
    return self.__collection.find({})
```

## 4. Szczegółowy opis użycia bazy

### a. Kolekcja urządzeń

Do głównego urządzenia, tj. Raspberry Pi, można podłączyć wiele urządzeń peryferyjnych, jak Arduino lub NodeMCU. Dane o każdym z nich przechowywane są w kolekcji 'devices' i ładowane przy uruchomieniu systemu.

```
devices_db = mongo.MongoCollection('devices')
```

```
def load_devices_from_db(db, dev_dict):
    devices_collection = db.getAll()
    for dev in devices_collection:
        if dev['dev_type'] == 'serial':
            try:
                serial_device = serialdevice.SerialDevice(dev['serial_port'])
                dev_dict[serial_device.get_name()] = serial_device
                print('Loaded device ', serial_device.get_name())
            except:
                print('Device ', dev['dev_name'], ' not found on port ', dev['serial_port'], ' - you can try add this device manually')
```

Przykładowy dokument z kolekcji "devices":

```
{
    "_id" : ObjectId("5ecfbb724a83ffa5b913760d"),
    "dev_name" : "dev2",
    "dev_type" : "serial",
    "serial_port" : "ttyUSB0",
    "services" : [
        "startAlarm",
        "soundSignal",
        "doubleSoundSignal",
        "stopAlarm",
        "LEDgreen",
        "LEDred",
        "LEDblue",
        "LEDyellow",
        "LEDblack",
        "LEDwhite"
    ],
    "registration_date" : ISODate("2020-05-28T15:24:02.801Z")
}
```

```
}
```

## b. Kolekcja logów

Wszystkie polecenia wydawane urządzeniom, wraz z wartościami zwróconymi, są przechowywane w kolekcji 'logs'. Dzięki temu możliwa jest późniejsza analiza danych, np. zmian temperatury w pokoju.

```
class Logger:
    def __init__(self):
        self.__logs = mongoCollection.MongoCollection("logs")

    def addlog(self, device, service, message):
        log = { "device": device.get_name(),
                "service": service,
                "date": datetime.datetime.now(),
                "message": message}
        self.__logs.send(log)
```

Przykładowy dokument z kolekcji "logs":

```
{
    "_id" : ObjectId("5ecfd62af56820cca443718f"),
    "device" : "dev2",
    "service" : "LEDblue",
    "date" : ISODate("2020-05-28T17:18:02.880Z"),
    "message" : "Showed blue color"
}
```

## c. Kolekcja zadań - harmonogram

System pozwala na planowanie zadań do wykonania w przyszłości cyklicznie lub o konkretnej porze. Zadania takie są przechowywane w kolekcji 'tasks' i wykonywane gdy nadejdzie ich czas.

Przykładowy kod dodający zadanie do bazy:

```
name = input("What should the job be called?")
dev = input("Which device does it use?")
com = input("What command should be executed?")
print("Should it be executed at a given time or periodically?")
mod = input("Available options: at, every")
print("What time unit are we using?")
unit = input("Available options: minute, hour, day, month, year")
num = input("And how many " + unit + "s?")
```

##walidacja wprowadzonych danych

```
job = {
    "name": name,
    "device": dev,
    "command": com,
    "modifier": mod,
    "number": num,
    "unit": unit
}
self.__task_base.send(job)
print("Task added!")
```

Przykładowy dokument z kolekcji "tasks":

```
{
    "_id" : ObjectId("5edf23d14f3e17c072dac041"),
    "name" : "Sound",
    "device" : "dev2",
    "command" : "doubleSoundSignal",
    "modifier" : "every",
    "number" : "3",
    "unit" : "minute"
}
```

#### d. Kolekcja użytkowników:

System obsługuje wielu użytkowników, z których każdy może posiadać uprawnienia dostępu do różnych urządzeń peryferyjnych. Dzięki wykorzystaniu bazy dokumentowej można bez trudu przypisać użytkownikowi dowolną ilość uprawnień. Są oni przechowywani w kolekcji 'users'.

