

Sebastian Dyjeta

nr albumu: 39258

kierunek studiów: informatyka

specjalność: Systemy 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 wyboru tematu pracy: 31.01.2019

Data dopuszczenia pracy do egzaminu:
(uzupełnia pisemnie Dziekanat)

Szczecin, 2020

Oświadczenie autora pracy dyplomowej

Oświadczam, że praca dyplomowa inżynierska pn.

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

napisana pod kierunkiem

dr. inż. Sławomira Jaszczała

jest w całości moim samodzielnym autorskim opracowaniem sporządzonym przy wykorzystaniu wykazanej w pracy literatury przedmiotu i materiałów źródłowych. Złożona w dziekanacie Wydziału Informatyki treść mojej pracy dyplomowej w formie elektronicznej jest zgodna z treścią w formie pisemnej.

Oświadczam ponadto, że złożona w dziekanacie praca dyplomowa ani jej fragmenty nie były wcześniej przedmiotem procedur procesu dyplomowania związanych z uzyskaniem tytułu zawodowego w uczelniach wyższych.

Sebastian Dyjda
podpis dyplomanta

Szczecin, dn. 28.04.2020

Spis treści

1	WSTĘP	4
1.1.1	<i>Zakres pracy.....</i>	4
1.1.2	<i>Zawartość pracy.....</i>	4
1.2	WERSJA ANGIELSKA	5
1.2.1	<i>Scope of work</i>	5
1.2.2	<i>Summary of sections</i>	5
1.3	ZAGADNIENIE HODOWLI ROŚLIN DOMOWYCH	6
1.4	PRZEGLĄD ZAUTOMATYZOWANYCH SYSTEMÓW DOMOWYCH.	7
1.5	PODOBNE PRODUKTY	8
1.6	UZASADNIENIE WYBORU TEMATU PRACY	14
1.7	WYMAGANIA SYSTEMU	14
1.7.1	<i>Wymagania funkcjonalne.....</i>	14
1.7.2	<i>Wymagania нефункционалне.....</i>	15
1.8	UŻYTE PROGRAMY	15
1.9	UŻYTE TECHNOLOGIE	16
2	IMPLEMENTACJA SYSTEMU	19
2.1	HARDWARE	19
2.1.1	<i>Analiza wymagań.....</i>	19
2.1.2	<i>Projekt systemu sterowania</i>	20
2.1.3	<i>Spis komponentów.....</i>	23
2.1.4	<i>Opis komponentów.....</i>	23
2.1.5	<i>Schemat układu</i>	30
2.2	SOFTWARE	33
2.2.1	<i>Konfiguracja serwera</i>	33
2.2.2	<i>Baza danych.....</i>	36
2.2.3	<i>API.....</i>	39
2.2.4	<i>Widoki</i>	41
2.2.5	<i>Algorytmy użycia aktorów.....</i>	44
2.2.6	<i>Wykonywanie funkcji w zaplanowanym czasie</i>	46
2.2.7	<i>Komunikacja z systemem bez użycia strony internetowej.....</i>	47
2.2.8	<i>Testy.....</i>	51
3	PODUMOWANIE	53
4	LITERATURA.....	54
5	SPIS ILUSTRACJI	56

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 osadzona 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, Obejmuje on opis stworzonych komponentów i wszelkie informacje potrzebne do zrozumienia czy odtworzenia projektu.

Słowa kluczowe : Raspberry PI, Python, REST API

1.1.1 Zakres pracy

- Opis użytych narzędzi.
- Opis założeń i wymagań systemu.
- Przygotowanie elementów sterujących.
- Konfiguracja serwera.
- Utworzenie bazy danych.
- Implementacja aplikacji internetowej.
- Implementacja API.
- Implementacja algorytmów użycia urządzeń elektronicznych.
- Przeprowadzenie testów API.
- Przeprowadzenie testów systemu.

1.1.2 Zawartość pracy

- Wstęp : Tutaj można znaleźć podstawowe założenia systemu, porównanie podobnych rozwiązań na rynku, oraz użyte programy i narzędzia które pozwoliły na stworzenie opisywanego systemu.
- Implementacja systemu : dział opisujący system składający się z dwóch części :
 - Hardware : Szczegółowy opis komponentów i ich połączenia.
 - Software : Opis utworzenia software`owej części systemu wraz z konfiguracją serwera. Opisuje od podstaw jak stworzyć serwer z REST`owym API, jak komunikować się z urządzeniami, oraz jaka logika stoi za działaniem systemu.

- Podsumowanie : spis rzeczy, które zaplanowano do dodania lub poprawienia.

1.2 Wersja angielska

The purpose of the described system is to automate process of plant upkeep. The System allows user to adjust the parameters via a website or Web API that allows an external application to use system. The supervisory system is designed to irrigate the plant and ensure its exposure to light in the designated time frames. The system is placed on the Raspberry Pi platform and the Flask miniframework. The system is intended for houseplants, which are reflected in the power and size of lamps and pump.

This document is a form of documentation of the created system, it contains a description of the created components and all information needed to understand or reconstruct the project.

Keywords : Raspberry PI, Python, REST API

1.2.1 Scope of work

- Description of used tools.
- Description of system requirements.
- Server configuration.
- Database creation.
- Web application implementation.
- API implementation.
- Implementation of algorithms for the use of electronic devices
- Preforming API test.
- Preforming system test.

1.2.2 Summary of sections

- Introduction : Here you can find basic system requirements, comparison of similar solutions on the market, used program and tools that allowed for creation of the described system.
- System implementation: section describing the system consisting of two parts :
 - Hardware : Detailed description of components and their connections.
 - Software : Description of creating the software part of the system along with server configuration. It describes from scratch how to create a server with REST API, how to communicate with devices, and what logic is behind the operations in the system.
- Summary : List of things that it plans to improve or add.

1.3 Zagadnienie hodowli roślin domowych

Rośliny doniczkowe w domu uspokajają, produkują tlen, oczyszczają powietrze i pomagają w utrzymaniu dobrego nawilżenia powietrza, co więcej poprawiają estetykę otoczenia i niektóre części roślin mają swoje zastosowanie w kuchni.

Każdy gatunek roślin doniczkowych uprawianych w domu potrzebuje indywidualnego podejścia, są jednak pewne ogólne zasady, których należy przestrzegać w stosunku do większości roślin doniczkowych.

Nawodnienie

Rośliny nie należy podlewać ani zbyt często, ani zbyt mocno, niedotrzymanie tych założeń kończy się gniciem korzeni i w wyniku śmiercią rośliny. Problemатyczne też jest określenie, czy roślina potrzebuje podlania. Sprawdzenie palcem czy gleba jest wystarczająco wilgotna jest o tyle ciężkie, że gleba nie jest wilgotna tak samo w każdej swojej warstwie, na powierzchni gleba jest mniej wilgotna co znaczy, że by sprawdzić glebę bez narzędzi nie dość że musimy mieć dobre wyczucie, ale też nie obejdzie się bez ubrudzenia. Większość roślin nie może być podlewana z góry ponieważ zamoczenie liści ma negatywny skutek w połączeniu z naświetleniem, liście doznają oparzenia co sprawia, że usychają.

