



Zachodniopomorski  
Uniwersytet Technologiczny  
w Szczecinie



Wydział  
Informatyki

## **Sebastian Dyjeta**

nr albumu: 39258

kierunek studiów: informatyka

specjalność: Systemu komputerowe i oprogramowanie

forma studiów: stacjonarne 1-go stopnia

**Synteza cyfrowego układu sterowania wybranymi parametrami klimatycznymi w celu uzyskania optymalnego mikroklimatu do hodowli roślin.**

**Synthesis of a digital control system of selected climatic parameters to obtain the optimal microclimate for plant growth.**

praca dyplomowa inżynierska

napisana pod kierunkiem:

**dr. inż. Sławomir JaszczaK**

Katedra

**Katedra Sztucznej Inteligencji i Matematyki Stosowanej**

Data wydania tematu pracy: XXX

Data złożenia pracy: XXX

Szczecin, 2019

# Spis treści

|  |           |
|--|-----------|
| <b>SPIS TREŚCI</b>   | <b>2</b>  |
| <b>1 WSTĘP</b>   | <b>3</b>  |
| <b>2 ZAKRES PRACY</b>  | <b>3</b>  |
| <b>3 TEORIA</b>  | <b>3</b>  |
| 3.1 PROGRAMY   | 3         |
| 3.1.1 <i>Visual Studio Code</i>                                    | 3         |
| 3.1.2 <i>FileZilla Client</i>                                      | 4         |
| 3.1.3 <i>Putty</i>   | 4         |
| 3.1.4 <i>Postman</i>   | 4         |
| 3.2 HTML   | 4         |
| 3.3 PYTHON   | 4         |
| 3.4 FLASK  | 5         |
| 3.5 SQLALCHEMY   | 5         |
| 3.6 SQLITE   | 5         |
| 3.7 GIT  | 6         |
| <b>4 IMPLEMENTACJA SYSTEMU</b>                                     | <b>6</b>  |
| 4.1 WYMAGANIA  | 6         |
| 4.1.1 <i>Wymagania funkcjonalne</i>                                | 6         |
| 4.1.2 <i>Wymagania нефункционалне</i>                              | 7         |
| 4.2 HARDWARE   | 7         |
| 4.2.1 <i>Spis komponentów</i>                                      | 7         |
| 4.2.2 <i>Opis komponentów</i>                                      | 7         |
| 4.2.3 <i>Schemat układu</i>  | 9         |
| 4.2.4 <i>SPI</i>   | 9         |
| 4.3 SOFTWARE   | 10        |
| 4.3.1 <i>Konfiguracja serwera</i>                                  | 10        |
| 4.3.2 <i>Utworzenie bazy danych</i>                                | 13        |
| 4.3.3 <i>API</i>   | 15        |
| 4.3.4 <i>Widoki</i>  | 16        |
| 4.3.5 <i>Algorytmy użycia aktorów</i>                              | 19        |
| 4.3.6 <i>Wykonywanie funkcji w zaplanowanym czasie</i>             | 20        |
| 4.3.7 <i>Komunikacja z systemem bez użycia strony internetowej</i> | 22        |
| 4.3.8 <i>Testy</i>   | 25        |
| <b>5 DWA TYGODNIE DZIAŁANIA SYSTEMU</b>                            | <b>26</b> |
| <b>6 PLANY NA ROZWÓJ</b>   | <b>26</b> |
| <b>7 PUBLIKACJE</b>  | <b>27</b> |
| 7.1 STRONY   | 27        |
| 7.2 KSIĄŻKI  | 27        |
| 7.3 ARTYKUŁY   | 27        |

## 1 Wstęp

Celem pracy jest stworzenie systemu, który automatyzuje proces dbania o roślinę oraz pozwala na dostosowanie parametrów przez stronę internetową lub Web API które pozwala zewnętrznej aplikacji na korzystanie z opisanego systemu. System nadzorujący ma za zadanie nawadniać roślinie oraz zapewnić jej naświetlenie w wyznaczonym czasie. Całość postawiona jest na platformie Raspberry Pi oraz miniframeworku Flask. System jest przeznaczony do roślin doniczkowych co ma swoje odwzorowanie w sile i wielkości lamp oraz pompy.

Niniejszy dokument stanowi formę dokumentacji stworzonego systemu, zawiera on opis stworzonych komponentów i wszelkie informacje potrzebne do zrozumienia czy odtworzenia projektu.

## 2 Zakres pracy

1. Opis użytych narzędzi.
2. Opis założeń i wymagań systemu.
3. Przygotowanie elementów sterujących.
4. Konfiguracja serwera.
5. Utworzenie bazy danych.
6. Implementacja aplikacji internetowej.
7. Implementacja API.
8. Implementacja algorytmów użycia urządzeń elektronicznych.
9. Przeprowadzenie testów API.
10. Przeprowadzenie testów systemu.

## 3 Streszczenie działań

- Teoria :
- Implementacja systemu :
- 

## 4 Teoria

### 4.1 Programy

#### 4.1.1 Visual Studio Code

Jest edytorem kodów źródłowych stworzonym przez firmę Microsoft. Bazowo zajmuje niewiele miejsca na dysku i jest prosty w użyciu, dzieje tak dlatego ponieważ edytor pozwala na dodanie rozszerzeń. Przez to przed przystąpieniem do projektu musimy poświęcić nieco uwagi rozszerzeniom których potrzebujemy ale wybór narzędzi jest przez to bardziej świadomy i mamy

pewność że żadna funkcjonalność narzędzia nie uchodzi naszej uwadze. Co ważne edytor działa na wszystkich popularnych systemach operacyjnych więc możemy przenieść swoje ustawienia niezależnie od miejsca pracy.

#### 4.1.2 FileZilla Client

Jest darmowym oprogramowaniem do połączeń z serwerami FTP/FTPS/SFTP które pozwala na wymianę plików. W opisywanym projekcie oprogramowanie to było wykorzystywane to przesyłania kodu źródłowego między stacją roboczą na której kod był pisany a serwerem na którym system był uruchamiany i testowany.

#### 4.1.3 Putty

Jest oprogramowaniem pozwalającym na połączenia przez protokoły takie jak SCP/SSH/Telnet. Korzystałem z niego do podłączenia się do serwera na którym mieścił się system.

#### 4.1.4 Postman

Oprogramowanie do testowania API, bardzo przydatne do testowania pojedynczych endpointów przy tworzeniu API ale też do tworzenia zautomatyzowanych testów API którego pomocą nam zapewnić poprawne działanie API nawet przy długotrwałym rozwoju oprogramowania przez kilka osób.

### 4.2 HTML

HTML to skrót od HyperText Markup Language jest prostym językiem do budowania struktury stron internetowych. Stronę taką buduje się przez odpowiednie wykorzystanie tzw. Znaczników. Każda ze stron internetowych powinna być podzieloną na dwie części zawierające metadane i drugą zawierającą strukturę strony.

### 4.3 Python

Python jest interpretowalnym, interaktywnym oraz obiektowo zorientowanym językiem programowania. Pozwala na korzystanie z wysoko poziomowych struktur danych takich jak listy, tablice słownikowe, moduły, klasy, wyjątki itd. Ma niesamowicie prostą i elegancką składnię lecz jest potężnym i uniwersalnym językiem programowania. Został zaprojektowany w 1990 roku przez Guido van Rossum. Jak wiele innych języków skryptowych jest darmowy nawet dla zastosowań komercyjnych i może być użyty na praktycznie każdym nowoczesnym komputerze. Kod Python`owy jest kompilowany automatycznie przez interpreter do niezelaznego od platformy kodu bajtowego który jest wykonywany. Możemy wykonywać niemodyfikowane komponenty napisane w Python`nie w linux`się, Window`się, SunOS oraz OSF. Python z założenia jest modularny, kernel jest bardzo mały I może być rozrzedzony przez dodanie modułów rozszerzeń. Dystrybucja Python`a zawiera zróżnicowaną bibliotekę standardowych rozszerzeń

(niektóre napisane w Python`ie inne w C lub C++) które pozwalają na rzeczy takie jak manipulacja string`ów, czy podobne do Perl`a wyrażenia regularne. [1]

Jako że Python jest językiem otwarto-źródłowym posiada dużą społeczność która dodaje nowe moduły, co często pozwala na budowanie dużych aplikacji w krótkim okresie czasu.