Przykładowy kod obsługujący rejestrację użytkownika:

```
def register():
    form = RegistrationForm()
    if form.validate_on_submit():
        hashed_pwd = bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user = User(username=form.username.data, email=form.email.data,
                    password=hashed_pwd, privileges=[])
        user.send_to_db()
        flash(f'Account created for {form.username.data}!', 'success')
        return redirect(url_for('login'))
    elif form.email.data and form.username.data and form.password.data and
         form.confirm_password.data:
        flash(f'Account not created - user already exist', 'danger')
```

```
return render_template('register.html', title='Register', form=form)
```

Oraz funkcja umieszczająca użytkownika w kolekcji:

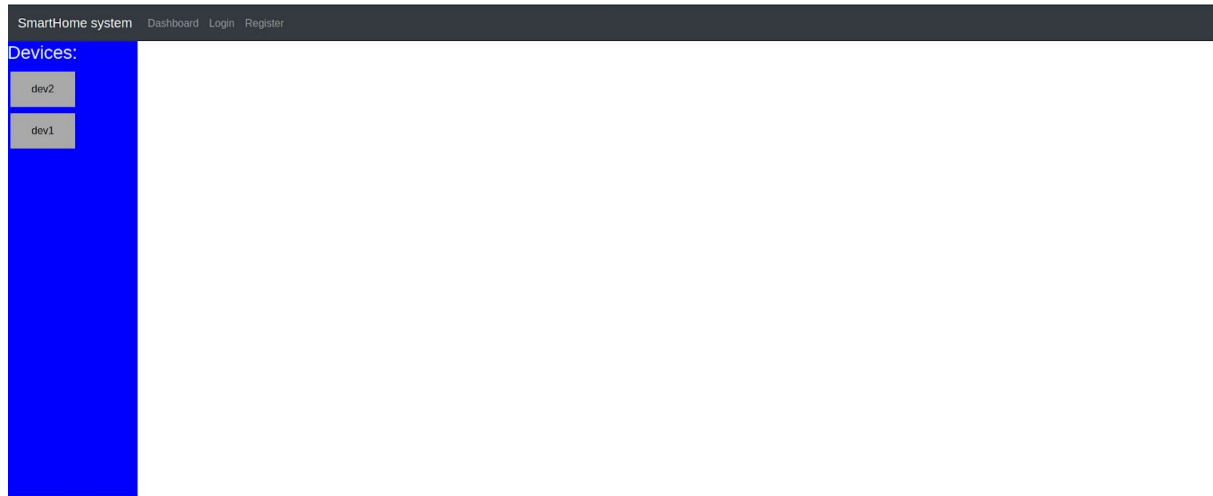
```
def send_to_db(self):
    self.db.send({
        "username": self.username,
        "email": self.email,
        "password": self.__password,
        "privileges": self.privileges
    })
```

Przykładowy dokument z kolekcji "users":

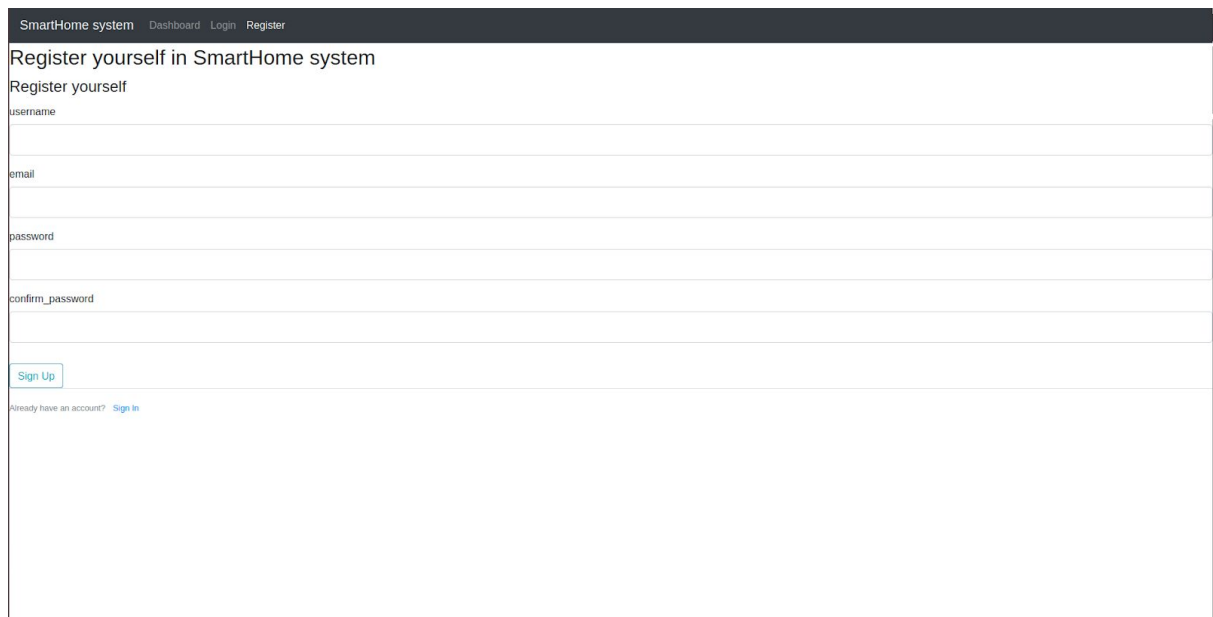
```
{
    "_id" : ObjectId("5edef7e027b797dc2099f2bd"),
    "username" : "asd",
    "email" : "asd@asd.com",
    "password": "$2b$12$P2u05nd5R.iojLNdusFbS.1eYZbteN1jkxVPNs7LQ
        guJSR8ofPjby",
    "privileges" : [
        "admin",
        "dev1",
        "Dev2"
    ]
}
```

Widoczne hasło zapisane w powyższym dokumencie jest haszem pochodzącym z modułu Bcrypt pythonowego frameworka Flask.

## 5. Przykładowe zrzuty ekranu z systemu oraz odpowiadające im reakcje peryferiów



### Widok niezalogowanego użytkownika

A screenshot of the user registration form in the SmartHome system. The top navigation bar is dark grey with the text 'SmartHome system' and links for 'Dashboard', 'Login', and 'Register'. The main content area is white and contains the following elements: the heading 'Register yourself in SmartHome system', the sub-heading 'Register yourself', and four input fields labeled 'username', 'email', 'password', and 'confirm\_password'. Below the input fields is a green 'Sign Up' button. At the bottom, there is a link that says 'Already have an account? Sign In'.

### Okno rejestracji użytkownika

SmartHome systemDashboardLoginRegister

Login

Log In

email

asd@asd.com

password

\*\*\*

☐ Remember me

Login

Forgot Password?