Oświetlenie

Rośliny rosną najlepiej, gdy mają stałe godziny naświetlenia. Stałe naświetlenie może być przerywane przez chmury, wcześniejsze zachodzenie słońca w niektórych częściach świata, czy zastönione przez przedmioty takie jak żaluzje czy szafa. Pomagają w tym specjalne lampy przystosowane do imitowania promieni słonecznych, należy jednak pamiętać, że za długie naświetlanie kończy się przemęceniem rośliny, co zazwyczaj kończy się tym, że roślina jest duża, ale traci swoje właściwości takie jak kolor i smak, często też roślina hodowana w ciągle naświetlonym środowisku jest na tyle duża i słaba, że łamie się pod własnym ciężarem.

Temperatura

Większość gatunków uprawianych w pomieszczeniach pochodzi z rejonu tropikalnego i subtropikalnego. W praktyce oznacza to, że u klimacie będącym w Polsce najlepiej rozwijają się przy dużej ilości światła, wysokiej temperaturze latem (pokojowej) i niskiej zimą oraz znacznej wilgotności powietrza.

O ile od wiosny do jesieni, nie ma z tym czynnikiem problemu, zimą zieleń powinna przejść okres spoczynku. Warto więc przenieść rośliny do chłodnego pomieszczenia (13-16 °C). Niezdrowe dla roślin jest też hodowanie w pomieszczeniu z przeciągiem.

Gleba

Gleba wpływa na wzrost i rozwój roślin. Dobra ziemia do kwiatów doniczkowych musi zapewniać prawidłowy drenaż, dobre krążenie powietrza i składniki pokarmowe. Musi być luźna i przewiewna, aby korzenie mogły swobodnie rosnąć i mieć dostęp do powietrza.

W zależności od rośliny, roślina powinna rosnąć w glebie z odpowiednim składem podłoża, ponieważ potrzeby pokarmowe poszczególnych grup roślin różnią się między sobą. Ziemia do roślin doniczkowych powinna uwzględniać wymagane pH podłoża dla roślin z danej grupy, wymagania co do utrzymywania wilgoci, przewiewności czy zasobności w składniki odżywcze. Dla większości roślin można znaleźć gotowe mieszanki pozwalające na optymalny wzrost.

Donica

Donica dla rośliny powinna być odpowiednio duża, na tyle by jej korzenie mogły się odpowiednio rozrosnąć (co zależy od rośliny). Ważnym elementem donicy są otwory na dole do przepuszczania wody, brak takich otworów zwiększa wilgoć w dolnej warstwie gleby co prowadzi do gnicia korzeni. Rośliny należy przesadzać średnio co 2 lata do większej donicy.

1.4 Przegląd zautomatyzowanych systemów domowych.

Systemy automatycznej hodowli roślin domowych mogą stanowić elementy automatyki budynkowej w ramach tzw. smart home i być uzupełnieniem już istniejących układów np. systemów kuchennych takich jak Termomix (urządzenie wielofunkcyjne) czy maszyn do Sous Vide (wolnego gotowania). Urządzeń do zautomatyzowanej pielęgnacji roślin nie ma na rynku wiele (kilka produktów opisanych jest dalej), a istniejące rozwiązania zostawiają miejsce na udoskonalenia.

W praktyce zautomatyzowane systemy domowe zwane smart home opierają się o technologie mobilne tj. zazwyczaj sieci radiowe. Zaletą zastosowania urządzeń mobilnych (telefonów, tabletów i innych) do sterowania urządzeniami domowymi jest ich uniwersalność i popularność. Ma to w sobie kilka zalet z perspektywy tworzenia systemów uwzględniających te urządzenia. Jedną z zalet jest to, że użytkownik większość czasu posiada jakiś rodzaj urządzenia mobilnego przy sobie, jest do niego przyzwyczajony i wie jak z niego korzystać. Kolejną zaletą jest to, że zmniejsza to koszty samego systemu ponieważ nie ma potrzeby tworzenia dedykowanego urządzenia do interakcji z systemem.

Zazwyczaj do każdego z tak kontrolowanych systemów istnieje dedykowana aplikacja co pozwala na łatwiejsze przewidzenie efektów działania, ale też wprowadza ograniczenie do konkretnego systemu/urządzenia pod które aplikacja jest napisana.

Istnieją narzędzia pozwalające na tworzenie aplikacji mult-platformowych takie jak Xamarin (do tworzenia aplikacji na telefony), czy .NET Core (do tworzenia aplikacji na komputery). Innym rozwiązaniem problemu obsługi kilku platform jest stworzenie strony internetowej, gdyż strona internetowa może być uruchomiona na każdym urządzeniu wyposażonym w przeglądarkę internetową. Należy pamiętać, że pomimo możliwości otworzenia strony internetowej na różnych urządzeniach, nie na każdym urządzeniu strona będzie wyglądać tak samo, więc należy wziąć to pod uwagę zarówno przy tworzeniu, jak i testowaniu danego rozwiązania.

1.5 Podobne produkty

Automated Indoor Gardener



Rys. 1. Zdjęcie poglądowe urządzenia Automated Indoor Gardener [1.2]

Stworzony przez grupę Hacker Shack. Urządzenie dbające o to, żeby w danych godzinach włączała się lampa oraz pompa nawadniająca

Zalety:

- Urządzenie jest kompaktowe.

- Można dostosować nachylenie lampy.
- Najtańsze urządzenie w zestawieniu (około 200zł).
- Projekt jest otwartoźródłowy, co pozwala na rozwój urządzenia we własnym zakresie.

Wady:

- Nie ma żadnych urządzeń pomiarowych sprawdzających czy kwiat faktycznie potrzebuje nawodnienia czy nasłonecznienia.
- Brak interfejsu umożliwiającego konfigurację ustawień, żeby zmienić godziny nasłonecznienia, bądź nawadniania potrzebujemy rozkręcić obudowę i podłączyć się bezpośrednio do Raspberry PI.
- Nie jest to produkt komercyjny, co oznacza że nie ma możliwości kupna gotowego urządzenia.

Źródło : <https://www.hackster.io/hackershack/automated-indoor-gardener-a90907>

FarmBot



Rys. 2. Zdjęcie poglądowe urządzenia FarmBot [1.3]

Stworzony przez grupę o tej samej nazwie. Urządzenie wielofunkcyjne pozwalające na automatyczne : sadzenie roślin, podlewanie, usuwanie chwastów.

Cena : 7461 zł – 22840 zł (w zależności od modelu)

Zalety:

- Dobrze zaprojektowany interface użytkownika.
- Urządzenie automatycznie sadi rośliny.
- Opcja wykrywania i usuwania chwastów
- Jedno urządzenie może obsłużyć kilka rodzajów roślin.
- Projekt jest otwarcie źródłowy.

Wady:

- Wysoka cena.
- Brak regulacji oświetlenia.
- Rozmiar – najmniejsza wersja ma rozmiar 1,5m x 3m x 1,5m

Źródło : <https://farm.bot/>

AeroGarden Harvest



Rys. 3. Zdjęcie poglądowe urządzenia Aerogarden Harvester [1.4]

Stworzony przez firmę Miracle-gro. Urządzenie powiadamia użytkownika o potrzebie podlania oraz o potrzebie włączenia/wyłączenia lampy.