#### 4.4 Flask

Flask prostym frameworkiem który szybko pozwala nam na utworzenie API aplikacji napisane w Pythonie. Flask jest microframeworkiem co znaczy że nie potrzebuje żadnych zewnętrznych narzędzi ani bibliotek. Nie ma on walidacji formularzy weryfikacji użytkownika, ORMów itd. Daje nam tylko bazową funkcjonalność co pozwala nam na napisanie tych funkcjonalności samemu bądź na skorzystanie z wybranych przez siebie bibliotek. Dla zainteresowanych bardziej kompleksowym frameworkiem do API w pythonie zachęcam do poczytania o Django. Django posiada bazowo duża ilość funkcjonalności co sprawia że używa więcej zasobów i wymaga znacznie większej znajomości narzędzie ponieważ wiele rzeczy generuje automatycznie. Gdybym użył Django do opisywanego projektu, nie skorzystał bym z wielu funkcjonalności które oferuje, a Flask pozwolił mi na szybsze dobranie wymaganych komponentów co sprawia że system jest nie tylko bardziej przejrzysty, używa mniej zasobów oraz zajmuje mniej miejsca w pamięci.

#### 4.5 SQLAlchemy

SQLAlchemy jest ORMem (Object Relational Mapper) do Pythona, co znaczy że pozwala nam na przetworzenie obiektu z bazy danych na obiekt którym można posłużyć się w programie i na odwrót. Narzędzie to zapewnia nam zestaw wzorców zaprojektowanych do wydajnego dostępu do bazy danych, zaadaptowanych do języka jakim jest Python.

#### 4.6 SQLite

SQLite jest systemem do zarządzania bazami danych. Posiada on wbudowany silnik bazy danych i nie używa osobnego procesu serwera jak większość znanych baz danych. Niezależnie od złożoności bazy danych SQLite zapisuje wszystko w jednym pliku na dysku co nie tylko zapewnia łatwe przeniesienie systemu ale też ułatwia tworzenie kopii zapasowych.

Ważne cechy SQLite:

- Wieloplatformowość : można użyć go do aplikacji desktopowej, mobilnej lub w systemach osadzonych niezależnie od systemu operacyjnego.
- Nie wymaga serwera : SQLite w przeciwieństwie do większości systemów zarządzania bazami danych jest zintegrowany z aplikacją i nie wymaga osobnego serwera po którym łączymy się protokołem TCP/IP jak w przypadku MySQL, PostgreSQL itd.

- Transakcyjny : wszystkie operacje na bazie są w pełni zgodne z ACID więc w przypadku wielu zapytań lub ich przerwania nie musimy się martwić o dane w bazie.

#### 4.7 GIT

Narzędzie bez którego większość programistów nie wyobraża sobie pracy. GIT jest systemem kontroli wersji, co oznacza że śledzi wszelkie nasze zmiany w kodzie źródłowym od początku naszego projektu. Pozwala nam na cofnięcie się do poprzedniej wersji kodu lub zobaczenie ostatnich zmian dodanych przez współpracownika. Każde pobrane repozytorium jest samodzielne i w razie awarii głównego, można w pełni odtworzyć z je z repozytoriów lokalnych.

## 5 Implementacja systemu

### 5.1 Wymagania

#### 5.1.1 Wymagania funkcjonalne

- Aplikacja do kontroli urządzenia musi być dostępna przez przeglądarki internetowe.
- Aplikacja musi skalować się urządzenia mobile.
- Aplikacja posiada Web API pozwalające na komunikację z system bez użycia przeglądarki.
- Użytkownik może zapisać ustawienie rośliny.
- System musi zapewniać roślinie podane przez użytkownika parametry.
- System wykonuje polecane mu zadania w wyznaczonych przez użytkownika odstępach czasowych chyba że sensory nie wykazują takiej potrzeby.
- System potrafi nawodnić roślinę.
- System potrafi dać światło roślinie.
- System potrafi zmierzyć aktualną wilgoć gleby oraz siłę nasłonecznienia.
- System mierzy siłę nasłonecznienia w więcej niż 1 punkcie
- System wymaga maksymalnie 2 źródeł zasilania.
- Aplikacja jest umieszczona na Raspberry PI
- Użytkownik może włączyć lampy kiedy chce.
- Użytkownik może włączyć pompę kiedy chce.
- System rejestruje czas ostatniego użycia lamp.
- System rejestruje czas ostatniego użycia pompy.
- System pokazuje na stronie głównej aktualny stan aktorów.

### 5.1.2 Wymagania нефункционаłne

- System jest w stanie działać nieprzerwanie przynajmniej przez tydzień .
- Odpowiedź na każde żądanie odbywa się w mniej niż 200 ms.

## 5.2 Hardware

### 5.2.1 Spis komponentów

| Komponent  | cena   |
|--|--------|
| Raspberry Pi 3 B+                                | 200 zł |
| Moduł przekaźników 4 kanały<br>250VAC / cewka 5V | 25 zł  |
| Konwerter ADC MCP3008                            | 10 zł  |
| fotorezystor GL5528 x2                           | 4 zł   |
| Czujnik wilgoci gleby                            | 8 zł   |
| Taśma LED 300 GROW 1m                            | 50 zł  |
| Pompa 5V   | 20 zł  |
| razem  | 317 zł |

### 5.2.2 Opis komponentów

#### 5.2.2.1 Raspberry Pi 3 B+

Jest jednopłytkowym mikrokomputerem który posiada większość cech standardowego komputera takich jak złącza HDMI i USB, bezprzewodowe połączenie do sieci itd. System operacyjnym dedykowanym dla Raspberry jest Raspbian który jest bazowany na Debianie (dystrybucja linuxa). Dwie Głównie wyróżniające cechy Raspberry Pi to jedynie 5V wymaganego zasilania oraz GPIO (general purpose input/output) które pozwalają na kontrole oraz odczyt komponentów elektrycznych co sprawia że Raspiberry Pi jest dobrym narzędziem to wszelkich projektów elektronicznych które potrzebują dostępu do internetu, większej ilości obliczeń lub komunikacji z innymi urządzeniami. Dlatego też możemy znaleźć Raspiberry w wielu projektach związanych z IoT oraz smart home.

#### 5.2.2.2 Moduł przekaźników

Przekaźnik to w zasadzie przełącznik używający elektromagnesu. Elektromagnes potrzebuje niskiego napięcia by aktywować przełącznik co pozwala nam na włączenie/wyłączenie urządzenia które wymaga większego napięcia. Zapewniam nam to odseparowania źródeł zasilania co chroni nas przed przebicciem które może doprowadzić do uszkodzeń pozostałych komponentów które nie są przystosowane to takich wartości napięcia.

#### 5.2.2.3 ADC (konwerter analogowo cyfrowy)

Jako że Raspberry Pi nie jest w stanie sam odczytać wartości analogowych potrzebuje pomocy w postaci konwertera ADC. Konwerter analogowo cyfrowy to urządzenia które zmienia ciągły sygnał analogowy na przybliżony sygnał cyfrowy. Odbywa się to przez próbkowanie sygnału analogowego ze stałą częstotliwością. Dokładność konwersji zależy głównie od 2 parametrów, czyli czasu konwersji oraz zakresu sygnału cyfrowego. W przypadku użytego ADC MCP3008 jest to 10us czasu konwersji oraz konwersja do 10 bitowej wartości czyli w zakresie 0 – 1024.