Don't have an account? Register

Okno logowania użytkownika

SmartHome systemDashboardAdmin's cockpitLogged as asdLogOut

Devices:

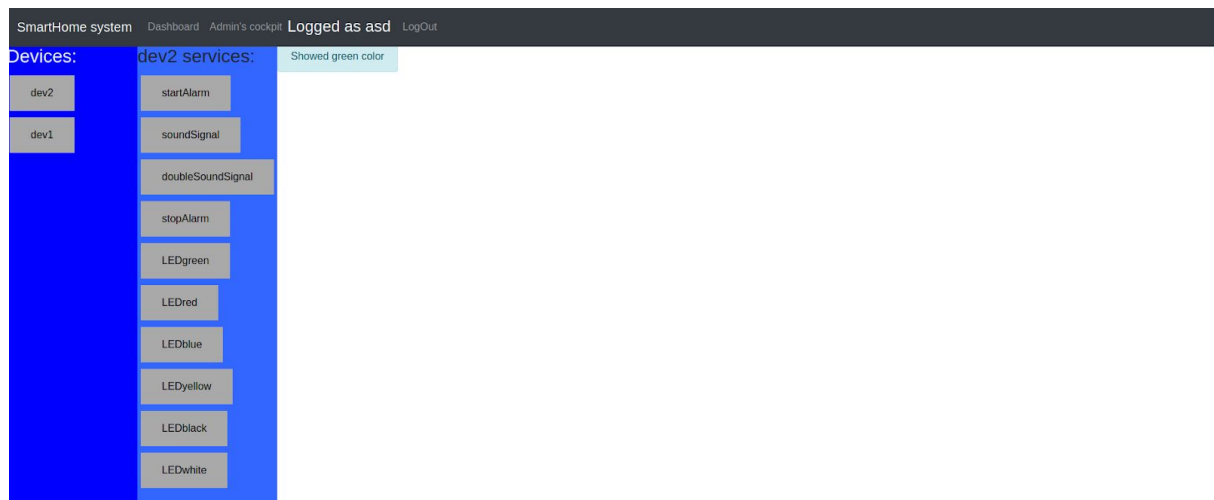
dev2

dev1

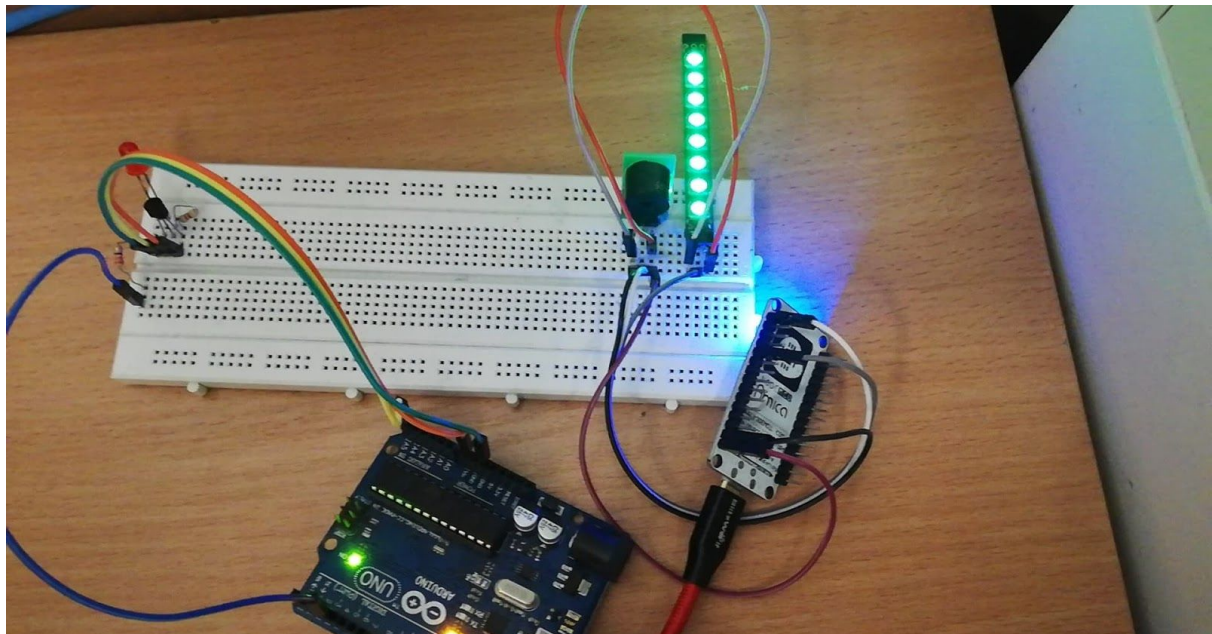
Logged in!