Cena : 730zł

Zalety:

- Dostosowywana wysokość lampy.
- Informowanie użytkownika.

Wady:

- Mały pojemnik na wodę.
- Za mocne światło wysuszające roślinę.
- Brak automatyzacji.

- Zwykła lampa.
- Powiadomienia przychodzą niezależnie od godziny co sprawiało, że użytkownicy często dostawali powiadomienia w godzinach nocnych

Źródło : <https://www.aerogarden.com/>

SmallGarden 2



Rys. 4. Zdjęcie poglądowe urządzenia Smallgarden 2 [1.5]

Stworzony przez firmę EDN. W tym produkcie ciekawe jest zastosowanie gotowych pojemników z nasionami i ziemią co sprawia, że użytkownik nie musi mieć dużej wiedzy na temat rośliny.

Cena : 764 zł

Zalety:

- Powiadomienia o potrzebie podlania.
- Włączenie/wyłączenie lamp w odpowiednich godzinach.
- Konfigurowalne przez urządzenia mobilne.

Wady:

- Nisko zawieszona lampa, co nie pozwala roślinie wyrosnąć.
- Brak automatycznego podlania.

- Brak sensorów światła.

Źródło : <https://www.edntech.com/pages/smallgarden>

Smart Garden 9



Rys. 5. Zdjęcie poglądowe urządzenia Smart Garden 9 [1.6]

Stworzony przez firmę Click and Grow. Najbardziej polecany produkt, automatyzuje proces podlewania i oświetlania rośliny. Tak jak poprzedni produkt korzysta z specjalnie przygotowanych roślin.

Cena : 899 zł

Zalety:

- Wysoko osadzone lampy.
- Specjalne lampy do roślin.
- Automatyczne dostosowywanie ustawień roślin na podstawie pojemnika
- Rozbudowana, wizualnie przyjemna aplikacja do interakcji z urządzeniem.

Wady:

- Możliwość sadzenia jedynie roślin od producenta.

Źródło : <https://sklep.smart-garden.pl/Smart-Garden-9-Dark-Grey>

Na podstawie analizy cech funkcjonalnych dostępnych na rynku urządzeń wyprowadzono następujące wnioski, które zostaną wykorzystane w trakcie realizacji projektu:

Oświetlenie

Jedną z ważniejszych funkcjonalności jakie zapewniają takie urządzenia to oświetlenie rośliny, by było to optymalne należy użyć do tego specjalnych lamp (jak w przypadku SmartGarden 9) oraz umieścić je po bokach rośliny, co nie będzie ograniczało jej wzrostu jak w przypadku większości modeli. Ponadto należy zadbać by roślina dostawała dodatkowe oświetlenie wyłącznie w godzinach, których go potrzebuje oraz wtedy, gdy światło słoneczne nie będzie dla danej rośliny wystarczające, co nie było zaadresowane w żadnym z przeglądanych produktów.

Nawodnienie

Musi odbywać się automatycznie oraz tylko wtedy, gdy roślina tego potrzebuje.

Konstrukcja urządzenia

Większość produktów daje gotowe obudowy, co ogranicza wielkość roślin. Projektowane urządzenie powinno być zbiorem komponentów, które można dodać do większości zwykłych doniczek co pozwoli na nieograniczanie się, co do wielkości czy rodzaju roślin.

Interakcja użytkownika z urządzeniem

Konfiguracja urządzenia powinna być możliwa z urządzeń mobilnych niezależnie od systemu. Komunikaty nie mogą być natarczywe. Informacje przekazywane użytkownikowi powinny być pogrupowane by użytkownik nie czuł natłoku informacji.

1.6 Uzasadnienie wyboru tematu pracy

Po analizie dostępnych na rynku produktów stwierdzono, że możliwe będzie stworzenie prototypu o poprawionych cechach funkcjonalnych.

Proces projektowania urządzenia będzie również okazją do pogłębienia wiedzy z zakresu automatyzacji procesów jak i komunikacji w systemach automatycznych. Urządzenie wpisuje się w aktualne trendy tj. IoT (ang. Internet of Things), smart home.

1.7 Wymagania systemu

1.7.1 Wymagania funkcjonalne

- Aplikacja do kontroli urządzenia musi być dostępna przez przeglądarki internetowe.
- Aplikacja musi skalować się pod urządzenia mobile.

- Aplikacja pozwala na komunikację z systemem przez Web API.
- 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 wilgotność gleby oraz natężenie nasłonecznienia.
- System mierzy siłę nasłonecznienia w więcej niż jednym punkcie.
- System wymaga maksymalnie dwóch 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.

1.7.2 Wymagania нефункционалне

- System jest w stanie działać nieprzerwanie przynajmniej przez dwa tygodnie.
- Odpowiedz na każde żądanie odbywa się w czasie mniejszym niż 200 ms.

1.8 Użyte Programy

Visual Studio Code

Jest edytorem kodów źródłowych stworzonym przez firmę Microsoft. Podstawowa wersja zajmuje niewiele miejsca na dysku, ponieważ edytor pozwala na dodanie rozszerzeń takich jak obsługa kontroli wersji, podpowiadanie składni itd. Przez to przed przystąpieniem do projektu należy poświęcić nieco uwagi rozszerzeniom, których potrzebujemy. Wybór narzędzi jest przez to bardziej świadomy, co zapewnia, że żadna funkcjonalność narzędzia nie ujdzie uwadze użytkownika. Co ważne edytor działa

na wszystkich popularnych systemach operacyjnych, więc preferencje użytkownika i zestaw wtyczek może być przenoszony na inne stacje robocze niezależnie od obsługiwanego systemu.

FileZilla Client

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

Putty

Jest oprogramowaniem emulującym terminal, co pozwala na połączenie przez protokoły takie jak SCP/SSH/Telnet. Oprogramowanie zostało wykorzystane do komunikacji z serwerem, na którym znajdował się system. Do korzystania z systemu program nie jest potrzebny, używany był jedynie do wstępnej konfiguracji serwera oraz testowania systemu.

Postman

Oprogramowanie do testowania Web API, bardzo przydatne do testowania poszczególnych endpointów (miejsc do których odwołuje się użytkownik chcący wywołać konkretną reakcję serwera) przy tworzeniu Web API, ale też do tworzenia zautomatyzowanych testów Web API, które pomogą zapewnić poprawne działanie Web API, nawet przy długotrwałym rozwoju oprogramowania przez kilka osób.

1.9 Użyte Technologie

HTML

HTML (ang. 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.

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 prostą i elegancką składnię. Jest językiem działającym na wielu platformach. Został zaprojektowany w 1990 roku przez Guido van Rossum. Jak wiele innych języków skryptowych Python jest darmowy nawet w zastosowaniach komercyjnych. Kod

Python`owy jest kompilowany automatycznie przez interpreter do niezależnego od platformy, wykonywalnego kodu bajtowego. Python z założenia jest modularny, kernel jest niewielki i może być rozszerzony przez dodanie modułów. 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.

Python jest językiem otwarto-źródłowym co jest jednym z powodów dla którego gromadzi się wokół niego dużą społeczność rozwijająca technologie, ale też dodającą nowe moduły i inne narzędzia przyspieszające proces tworzenia aplikacji oraz jej optymalizacji, co często pozwala na budowanie dużych aplikacji w krótkim czasie.