#### 5.2.2.4 Fotorezystor

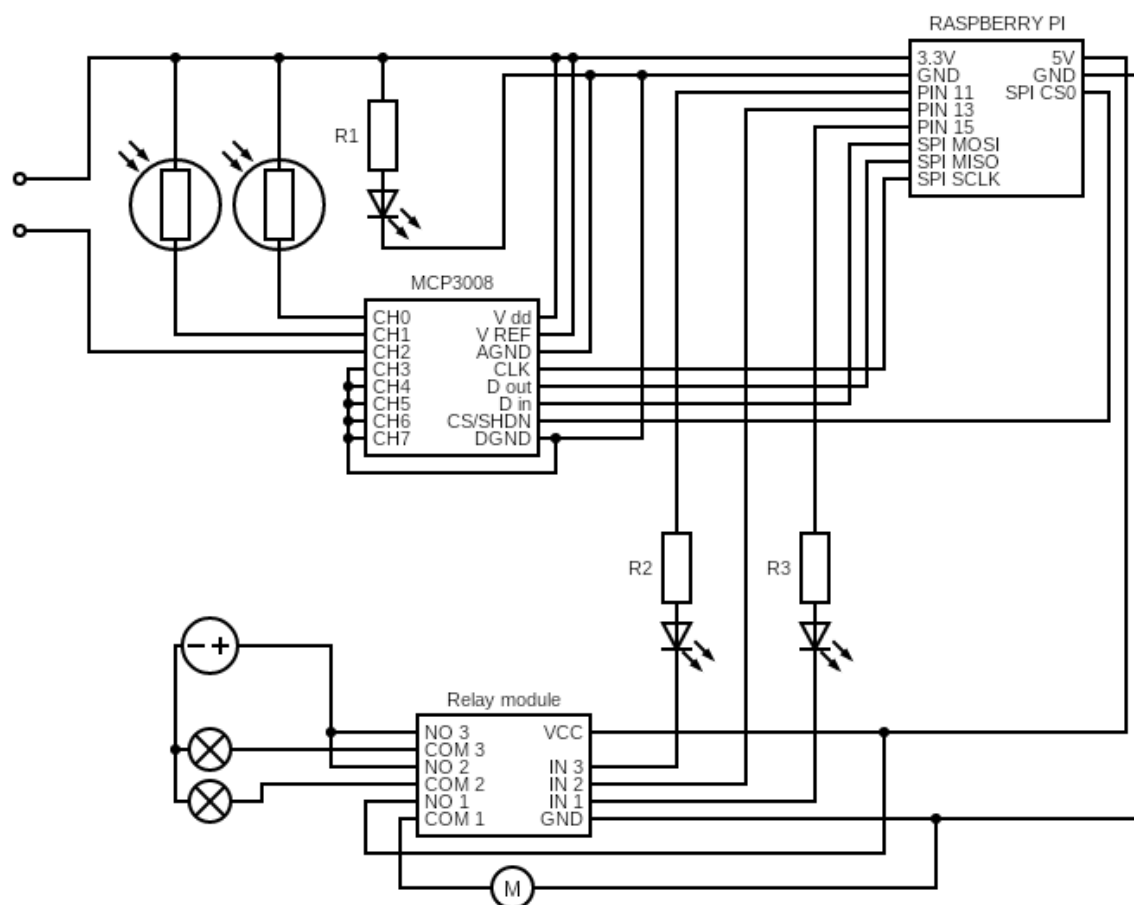
Czyli rezystor którego rezystancja jest zależna od siły nasłonecznienia. Pozwala nam on na pomiar wartości nasłonecznienia przez zliczenie różnicy napięć przed i za elementem, co pomoże nam określić czy roślina wymaga dodatkowego naświetlenia.

#### 5.2.2.5 Czujnik wilgoci gleby

Czujnik wilgotności gleby składa się z dwóch sond, które umożliwiają przepływ prądu przez wilgotną glebę. Można go bardzo łatwo wykorzystać, po prostu wkładając czujnik do gleby i odczytując wyniki za pomocą ADC, działa to ponieważ im większa wilgoć tym mniejsza rezystancja gleby.



### 5.2.3 Schemat układu



Rysunek 1 schemat opisywanego układu (wykonanie własne)

### 5.2.4 SPI

Bazowo Raspberry nie pozwala na odczyt danych analogowych, do tego potrzebujemy nie tylko konwertera ADC ale też połączenia przez konkretne piny protokołem SPI.

Szeregowy interfejs urządzeń peryferyjnych (Serial Peripheral Interface) jest protokołem komunikacji używanym do przesyłania danych pomiędzy mikro-komputerami i peryferiami. SPI używa czterech osobnych połączeń do komunikacji z docelowym urządzeniem, te połączenia to :

- Zegar (CLK – serial clock) – pin zegara zbiera impulsy o regularnej ustalonej częstotliwości. Dla ADC zegar wysyła impuls bazujący na wzroście krawędzi, czyli przejściu z niskiego do wysokiego napięcia.
- (MISO – Master Input Slave Output) – pin używany przez Raspberry PI do odbioru danych z urządzenia, dane są zbiera przy każdym impulsie zegara.

- (MOSI – Master Output Slave Input) – pin używany przez Raspberry Pi do wysyłania danych. Dane również wysłane dopiero przy impulsie zegara.
- (CS – Chip Select) – wybór która urządzenie SPI jest w użyciu. Ponieważ kilka urządzeń może dzielić CLK, MOSI i MISO lecz tylko aktualnie aktywne urządzenie z sygnałem niskim jest brane pod uwagę.

## 5.3 Software

### 5.3.1 Konfiguracja serwera

#### 5.3.1.1 Konfiguracja dostępu

Na początku pracy z Raspberry Pi musimy zadbać by udostępniało one odpowiednie porty do połączenia się z nim bez potrzeby podłączania peryferiów. Można co prawda napisać cały projekt bezpośrednio na mikrokomputerze ale jednak jego rozmiary świadczą też o jego wydajności co z poziomu działania systemu jest w pełni wystarczające, tak praca na nim może być nieco niekomfortowa gdy jesteśmy przyzwyczajeni do prędkości i funkcjonalności (np. kilka monitorów) które oferują standardowe stacje robocze.

Zacznijmy od ustawienia statycznego adresu IP. Można to zrobić na kilka sposobów, osobiście preferuje zrobić to w pliku konfiguracyjnym na Raspberry Pi ponieważ wtedy naszą konfigurację zabieramy wszędzie tam gdzie zabierzemy nasz system, co by nie miało miejsca gdybyśmy ustawili statyczny adres IP na routerze. Plik znajduje się pod ścieżką :

```
pi@raspberrypi:~ $ sudo nano /etc/dhcpd.conf
```

*Rysunek 2 fragment konsoli*

następnie schodzimy na dół pliku gdzie znajdziemy interface sieciowy wlan0, powinno to wyglądać tak :

```
interface wlan0
static ip_address=192.168.1.111/24
static routers=192.168.1.1
static domain_name_servers=8.8.8.8
```

*Rysunek 3 fragment konsoli*

w miejsce `ip_address` wpisujemy preferowany statyczny adres IP. Pamiętajcie o tym że adres musi znajdować się w tej samej podsieci co wasza stacja robota. Przypominam że by sprawdzić w jakiej podsieci jesteśmy powinniśmy w konsoli wpisać

```
pi@raspberrypi:~ $ ifconfig
```

*Rysunek 4 fragment konsoli*

i wyszukać interesujący nasz interface.

```

pi@raspberrypi:~ $ ifconfig
eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
    ether b8:27:eb:16:4e:ed txqueuelen 1000 (Ethernet)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 0 bytes 0 (0.0 B)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 0 bytes 0 (0.0 B)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.111 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::4373:cae3:7d4d:5aee prefixlen 64 scopeid 0x20<link>
    ether b8:27:eb:43:1b:b8 txqueuelen 1000 (Ethernet)
    RX packets 8798 bytes 12192945 (11.6 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 3747 bytes 446795 (436.3 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

Rysunek 5 fragment konsoli

Kolejną rzeczą którą musimy zrobić jest włączenie serwera SSH, Raspberry Pi ma taką funkcjonalność bazowo, ale jest ona wyłączona. By włączyć serwer SSH musimy wejść do programu konfiguracyjnego

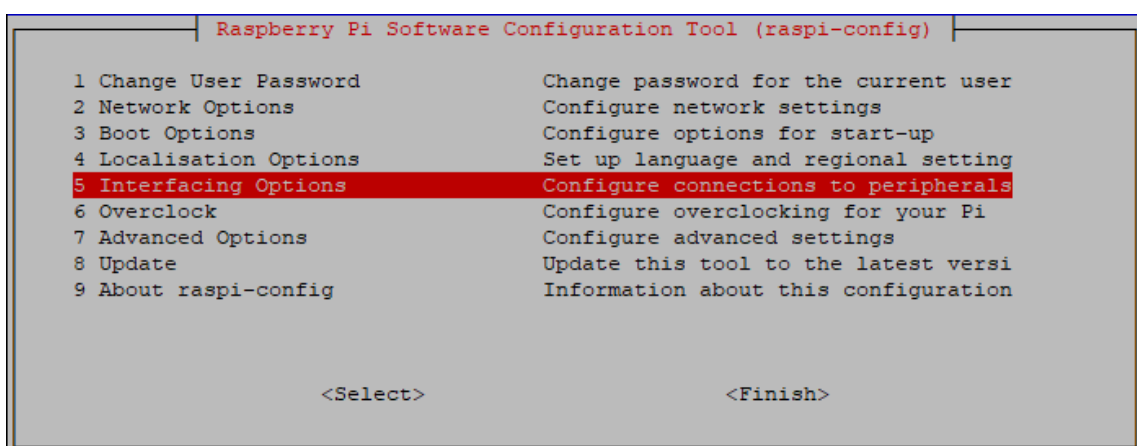
```