Panel zalogowanego użytkownika

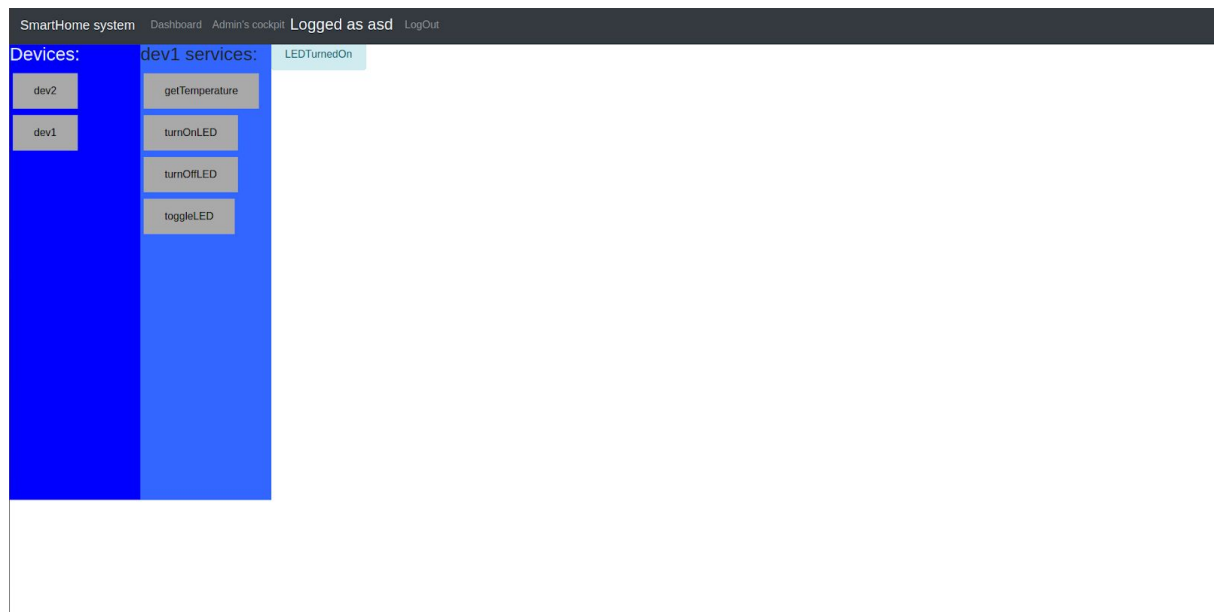




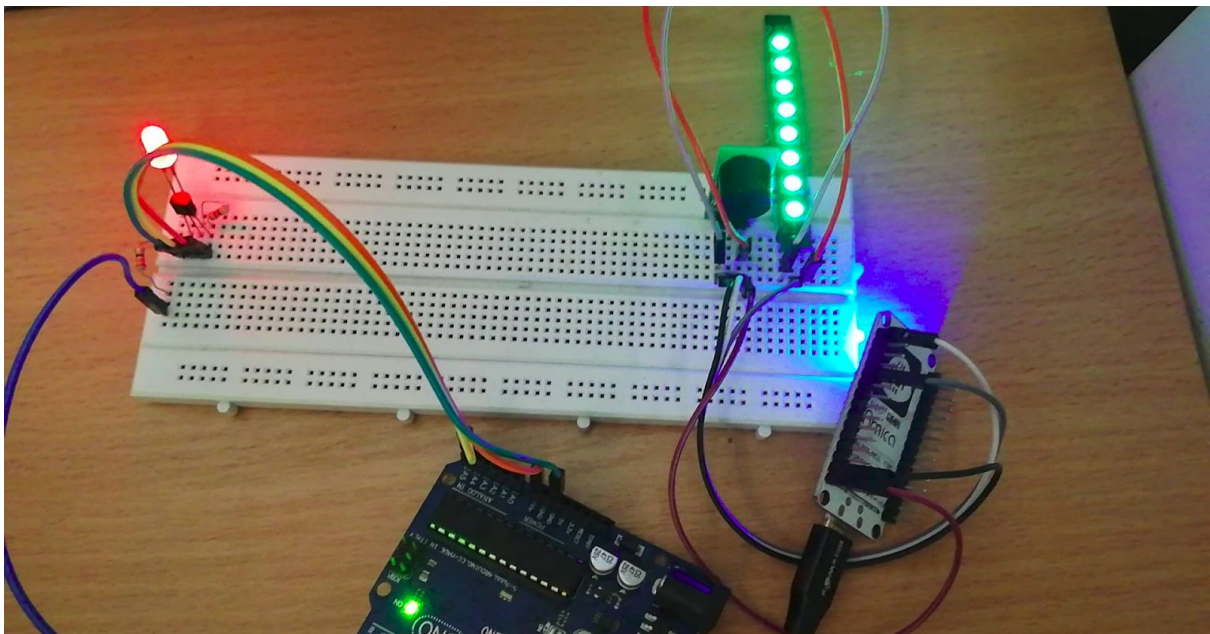
**Widok po uruchomieniu zielonego podświetlenia paska ledowego**

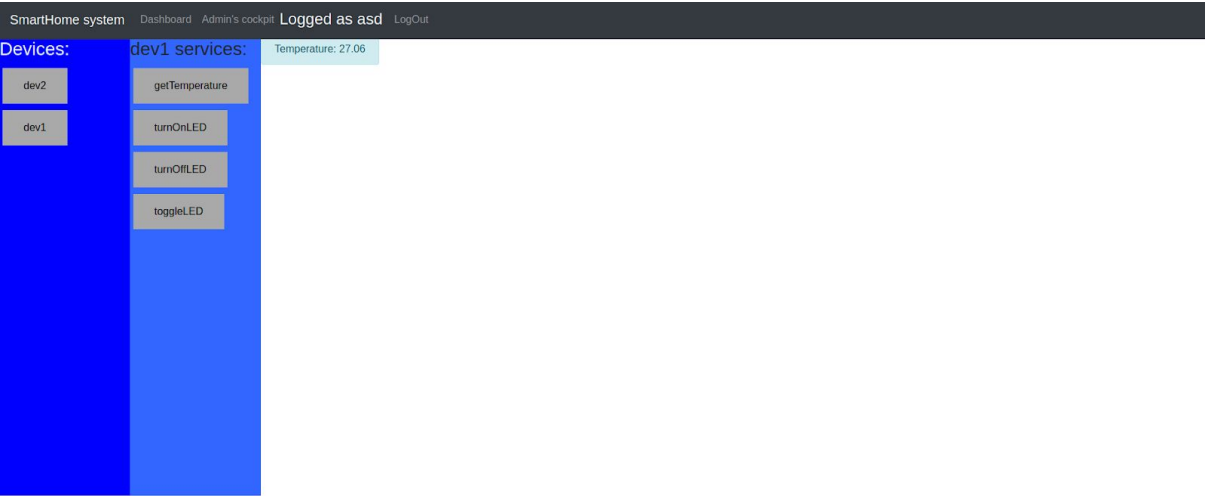