Flask

Flask jest prostym frameworkiem, który szybko pozwala na utworzenie API aplikacji napisanie w Pythonie. Flask jest microframeworkiem, co znaczy, że nie wymaga żadnych zewnętrznych narzędzi ani bibliotek. Nie ma on walidacji formularzy weryfikacji użytkownika, ORMów itd. Zapewnia tylko podstawową funkcjonalność, co pozwala na napisanie tych funkcjonalności samodzielnie bądź na skorzystanie z wybranych bibliotek. Zainteresowani bardziej skomplikowanym framworkiem do tworzenia Web API w Pythonie mogą zapoznać się z Django [<https://www.djangoproject.com>]. Django nawet w podstawowej formie zawiera szeroki zakres funkcjonalności co sprawia, że używa dużo zasobów w porównaniu z Flaskiem. Django wymaga większej wiedzy potrzebnej do skonfigurowania podstawowej aplikacji webowej oraz generuje dużą ilość plików. Do opisywanego projektu wybrany został Flask, ponieważ wiele funkcjonalności Django nie byłoby wykorzystanych, co niepotrzebnie komplikowałoby projekt oraz zwiększyło by użycie zasobów.

SQLAlchemy

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

SQLite

SQLite jest systemem do zarządzania bazami danych. Jest wyposażony w 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 (ang. atomicity, consistency, isolation, durability), więc w przypadku wielu zapytań lub niepowodzenia operacji, dane w bazie nie zostaną uszkodzone.

GIT

GIT jest systemem kontroli wersji co oznacza, że śledzi wszelkie nasze zmiany w kodzie źródłowym od początku naszego projektu. Pozwala 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.

2 Implementacja systemu

2.1 Hardware

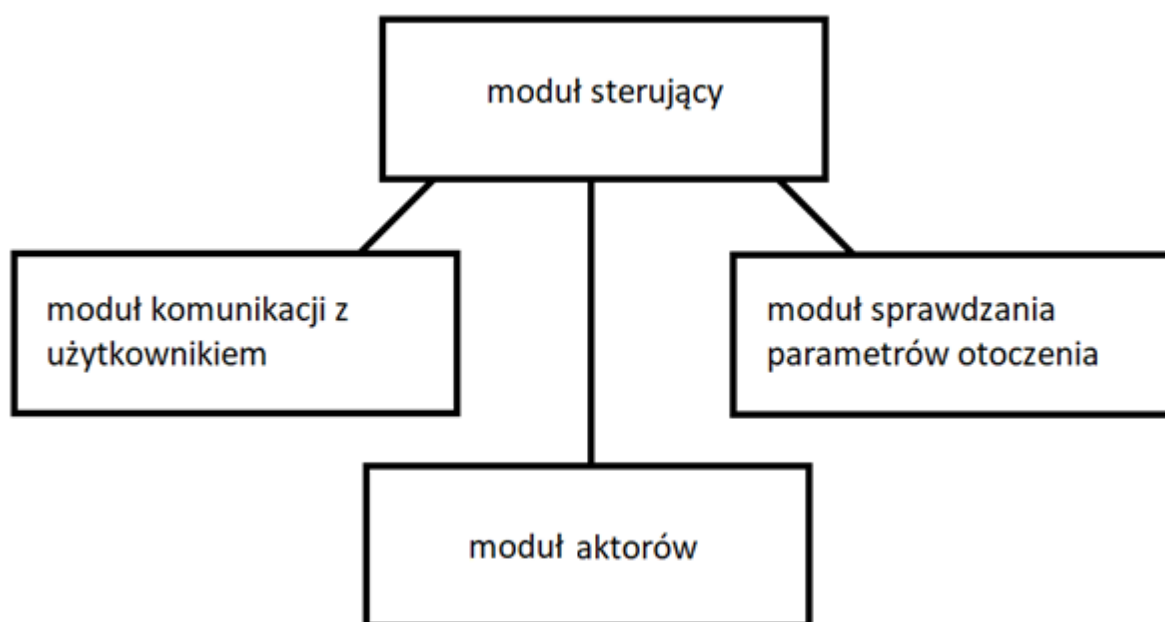
2.1.1 Analiza wymagań

Zadaniem systemu jest zapewnienie roślinie optymalnych warunków wzrostu, przez cykliczne zapewnienie jej odpowiedniego naświetlenia oraz nawodnienia.

Na podstawie założeń wstępnych określone następujące funkcje systemu :

- pomiar wilgotności gleby
- pomiar naświetlenia
- sterowanie pompą wodną
- sterowanie oświetleniem
- komunikacja z użytkownikiem – ustawienia parametrów rośliny oraz sprawdzanie stanu układu
- sygnalizacja użytych aktorów

Schemat obrazujący poszczególne funkcje urządzenia :



Rys. 6. Schemat funkcji [wykonanie własne]

Specyfikacja wymagań systemu sterującego funkcjami urządzenia

Szczegółowe funkcje urządzenia :

- załączenie systemu
 - warunki wstępne :
 - właściwie działający moduł sterowania (obecność zasilania, dostępność połączenia do sieci lokalnej)
- pomiar naświetlenia

- musi odbywać się przez urządzenie niezależne od sterownika,
- napięcie zasilające urządzenia pomiarowe = 3.3VDC,
- warunki wstępne :
 - włączony system
 - dostępne dwa fotorezystory
- pomiar wilgotności gleby
 - musi odbywać się przez urządzenie niezależne od sterownika,
 - napięcie zasilające urządzenia pomiarowe = 3.3VDC,
 - warunki wstępne :
 - włączony system
- komunikacja z użytkownikiem (czynności użytkownika):
 - dodanie ustawień rośliny do bazy danych
 - wybór ustawień rośliny
 - włączenie/wyłączenie nadzorowania rośliny
 - włączenie/wyłączenie pompy
 - włączenie/wyłączenia lamp
 - pobranie aktualnego stanu systemu (pomiar, aktualnie używane aktory)
 - warunki wstępne :
 - włączony system
 - urządzenie z połączeniem do sieci lokalnej
- obsługa pompy
 - napięcie zasilające urządzenie = 5VDC
 - pompa może być włączona/wyłączona przez program nadzorujący lub przez użytkownika
 - warunki wstępne :
 - włączony system
 - dostępna woda w zbiorniku
- obsługa lamp
 - napięcie zasilające urządzenia = 12VDC
 - lampy mogą być włączone/wyłączone przez program nadzorujący lub przez użytkownika
 - warunki wstępne :
 - włączony system