pi@raspberrypi:~ $ sudo raspi-config

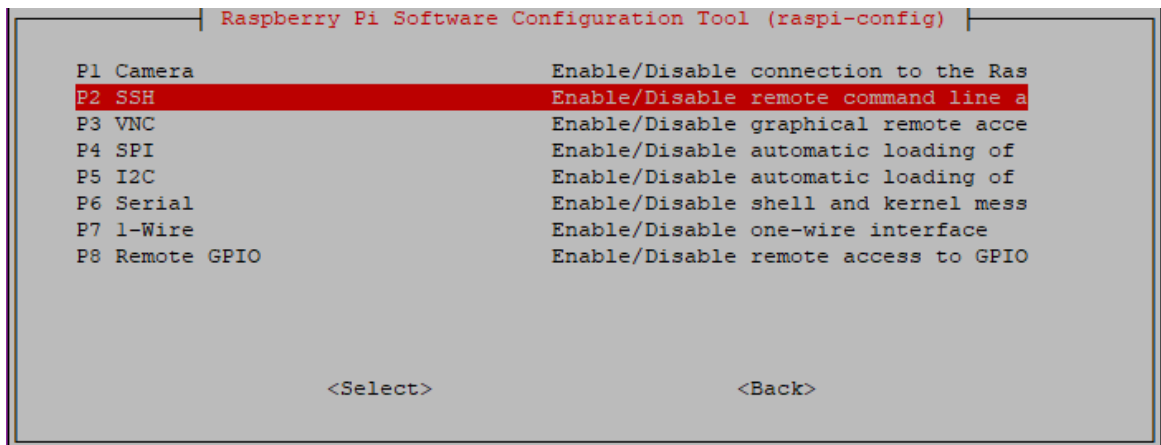
```

Rysunek 6 fragment konsoli

A następnie wybrać podkreślone na czerwono opcje :



Rysunek 7 fragment konsoli



Rysunek 8 fragment konsoli

Teraz zostało nam wpisać w konsoli :

```
pi@raspberrypi:~ $ sudo shutdown
```

Rysunek 9 fragment konsoli

by wyłączyć Raspberry Pi. Teraz należy odłączyć peryferia od Raspberry Pi i po ponownym włączeniu mamy już dostęp do urządzenia z poziomu swojego komputera.

#### 5.3.1.2 Włączenie aplikacji przy starcie systemu.

Dodajemy plik konfiguracyjny zawierający skrypt uruchamiający naszą aplikację do ścieżki :

```
pi@raspberrypi:~ $ cd /lib/systemd/system/
```

Rysunek 10 fragment konsoli

Następnie musimy użyć 2 komend które kolejno : załaduje nasz serwis oraz włączy go

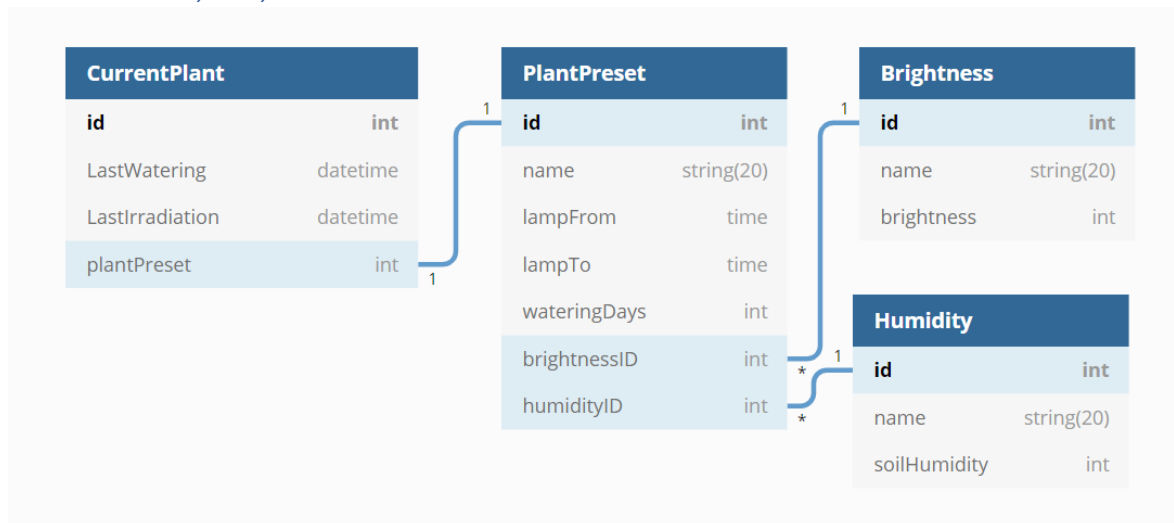
```
pi@raspberrypi:~ $ sudo systemctl daemon-reload
pi@raspberrypi:~ $ sudo systemctl enable sample.service
```

Rysunek 11 fragment konsoli

Od tej pory nasza aplikacja będzie włączać się przy każdym starcie systemu jako serwis.

## 5.3.2 Utworzenie bazy danych

### 5.3.2.1 Schemat bazy danych



Rysunek 12 schemat bazy danych (opracowanie własne)

CurrentPlant – tabela która przechowuje informacje o aktualnie wybranych ustawieniach oraz ostatnie daty podlania oraz naświetlenia rośliny.

PlantPreset – ustawienia systemu. Mogą być dodawane przez użytkownika.

Brightness – tablica zawierająca docelowe wartości naświetlenia w formacie zrozumiałym dla systemu oraz przyjazną dla użytkownika nazwę (np. „niewielkie naświetlenie”).

Humidity – tablica zawierająca docelowe wartości wilgotności gleby w formacie zrozumiałym dla systemu oraz przyjazną dla użytkownika nazwę (np. „wysoka wilgotność”).

### 5.3.2.2 Dodanie bazy danych

#### 5.3.2.2.1 Code first approach

W podejściu tym tworzymy najpierw modele(klasy) obiektów które chcemy mieć w bazie i dopiero później na ich podstawie konstruowana jest (często automatycznie) baza danych.

#### 5.3.2.2.2 Implementacja modeli

```
class PlantPreset(db.Model):
    __tablename__ = 'plantPreset'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(20), unique=True, nullable=False)
    lampFrom = db.Column(db.Time, nullable=True)
    lampTo = db.Column(db.Time, nullable=True)
    wateringDays = db.Column(db.Integer, nullable=True)
    brightnessID = db.Column(db.Integer, db.ForeignKey('brightness.id'), nullable=True)
    humidityID = db.Column(db.Integer, db.ForeignKey('humidity.id'), nullable=True)
```

Rysunek 13 fragment pliku „models.py”

Każdy model widziany przez SQLAlchemy musi dziedziczyć po klasie Model oraz posiadać kolumny opisane w formacie :

```
nazwaZmiennej = db.Column(db.typDanych, dodatkowe parametry)
```

Rysunek 14 przykład dodania kolumny w SQLAlchemy

Przez dodatkowe parametry mam na myśli auto inkrementację, indeksowanie itp. Zmienna `__tablename__` nie jest konieczna ale pozwala nam nadanie dowolnej nazwy dla tabeli niezależnie od nazwy klasy modelu, domyślną nazwą dla PlantPreset była by nazwa `plant_preset`.

Teraz musimy utworzyć bazę danych z napisanych modeli, używamy do tego prostej komendy :

```
Models.db.create_all()
```

Rysunek 15 fragment pliku „baseDBSetup.py”

#### 5.3.2.3 Połączenie z bazą danych