**Efekt wywołania serwisu - pasek świeci na zielono**

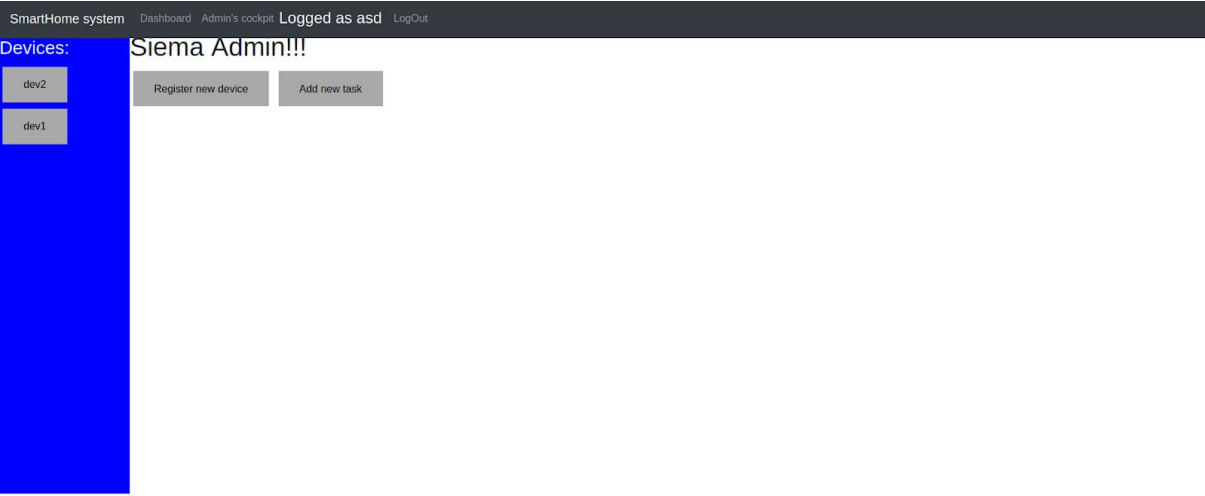


Widok po wywołaniu serwisu uruchomienia diody w urządzeniu “dev1”, a poniżej widać też załączoną tą diodę.