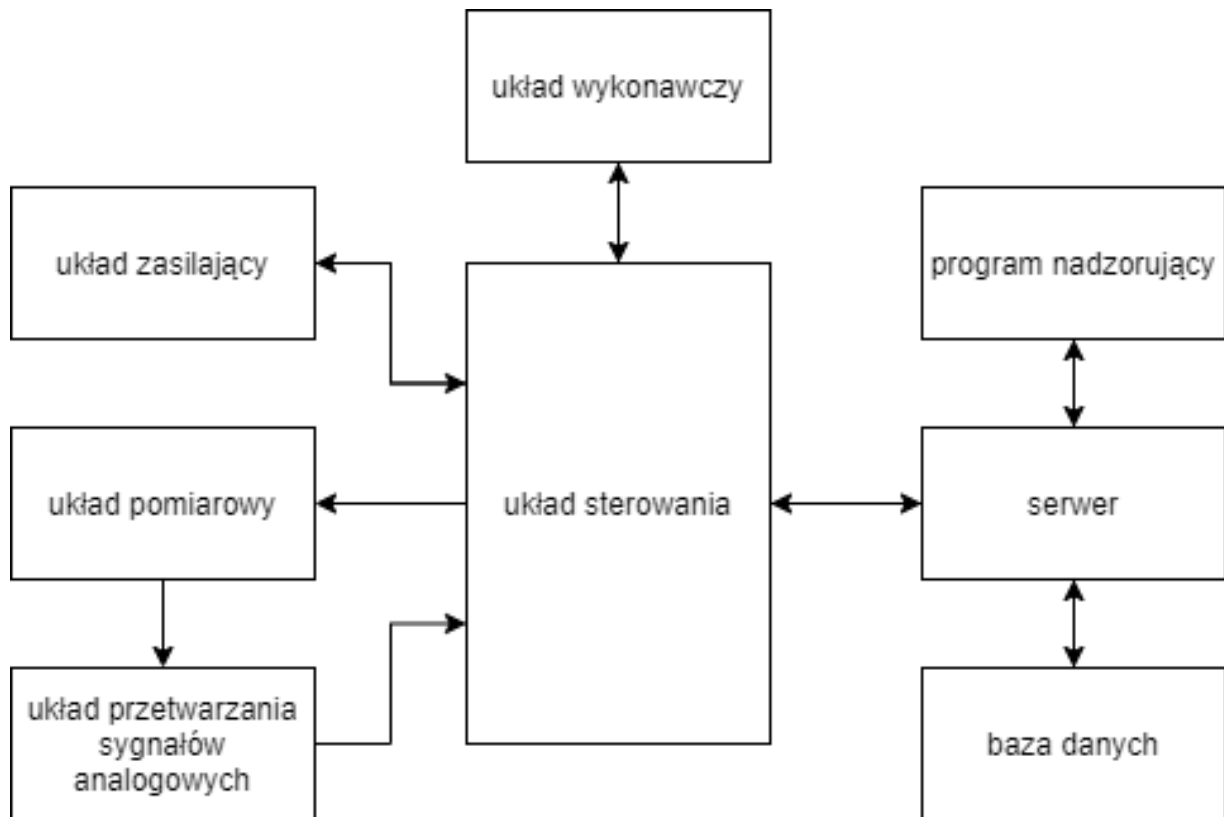
2.1.2 Projekt systemu sterowania

Dekompozycja systemu sterowania na funkcjonalne moduły

System można podzielić na szereg odrębnych, zależnych funkcjonalnie od siebie komponentów:

- układ sterowania
- układ zasilający
- układ pomiarowy
- układ wykonawczy
- układ przetwarzania sygnałów analogowych

- serwer
- baza danych
- program nadzorujący potrzeby rośliny



Rys. 7. Schemat modułów funkcjonalnych [wykonanie własne]

Opis funkcji poszczególnych komponentów systemu :

- układ sterowania

Konstrukcję oparto o platformę Raspberry Pi. W układzie znajduje się program odpowiedzialny za komunikację pozostałych elementów. Po zsumowaniu wszystkich wejść i wyjść poszczególnych komponentów systemu, okazało się, że Raspberry Pi musi mieć wolne co najmniej 3 wyjścia binarne oraz magistrale SPI do odbierania sygnałów analogowych. Ze względu na rodzaj przetwarzanych sygnałów wyjścia binarne mają być typu tranzystorowego.

- układ zasilający

W podanym rozwiązaniu zostaną zastosowane dwa układy zasilające dla układu lamp oraz pozostałych części układu. W obu przypadkach dostarczone napięcie zasilania powinno wynosić 12 VDC. W przypadku lamp zastosowano zasilacz 12VDC / 1.5A. Natomiast w przypadku reszty układu zastosowano zasilacz 5VDC / 1.55A.

- układ pomiarowy

Ma za zadanie odczytywać bieżącą wartość określonej zmiennej i w postaci przetworzonej tj. prądowej przesyłać do układu przetwarzania sygnałów analogowych.

Wejścia :

- stan czujnika wilgotności – 1 bit

- stan fotorezystora – 1 bit
- stan fotorezystora – 1 bit

Wyjścia :

- wartość pomiaru wilgotności gleby – postać prądowa
- wartość pomiaru fotorezystora – postać prądowa
- wartość pomiaru fotorezystora – postać prądowa

- układ wykonawczy

Ma za zadanie wykonywać operacje włączenia/wyłączenia lamp lub pompy.

Wejścia :

- stan lampy – 1 bit
- stan lampy – 1 bit
- stan pompy – 1 bit

- układ przetwarzania sygnałów analogowych

Układ zawiera 8 kanałów (w tym 3 wykorzystane w projekcie) które przyjmują sygnał w postaci prądowej i po przetworzeniu przesyłają go do układu sterującego który odczytuje odebrany sygnał jako 10 bitową wartość.

Wejścia :

- kanał 0 : wartość fotorezystora – postać prądowa
- kanał 1 : wartość fotorezystora – postać prądowa
- kanał 2 : wartość czujnika wilgotności gleby – postać prądowa

- serwer

Odpowiada za komunikację użytkownika z systemem, operuje na bazie danych oraz wysyła żądania akcji do układu sterującego.

- baza danych

Przechowuje dane zapisane przez użytkownika oraz udostępnia serwerowi.

- program nadzorujący potrzeby rośliny

Po przyjęciu ustawień rośliny od serwera, informuje go kiedy serwer musi wysłać żądanie o sprawdzeniu parametrów rośliny. Dzieje się to w podanym przez użytkownika interwałach.

2.1.3 Spis komponentów

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

2.1.4 Opis komponentów

Raspberry Pi 3 B+



Rys. 8. Zdjęcie poglądowe Raspberry Pi [1.7]