```
#!/usr/bin/env python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime, time

app = Flask(__name__)

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///site.db'

db = SQLAlchemy(app)
```

Rysunek 16 przykładowy kod łączący SQLite, Flask i SQLAlchemy

W tym przykładzie możemy zobaczyć dlaczego tak dużo osób preferuje język Python zwłaszcza przy tworzeniu małych aplikacji. Kilka linii kodu i już nasza baza danych jest podłączona. Kodu co prawda jest mało ale integrujemy tutaj aż 3 komponenty czyli :

- SQLite
- SQLAlchemy
- Flask

#### 5.3.2.4 CRUD (Create, Read, Update, Delete) w SQLAlchemy

Każda baza danych udostępnia podstawowe 4 funkcjonalności znane jako CRUD czyli :

- Dodanie do bazy (Create)

```
brightness = Models.Brightness(name = "medium" , brightness = 450)
Models.db.session.add(humidity2)
Models.db.session.commit()
```

Rysunek 17 fragment pliku „baseDBSetup.py”

- Odczyt z bazy (Read)

```
currentPreset = Models.CurrentPlant.query.first()
presetDetails = Models.PlantPreset.query.filter_by(id=currentPreset.plantPreset).first()
```

Rysunek 18 fragment pliku „main.py”

- Edytowanie danych w bazie (Update)

```
plant = Models.CurrentPlant.query.first()
plant.plantPreset = request.form['preset']
Models.db.session.commit()
```

Rysunek 19 fragment pliku „main.py”

- Usuwanie z bazy (Delete)

```
brightness = Models.Brightness.query.first()
Models.db.session.delete(brightness)
Models.db.session.commit()
```

Rysunek 20 przykładowy kod usuwający wpis w bazie danych

### 5.3.3 API

#### 5.3.3.1 Podstawy Web API

```
1  #!/usr/bin/env python
2  from flask import Flask
3
4  app = Flask(__name__)
5
6  @app.route('/')
7  def hello_world():
8      return 'Hello, World!'
9
10 if __name__ == '__main__':
11     app.run(host='0.0.0.0', port = 80 , debug = True)
```

Rysunek 21 przykładowy kod bazowej aplikacji wykorzystującej Flask

Na przykładzie powyżej widzimy najprostsze API w frameworku Flask, linii 6-7 dodaje nam bazowy endpoint (nasz adres bez żadnej podstrony) który zwróci nam „Hello, World!”. Linia 11 mówi o tym że

- Host='0.0.0.0' – strona jest dostępna na adresie aktualnej maszyny

- Port = 80 – strona jest dostępna na porcie 80 czyli bazowym porcie http
- Debug = True – mówi że serwer jest włączony w trybie Debug. Należy pamiętać by finalnie opcja ta była ustawiona na False, ponieważ tryb ten pozwala na poznanie całej struktury aplikacji oraz wywołanie dowolnych funkcji.

```
@Models.app.route('/changePlantSettingsHandler', methods=['GET', 'POST'])
```

*Rysunek 22 fragment pliku „main.py”*

W Flasku jak w przypadku każdego frameworka do tworzenia API o architekturze RESTowej tworzymy tak zwane endpointy do których wysyłane są żądania HTTP, endpoint składa się z adresu domeny, parametrów oraz metody. Jeden URL może mieć do siebie przypisane kilka endpointów w zależności od użytej metody, warto pamiętać że domyślnie przeglądarka internetowa wysyła rządzenie GET.

```
@Models.app.route('/lamp/<state>')
```

*Rysunek 23 fragment pliku „main.py”*

Flask pozwala nam też na obsługę zmiennych przesłanych przez użytkownika w URL w tym przypadku „lamp” jest czytane jako endpoint a następna rzecz po „/” będzie odczytana jako wartość zmiennej. W ten sposób przesyła się proste oraz niewrażliwe dane. Bardziej złożone dane lub te wrażliwe wysyłamy w ciele zapytania.

Po pozytywnym rozpatrzeniu żądania wysyłamy zazwyczaj kod statusu 200 (kod sukcesu) oraz odpowiedź która w architekturze REST jest zazwyczaj plikiem JSON, choć w przypadku opisywanego systemu w większości przypadków będzie to strona internetowa.

Napisałem zazwyczaj ponieważ są odstępstwa od tych reguł nawet jeśli żądanie zakończone zostaje pozytywnie. W Przypadku chęci usunięcia czegoś z bazy (metoda DELETE) standardową praktyką jest zwrócenie kodu 204 (brak zawartości), lub w przypadku przekierowania na inną stronę dostaniemy kod 302.

Przekierowanie do innego endpointa wygląda w następujący sposób :

```
return redirect(url_for('changePlantSettings'))
```

*Rysunek 24 fragment pliku „main.py”*

## 5.3.4 Widoki

### 5.3.4.1 Generowanie strony

Gdy endpoint kończy się w ten sposób



```
return render_template('index.html' , pumpStatus = pumpStatus , lastWatering=
```

Rysunek 25 fragment pliku „main.py”

Zwracamy użytkownikowi status 200 wraz z wygenerowaną stroną w ciele odpowiedzi. Przebieg generowania strony jest następujący :

Endpoint odnosi się do konkretnego pliku html (w tym przypadku „index.html”) który powinien zaczynać się od

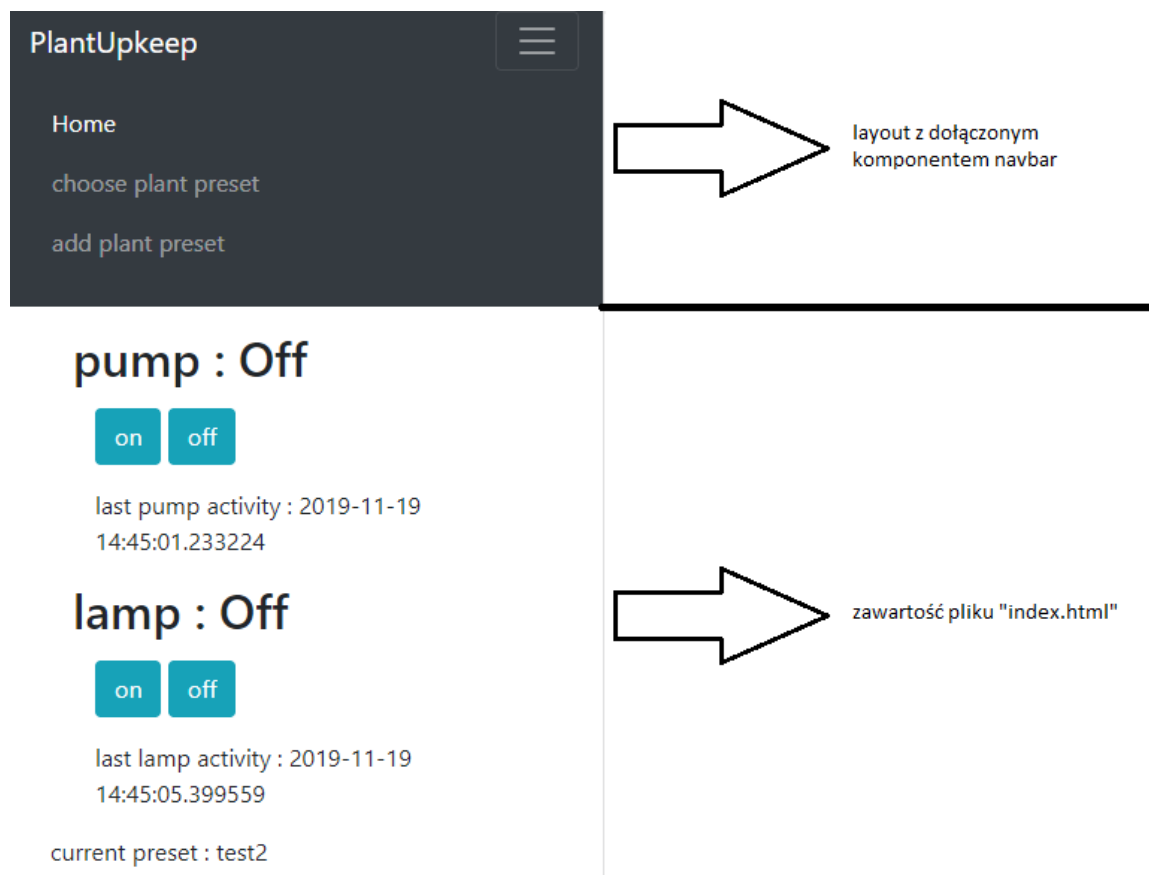
```
{% extends 'layout.html' %}
```

Który łączy do strony część wspólną dla wszystkich podstron, następnie piszemy standardowy kod HTML. Możemy też doładować w dowolnym miejscu komponent