**Pomiar temperatury na termometrze obok czerwonej diody.**



**Panel administratorski**

SmartHome system Dashboard Admin's cockpit Logged as asd LogOut

### Register new device

Register new device

mode

port

Add device

## Panel rejestracji urządzenia dostępny z poziomu Administratora

SmartHome system Dashboard Admin's cockpit Logged as asd LogOut

### Add new task to Scheduler

Add new task

Name

Device

Service

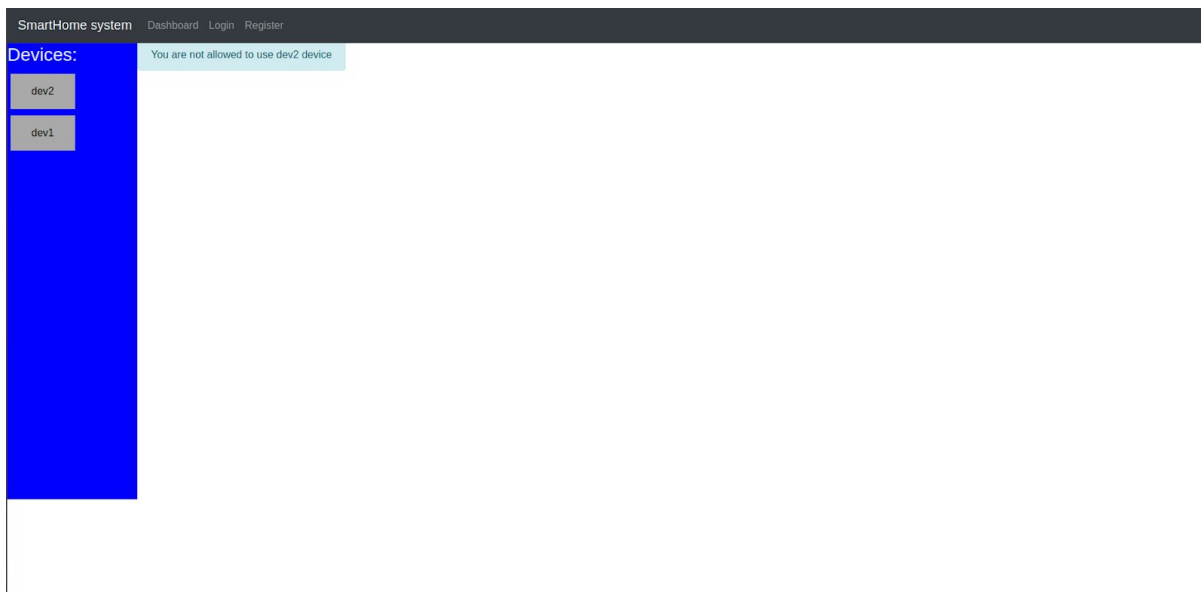
Modifier (at/every)

Unit (Available options: minute, hour, day, month, year)

Interval

Add task

## Panel dodawania zadania do schedulera.



**Nieudana próba wywołania serwisu przez użytkownika niezalogowanego lub bez odpowiednich uprawnień**