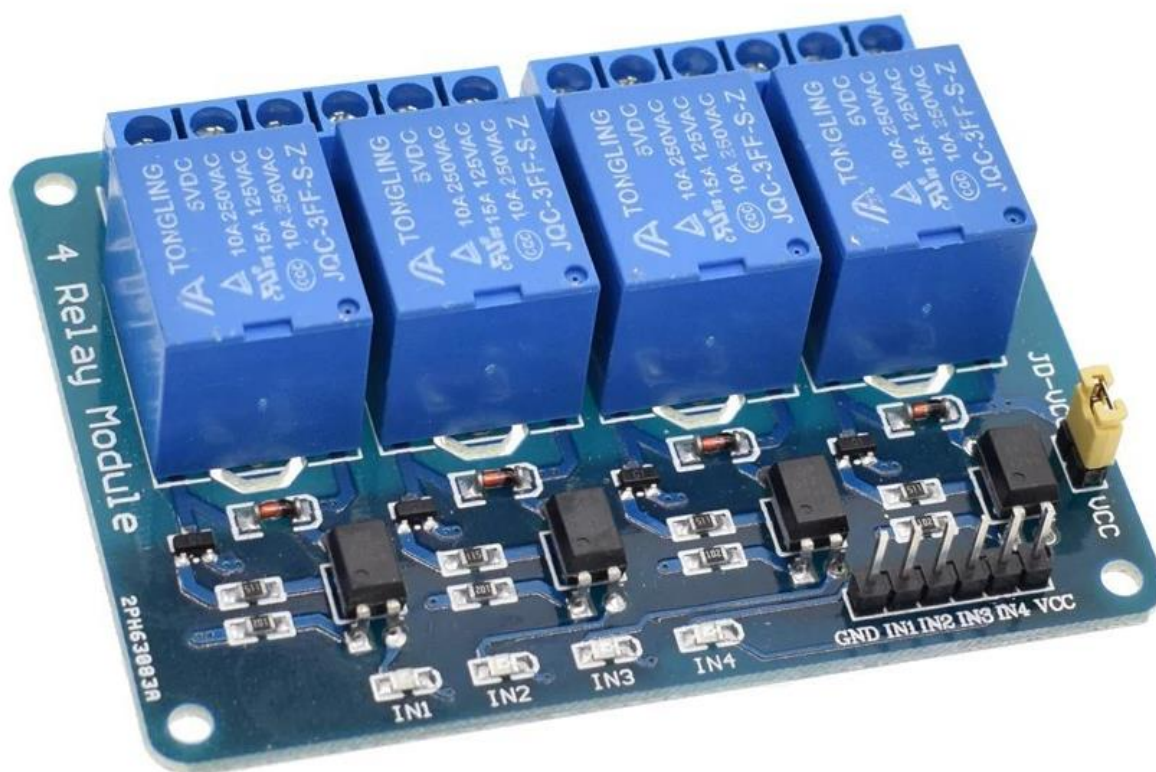
Jest jednopłytkowym mikrokomputerem, który zawiera większość cech standardowego komputera takich, jak złącza HDMI i USB, bezprzewodowe połączenie do sieci itd. System operacyjnym dedykowanym dla Raspberry jest Raspbian, który opiera się na Debianie (dystrybucja linuxa). Dwie

główne wyróżniające cechy Raspberry Pi to jedynie 5V wymaganego zasilania oraz GPIO (ang. general purpose input/output), które pozwalają na kontrole, oraz odczyt komponentów elektrycznych, co sprawia, że Raspberry Pi jest dobrym narzędziem to wszelkich projektów elektronicznych, które potrzebują dostępu do internetu, większej liczby obliczeń lub komunikacji z innymi urządzeniami. Dlatego też możemy znaleźć Raspberry PI w wielu projektach związanych z IoT oraz smart home.

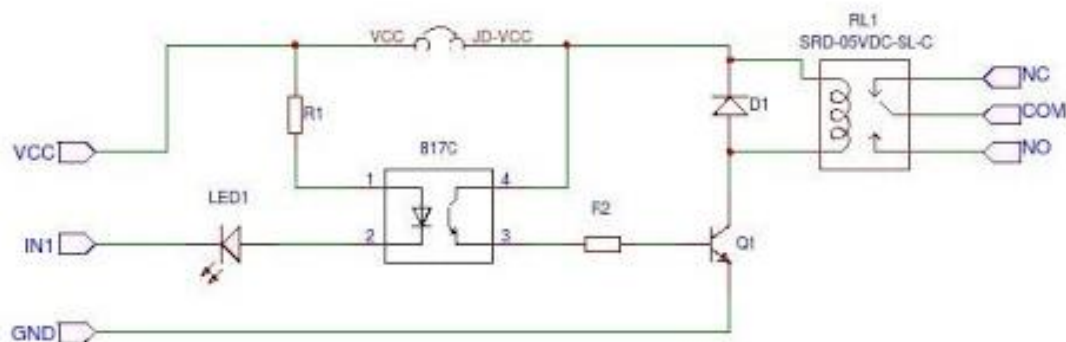
Szerzej na temat Raspberry PI można przeczytać :

- Oficjalna strona producenta : <https://www.raspberrypi.org/>
- Oficjalny magazyn : The MagPi [<https://magpi.raspberrypi.org/>]
- Książka : „Elektronika z wykorzystaniem Arduino i Raspberry Pi” – Simon Monk

Moduł przekaźników



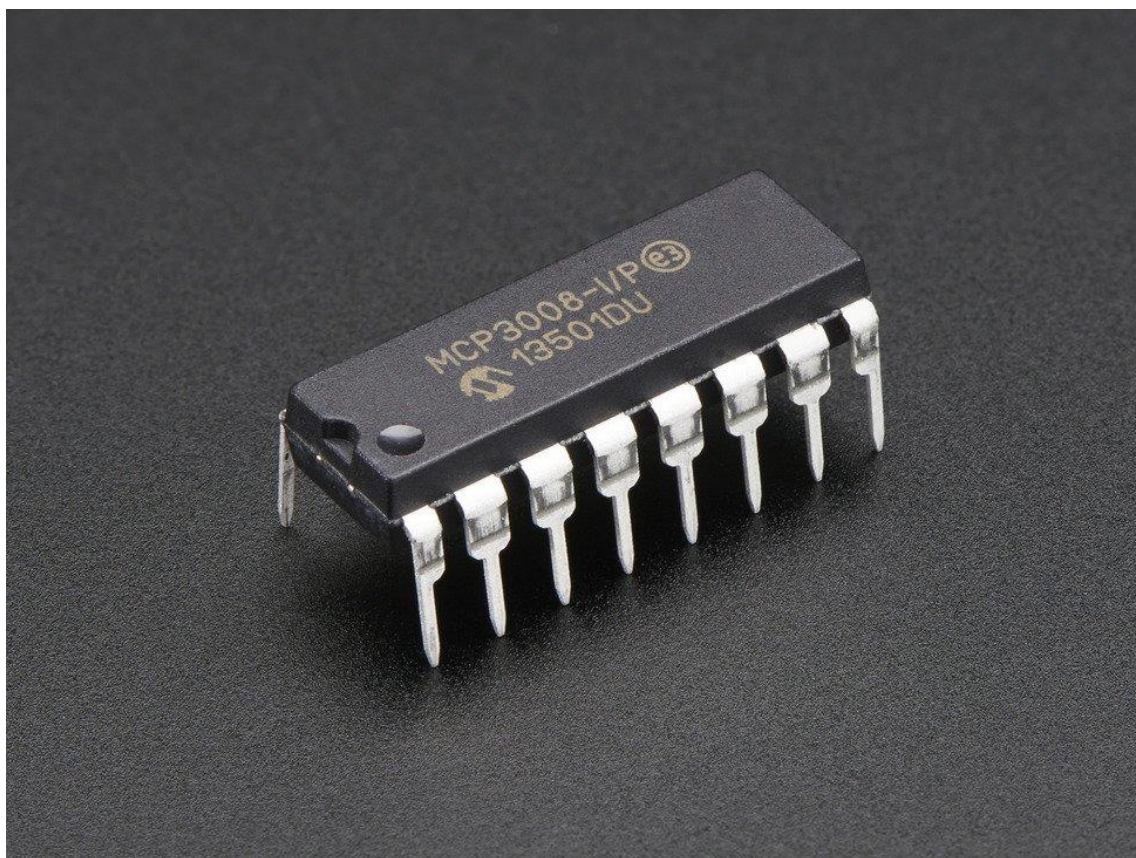
Rys. 9. Zdjęcie poglądowe modułu przekaźników [1.8]