```
{% include 'includes/navBar.html' %}
```

Co załaduje nam cały HTML z danego pliku. W tym przypadku layout zawierający metadane doładowuje navbar.html ponieważ są to dwie rzeczy które będą znajdować się na każdej stronie.

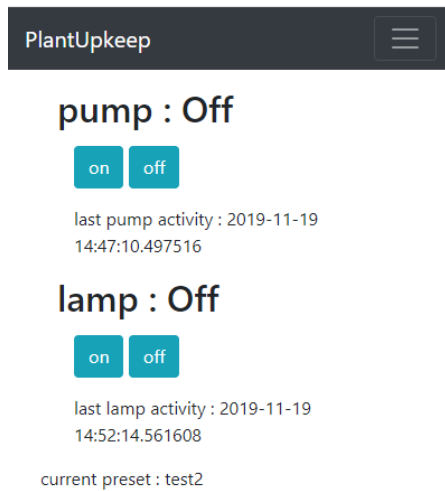
Wynik :



Rysunek 26 wygenerowany widok „index.html”

#### 5.3.4.2 Widoki w systemie

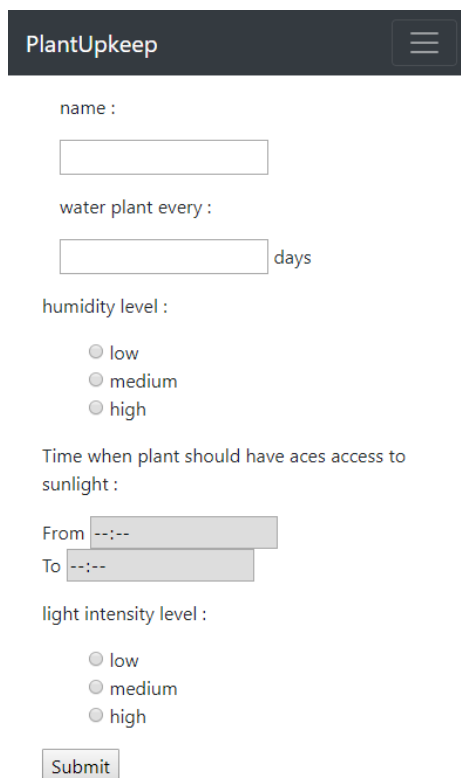
System zawiera 3 widoki



The screenshot shows the 'PlantUpkeep' application header. Below it, the 'pump : Off' section has 'on' and 'off' buttons, with the last activity timestamp '2019-11-19 14:47:10.497516'. The 'lamp : Off' section also has 'on' and 'off' buttons, with the last activity timestamp '2019-11-19 14:52:14.561608'. At the bottom, it shows 'current preset : test2'.

Home – widok w którym możemy zobaczyć aktualny stan aktorów oraz manualnie włączyć lub wyłączyć ich. Ponadto możemy zobaczyć czas ostatniego ich użycia oraz aktualnie włączone ustawienie.


Rysunek 27 wygenerowany widok „index.html”



The screenshot shows the 'Add new preset' form in the 'PlantUpkeep' application. It includes fields for 'name', 'water plant every' (in days), 'humidity level' (with radio buttons for low, medium, high), 'Time when plant should have access to sunlight' (with 'From' and 'To' time pickers), and 'light intensity level' (with radio buttons for low, medium, high). A 'Submit' button is at the bottom.

Add new preset – widok w którym możemy dodać nowe ustawienie dla rośliny.

Rysunek 28 wygenerowany widok „addPreset.html”

PlantUpkeep 

☒ example plant preset for Basil  
 sunlight : from 08:00:00 to 19:00:00  
  
 light level : high  
 water every 3 days  
 humidity level : medium

☐ test2  
 sunlight : from 08:00:00 to 17:00:00  
  
 light level : low  
 water every 1 days  
 humidity level : low

Submit

Choose preset – widok który wyświetla nam wszystkie wcześniej utworzone ustawienia i pozwala na wybranie aktualnego.

Rysunek 29 wygenerowany widok „chosePreset.html”

### 5.3.5 Algorytmy użycia aktorów

Samo korzystanie z aktorów jest dość proste: pobieramy z bazy danych docelową wartość parametru (nawodnienie lub nasłonecznienie) i jeśli sensor zwróci wartość poniżej wartości docelowej przypisany aktor jest włączany. Całe skomplikowanie programu nadzorującego polega na wyznaczeniu kiedy wspomniana wyżej operacja ma się wykonywać. Wygląda to następująco :

#### 5.3.5.1 Naświetlanie

```
def setupLamp():
    currentPreset = Models.CurrentPlant.query.first()
    presetDetails = Models.PlantPreset.query.filter_by(id=currentPreset.plantPreset).first()
    startTime = presetDetails.lampFrom
    stopTime = presetDetails.lampTo
    currentTime = datetime.now().time()
    if (currentTime > startTime and currentTime < stopTime) :
        sheduleLamp()
    schedule.every().days.at(str(startTime)).do(sheduleLamp)
    schedule.every().days.at(str(stopTime)).do(cancelSheduleLamp)

def sheduleLamp():
    schedule.every(5).minutes.do(Raspi.ilumantion).tag('lamp')

def cancelSheduleLamp():
    Raspi.turnOffLamps()
    schedule.clear('lamp')
```

Rysunek 30 fragment pliku „loop.py”

Mamy tutaj funkcję ustawiającą (setupLamp) oraz dwa zdarzenia (sheduleLamp i cancelSheduleLamp). Funkcja ustawiająca ma za zadanie zapisać dwa zdarzenia by wykonały się o podanych w bazie danych godzinach. Pierwsze zdarzenie co 5 minut wykonuje pomiar naświetlenia i dostosowuje stan lamp, a zadanie drugie wyłącza zadanie pierwsze by roślina w nocy mogła odpocząć od światła.

#### 5.3.5.2 Nawadnianie

```
def setupPump():
    currentPreset = Models.CurrentPlant.query.first()
    presetDetails = Models.PlantPreset.query.filter_by(id=currentPreset.plantPreset).first()
    lastWatering = datetime.now() - currentPreset.LastWatering
    daysSinceLastWatering = lastWatering.days
    if (daysSinceLastWatering >= presetDetails.wateringDays) :
        schedule.every().days.at("15:00").do(firstWatering)
        schedule.every().days.at("15:15").do(cancelShedulePump)
    else :
        waterAfter = presetDetails.wateringDays - daysSinceLastWatering
        schedule.every(waterAfter).days.at("15:00").do(firstWatering)
        schedule.every(waterAfter).days.at("15:15").do(cancelShedulePump)

def firstWatering():
    shedulePump()
    currentPreset = Models.CurrentPlant.query.first()
    presetDetails = Models.PlantPreset.query.filter_by(id=currentPreset.plantPreset).first()
    schedule.every(presetDetails.wateringDays).days.at("15:00").do(shedulePump)
    schedule.every(presetDetails.wateringDays).days.at("15:15").do(cancelShedulePump)
    return schedule.CancelJob

def shedulePump():
    schedule.every(3).seconds.do(Raspi.watering).tag('pump')

def cancelShedulePump():
    Raspi.turnOffPump()
    schedule.clear('pump')
```

Rysunek 31 fragment pliku „loop.py”

W tym przypadku też mamy funkcję ustawiającą (setupPump) oraz trzy zdarzenia (firstWatering, shedulePump oraz cancelShedulePump). Tutaj funkcja ustawiająca ma za zadanie ustawienie tylko pierwszego zdarzenia (firstWatering) ponieważ nawodnienie chcemy ustawić na x dni od poprzedniego nawodnienia a nie na x dni od aktualnej daty. Dopiero zdarzenie pierwszego nawodnienia ustawia nam regularne podlewanie rośliny co x dni (w zależności od ustawień rośliny). Sprawdzanie czujnika wilgoci odbywa się w wyznaczone dni co 3 sekundy od 15:00 do 15:15 tego samego dnia. Korekta działania pompy odbywa się o wiele częściej niż działania lamp ponieważ nadmiar wilgoci ma dużo większe znaczenie dla rośliny niż za długi czas naświetlania.

#### 5.3.6 Wykonywanie funkcji w zaplanowanym czasie

Do wykonywania zadań w danym okresie czasu wykorzystałem bibliotekę schedule stworzoną przez Daniela Badera. Biblioteka ta używa wzorca budowniczego do konfiguracji, co pozwala

zapisywać nam do planera wszelkie funkcje, bądź inne wywoływalne obiekty (callable) by wykonywały się w wybranych przez nas okresach czasu.

#### 5.3.6.1 Wzorzec „Budowniczy” (builder pattern)

Jest to wzorzec projektowy z kategorii wzorców kreacyjnych który ma na celu budowanie złożonego obiektu krok po kroku z mniejszych obiektów co daje nam większą kontrolę i przejrzystość obiektu.

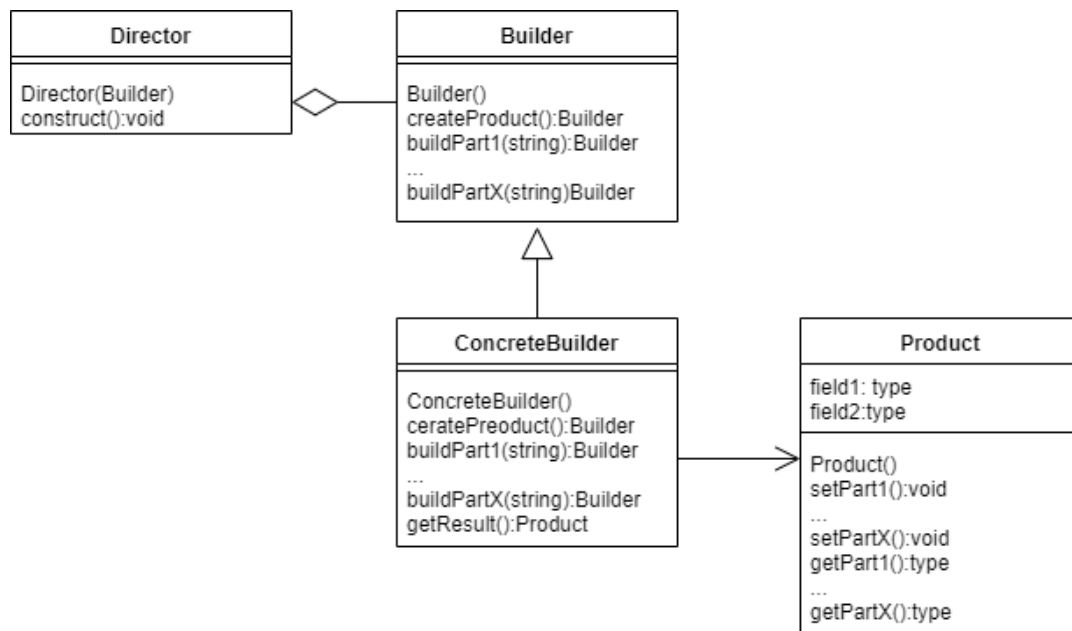
##### Zalety :

- Daje nam wyraźne rozróżnienie i dodatkową warstwę między konstrukcją i reperacją konkretnego obiektu stworzonego przez klasę.
- Pozwala na ponowne użycie kodu przy budowie różnych reprezentacji obiektu.
- Zasada pojedynczej odpowiedzialności (Single Responsibility Principle) : pozwala na izolację złożonego kodu konstrukcji od logiki biznesowej obiektu.

##### Wady:

- ogólna złożoność kodu jest większa, ponieważ wzorzec wymaga tworzenia wielu klas.

##### Diagram UML :



Rysunek 32 diagram UML (wykonanie własne)

By aktywować zadania zapisane w planerze musimy wykonać kod :

```
while True:
    schedule.run_pending()
    time.sleep(1)
```

Rysunek 33 przykład kodu

Co blokowało by serwer przed odpowiedziami na zapytania, więc trzeba ustawić pętlę nadzorującą zadania jako osobny wątek, do tego użyłem modułu „threading” który pozwala na tworzenie interfejsów wątków ponad niskopoziomowym modułem „thread”.

Dla wygody stworzyłem globalną flagę która pozwala na zakończenie wątku by ten mógł być zresetowany bądź po prostu wyłączony razem z serwerem.

```
def mainLoop():
    global loopEndFlag
    while loopEndFlag == False:
        schedule.run_pending()
        time.sleep(1)
```

Rysunek 34 fragment pliku „loop.py”

Do wyłączenia z serwera korzystamy z sygnału SIGINT (ctrl+c) więc trzeba też zadbać o to by sygnał wyłączał cały serwer w tym też pętlę nadzorującą, do tego skorzystałem z modułu „signal” który pozwala na przechwycenie sygnału i wyłączenie wszystkiego w wybranej przez programistę kolejności.