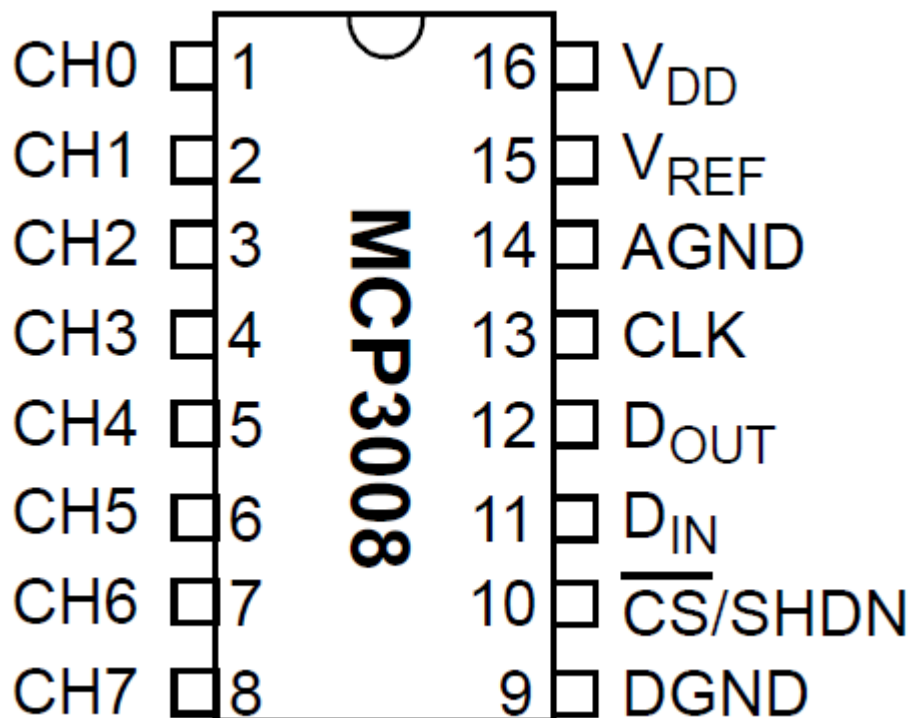
Rys. 10. Przykładowy obwód przekaźnika [1.9]

Przekaźnik to w zasadzie przełącznik używający elektromagnesu. Elektromagnes potrzebuje stałego lub zmiennego napięcia, w zależności od konstrukcji cewki, by powodować połączenie styków. Pozwala to na włączenie/wyłączenie urządzenia, które wymaga innego napięcia. Zapewnia to odseparowanie źródeł zasilania co chroni układ przed przebiegiem które może doprowadzić do uszkodzeń pozostałych komponentów które nie są przystosowane do takich wartości napięcia.

ADC (konwerter analogowo cyfrowy)

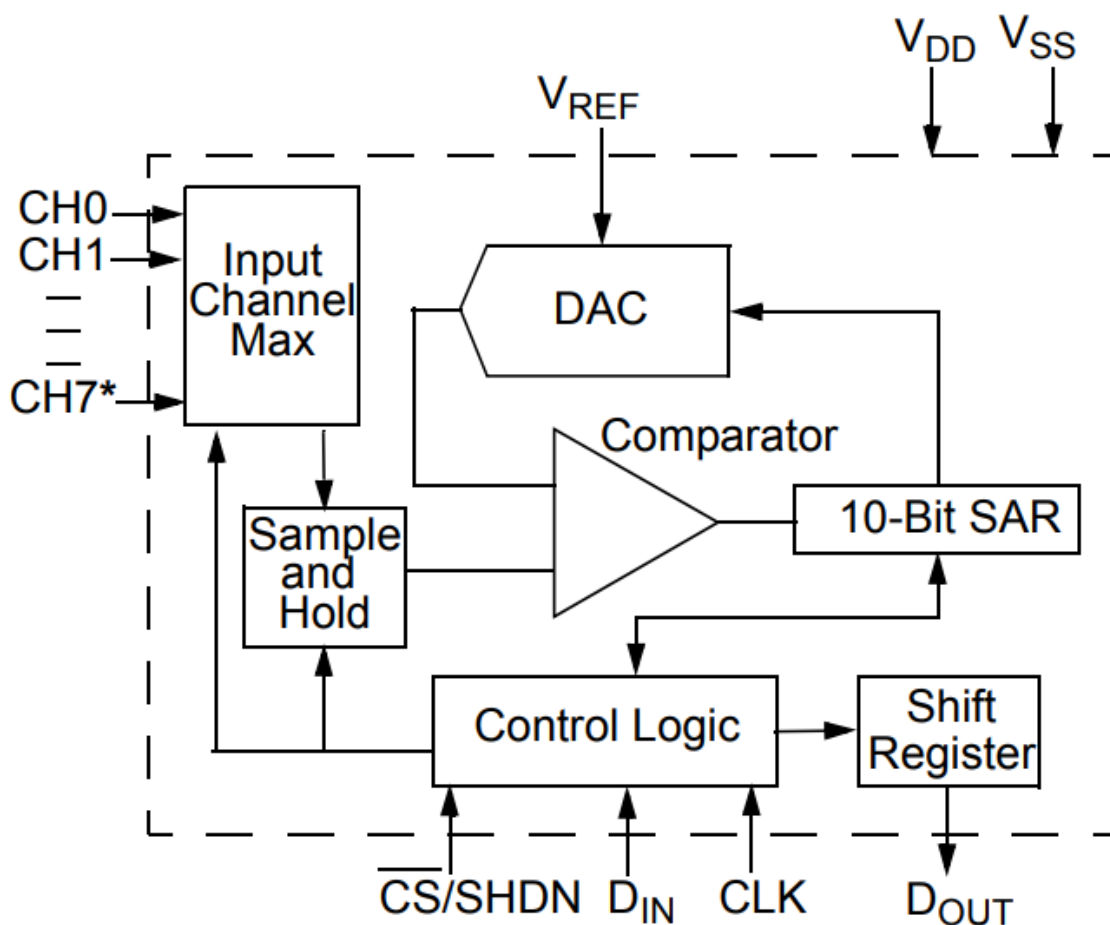


Rys. 11. Zdjęcie poglądowe MCP 3008 [1.10]



Rys. 12. Schemat MCP 3008 [1.10]