```
def signal_handler(sig, frame):
    Loop.turnOffSystem()
    sys.exit(0)
```

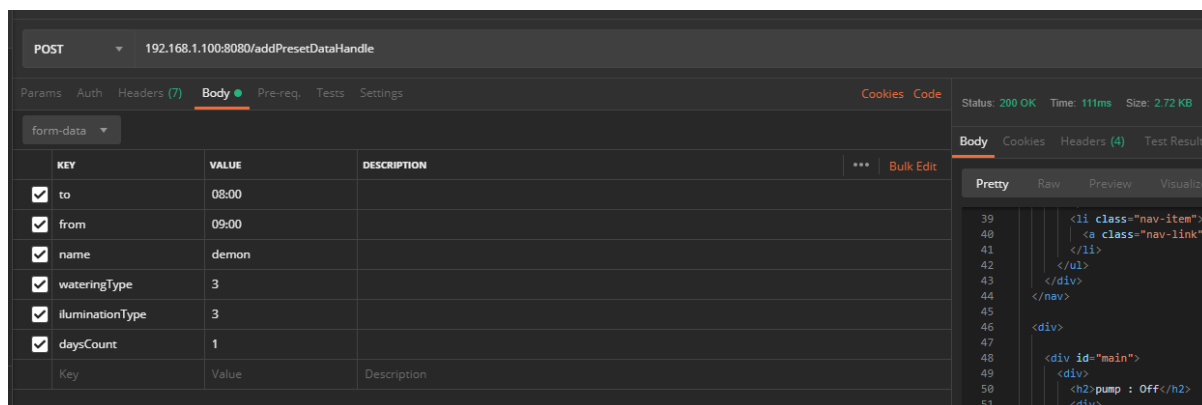
Rysunek 35 fragment pliku „main.py”

Przedstawiony kod dodaje zdarzenie wyłączenia nadzorowania i później serwera po otrzymaniu sygnału SIGINT

## 5.3.7 Komunikacja z systemem bez użycia strony internetowej

### 5.3.7.1 Przykład

Z systemem można komunikować się też przy pomocy samych żądań HTTP pozwala to na stworzenie innych programów zarządzających systemem bez potrzeby znajomości całego systemu. Jako przykład użyję żądania wysłanego z programu Postman który pozwala przetestować poszczególne endpointy.



Rysunek 36 fragment programu Postman pokazujący wysłane żądanie

W przykładzie tym wysyłamy żądanie pod URL : [adres serwera]/addPresetDataHandle używając metody POST i dołączając do ciała zapytania dane.

### 5.3.7.2 Spis endpointów

|               |                                  |            |  |
|---------------|----------------------------------|------------|--|
| Endpoint      | addPresetDataHandle              |            |  |
| Metoda        | POST                             |            |  |
| Ciało żądania | Formularz z danymi :             |            |  |
|               | Nazwa                            | Typ danych | Opis                                       |
|               | name                             | string     | Nazwa ustawienia                           |
|               | from                             | time       | Czas od którego roślina będzie naświetlana |
|               | to                               | time       | Czas do którego roślina będzie naświetlana |
|               | dayCount                         | int        | Co ile dni będzie naświetlana roślina      |
|               | illuminationType                 | int        | Id ustawień światła                        |
|               | wateringType                     | int        | Id ustawień wilgotności gleby              |
| Działanie     | Zapisuje nowe ustawienie rośliny |            |  |

|               |  |
|---------------|--|
| Endpoint      | illuminationtypes                                      |
| Metoda        | GET  |
| Ciało żądania | Brak   |
| Działanie     | Zwraca dostępne ustawienia oświetlenie w formacie JSON |

|          |               |
|----------|---------------|
| Endpoint | wateringtypes |
|----------|---------------|

|               |  |
|---------------|--|
| Metoda        | GET  |
| Ciało żądania | Brak   |
| Działanie     | Zwraca dostępne ustawienia nawodnienia w formacie JSON |

|               |                             |            |               |
|---------------|-----------------------------|------------|---------------|
| Endpoint      | changeplantsettingshandler  |            |               |
| Metoda        | POST                        |            |               |
| Ciało żądania | Formularz z danymi :        |            |               |
|               | Nazwa                       | Typ danych | Opis          |
|               | preset                      | int        | Id ustawienia |
| Działanie     | Ustawia aktualne ustawienie |            |               |

|               |  |
|---------------|--|
| Endpoint      | getplantsettings                                     |
| Metoda        | GET  |
| Ciało żądania | Brak   |
| Działanie     | Zwraca wszystkie zapisane ustawienia w formacie JSON |

|               |                             |
|---------------|-----------------------------|
| Endpoint      | on                          |
| Metoda        | PUT                         |
| Ciało żądania | Brak                        |
| Działanie     | Włącza nadzorowanie rośliny |

|               |                              |
|---------------|------------------------------|
| Endpoint      | off                          |
| Metoda        | PUT                          |
| Ciało żądania | Brak                         |
| Działanie     | Wyłącza nadzorowanie rośliny |

|               |              |
|---------------|--------------|
| Endpoint      | lamp/on      |
| Metoda        | PUT          |
| Ciało żądania | Brak         |
| Działanie     | Włącza lampy |



|               |               |
|---------------|---------------|
| Endpoint      | lamp/off      |
| Metoda        | PUT           |
| Ciało żądania | Brak          |
| Działanie     | Wyłącza lampy |

|               |              |
|---------------|--------------|
| Endpoint      | pump/on      |
| Metoda        | PUT          |
| Ciało żądania | Brak         |
| Działanie     | Włącza pompę |

|               |               |
|---------------|---------------|
| Endpoint      | pump/off      |
| Metoda        | PUT           |
| Ciało żądania | Brak          |
| Działanie     | Wyłącza pompę |

### 5.3.8 Testy

#### 5.3.8.1 Testy API

Do stworzenia automatycznych testów API w Postman`ie tworzymy tzw. kolekcje, kolekcja ma swoje zmienne oraz listę żądań. Każde żądanie zawiera :

- URL
- Nagłówek
- Ciało
- Metodę
- Testy

żądanie zawsze wywoływane są w tej samej kolejności ponieważ niektóre zmienne mogą być zmieniane przez odpowiedź na żądanie. Dobrze jest mieć testy każdego z endpointów z dwóch powodów

- Po zmianie kodu od razu wiemy gdy coś nie działa.
- Jest to swojego rodzaju dokumentacja jako że poza samym spisem endpointów mamy też przykładowe dane wejściowe.

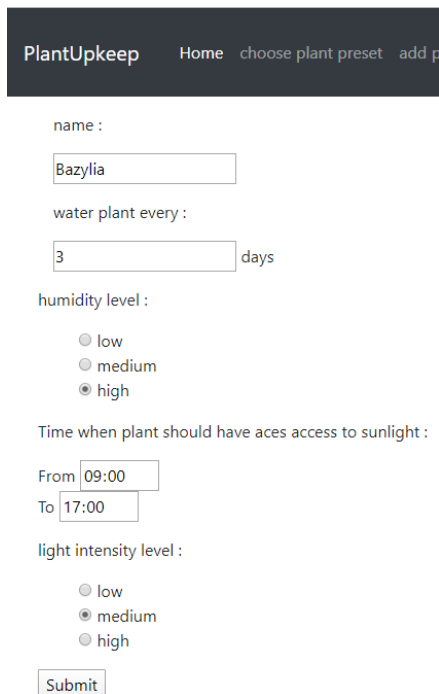
W testach możemy sprawdzić :

- kod odpowiedzi
- wartości w nagłówku
- wartości w ciele
- czas wykonania

Co pozwala nam na zapewnienie że API spełnia wymagania funkcjonalne jak i nie funkcjonalne.

## 6 Dwa tygodnie działania systemu

Na roślinie testową wybrałem bazylię. Jest to wymagająca roślina która przez swoją wrażliwość na nadmiar oraz niedomiar wody i światła powinna pokazać ewentualne mankamenty systemu.



The screenshot shows the PlantUpkeep web application interface. At the top, there is a navigation bar with the logo 'PlantUpkeep' and links for 'Home', 'choose plant preset', and 'add p'. The main form is titled 'name :' and contains a text input field with 'Bazylia' entered. Below this is a 'water plant every :' section with a numeric input field set to '3' and the unit 'days'. The 'humidity level :' section has three radio button options: 'low', 'medium', and 'high', with 'high' selected. The 'Time when plant should have access to sunlight :' section has two time input fields: 'From' set to '09:00' and 'To' set to '17:00'. The 'light intensity level :' section has three radio button options: 'low', 'medium', and 'high', with 'medium' selected. At the bottom of the form is a 'Submit' button.

Bazylia ma większe niż inne zioła zapotrzebowanie na wodę, jest jednak wrażliwa zarówno na jej brak jak i nadmiar. Najlepiej roślinę regularnie, ale z umiarem podlewać, aby utrzymać stałą wilgotność podłoża. Uwaga, zbyt dużo wody powoduje gnicie.

Jako roślina południowa bazylia preferuje ciepłe i słoneczne miejsce, osłonięte od wiatru. Im więcej światła, tym więcej olejku eterycznego i silniejszy aromat liści. Z moich obserwacji wynika jednak, że dobrze rośnie też w półcieniu, a zbyt ostre słońce na południowym balkonie nie do końca jej służy – ziemia szybko wysycha, a liście bledną.[2]

Data posadzenia :

## 7 Plany na rozwój

Aktualnie kod nadzorujący jest integralną częścią serwera. W przyszłości zamierzam sprawić by program nadzorujący był niezależną częścią umieszczoną na platformie Arduino która będzie komunikować się z głównym serwerem który będzie zawierać ustawienia i z którym komunikować będzie się użytkownik. Pozwoli to na nadzorowanie większej ilości roślin przez jeden serwer, aktualnie serwer może obsłużyć do 5 roślin ze względu na ograniczoną liczbę pinów.

Chciałbym też dodać możliwości kontroli większej ilości parametrów takich jak : wilgotność powietrza, mieszanka powietrza. Ale wymaga to hermetycznego środowiska oraz więcej wiedzy o danej roślinie.

Ważne jest by przedłużyć żywotności czujnika wilgoci gleby poprzez dawanie mu zasilania wyłącznie w czasie pomiaru wartości, co trzeba zrobić odpowiednio wcześniej by mcp3008 nie pobrał błędnego wyniku ze względu na swoje opóźnienie.

Zabezpieczenie danych oraz połączenia z serwerem.

Zbieranie informacji o roślinie i wyświetlanie ich w postaci wykresu na stronie głównej.

## 8 Publikacje

### 8.1 Strony

<https://www.sqlalchemy.org/>

<https://www.sqlitetutorial.net>

<https://schedule.readthedocs.io/en/stable/>

<https://raspberrypi-aa.github.io/session3/spi.html>

<https://refactoring.guru/design-patterns/builder>

<http://www.przesadzilam.pl/moj-balkon/uprawa-bazylii-w-doniczce/>[2]

### 8.2 Książki

Samouczek HTML – Karol Wierzchołowski

Learning Python 5ed – Mark Lutz

Mastering the Raspberry Pi - Warren Gay

### 8.3 Artykuły

PYTHON: A PROGRAMMING LANGUAGE FOR SOFTWARE INTEGRATION AND DEVELOPMENT -  
M. F. SANNER[1]