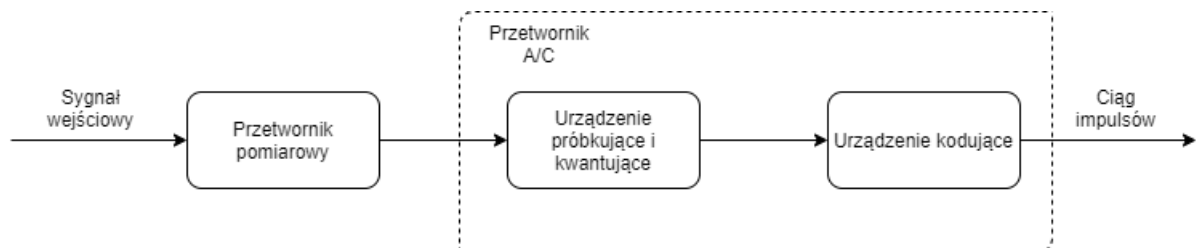
Jako że Raspberry Pi nie jest w stanie samo odczytać wartości analogowych potrzebuje pomocy w postaci konwertera ADC. Konwerter analogowo cyfrowy to urządzenia które zmienia ciągły sygnał analogowy na przybliżony sygnał cyfrowy. Odbywa się to przez próbkowanie sygnału analogowego ze stałą częstotliwością. Dokładność konwersji zależy głównie od 2 parametrów, czyli częstotliwości próbkowania oraz rozdzielczości konwersji. W projekcie wykorzystano moduł ADC MCP3008, w którym nominalny czas konwersji wynosi 10us i rozdzielczość 10bitów, co umożliwia próbkowanie z częstotliwością 100Hz i odczyt wartości w zakresie 0 – 1024.



Rys. 13. Diagram bloków funkcyjnych [1.12]

Zasada działania :

Sygnał analogowy zmienia się w czasie, żeby go przetworzyć pobierany jest fragment sygnału, który jest przechowywany przez czas konwersji (blok Sample and Hold na powyższym diagramie).



Rys. 14. Schemat blokowy ADC [wykonanie własne]

Kolejnym krokiem jest kwantyfikacja. Wartości sygnału w punkcie przypisywana jest najbliższa wartość jaką może przypisać konwerter (zależna od rozdzielczości konwersji). Punkty w których wartości są wyznaczane są zależne od sygnału zegarowego (mierzone na spadkach sygnału). Po przypisaniu wartość jest ona przetwarzana na wartość binarną i wysyłana do Raspberry PI.

Fotorezystor



Rys. 15. Zdjęcie poglądowe fotorezystora [1.13]

Czyli element półprzewodnikowy, w którym pod wpływem oświetlenia następuje zmiana jego przewodności, niezależnie od kierunku przyłożonego napięcia zewnętrznego. Oświetlenie fotorezystora powoduje zwiększenie przepływu prądu (zmniejsza się rezystancja), co pozwala na pomiar wartości nasłonecznienia, pomaga to określić czy roślina wymaga dodatkowego naświetlenia.

Użyty w systemie jest fotorezystor GL5537-1

Specyfikacja :

- Rezystancja w jasnym otoczeniu : 20-30k Ω
- Rezystancja w ciemności : 2M Ω
- Napięcie maksymalne (DC) : 150V
- Moc maksymalna : 100mW
- Rozmiar : 5x2x2mm
- Temperatura pracy : od -30°C do 70°C

Czujnik wilgotności gleby



Rys. 16. Zdjęcie poglądowe czujnika Iduino ME110 [1.13]

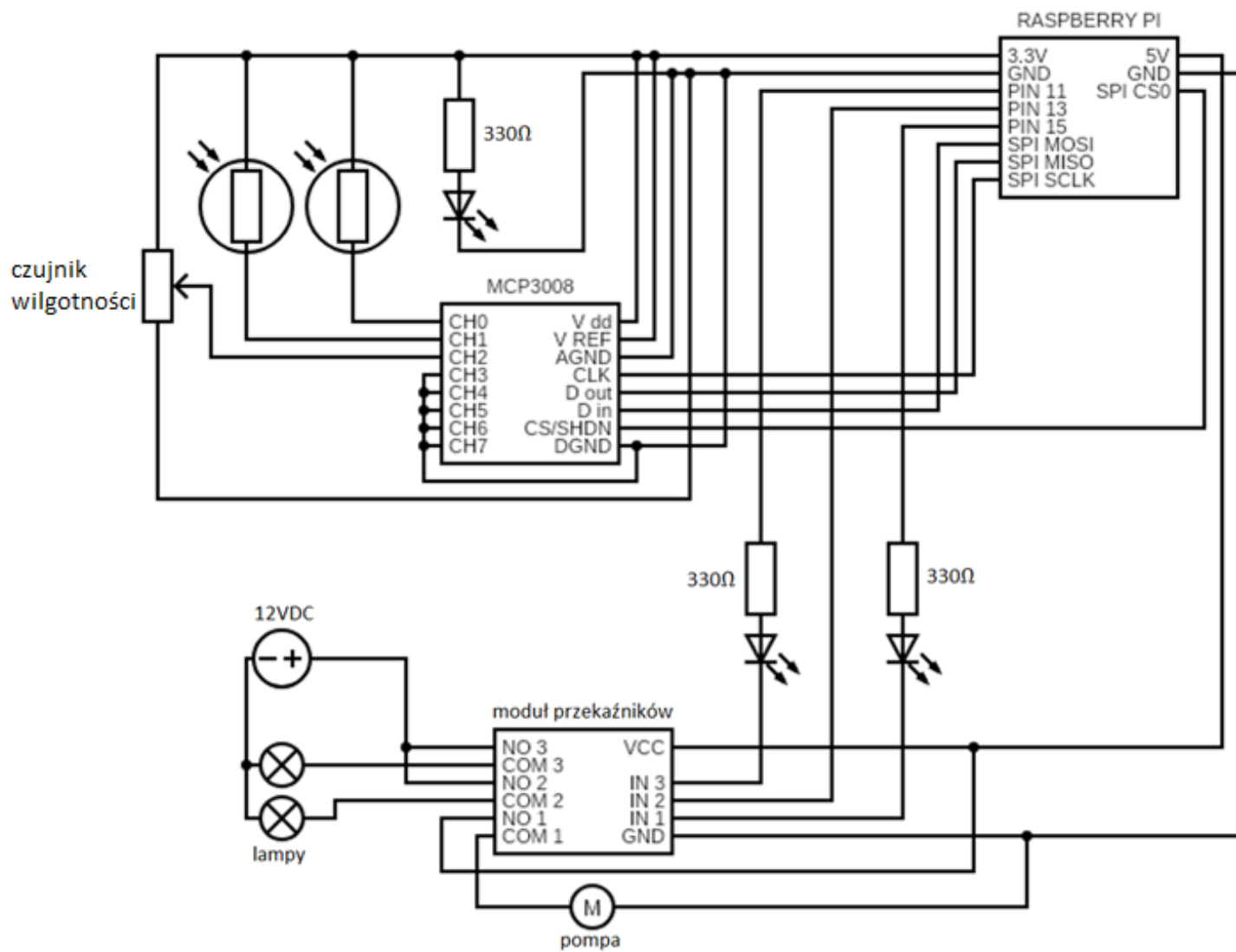
Czujnik wilgotności gleby składa się z dwóch sond, które umożliwiają przepływ prądu przez wilgotną glebę. Można go bardzo łatwo wykorzystać, po prostu wkładając czujnik do gleby i odczytując wyniki za pomocą ADC, działa to ponieważ im większa wilgoć tym mniejsza rezystancja gleby.

Użyty w systemie jest czujnik Iduino ME110

Specyfikacja :

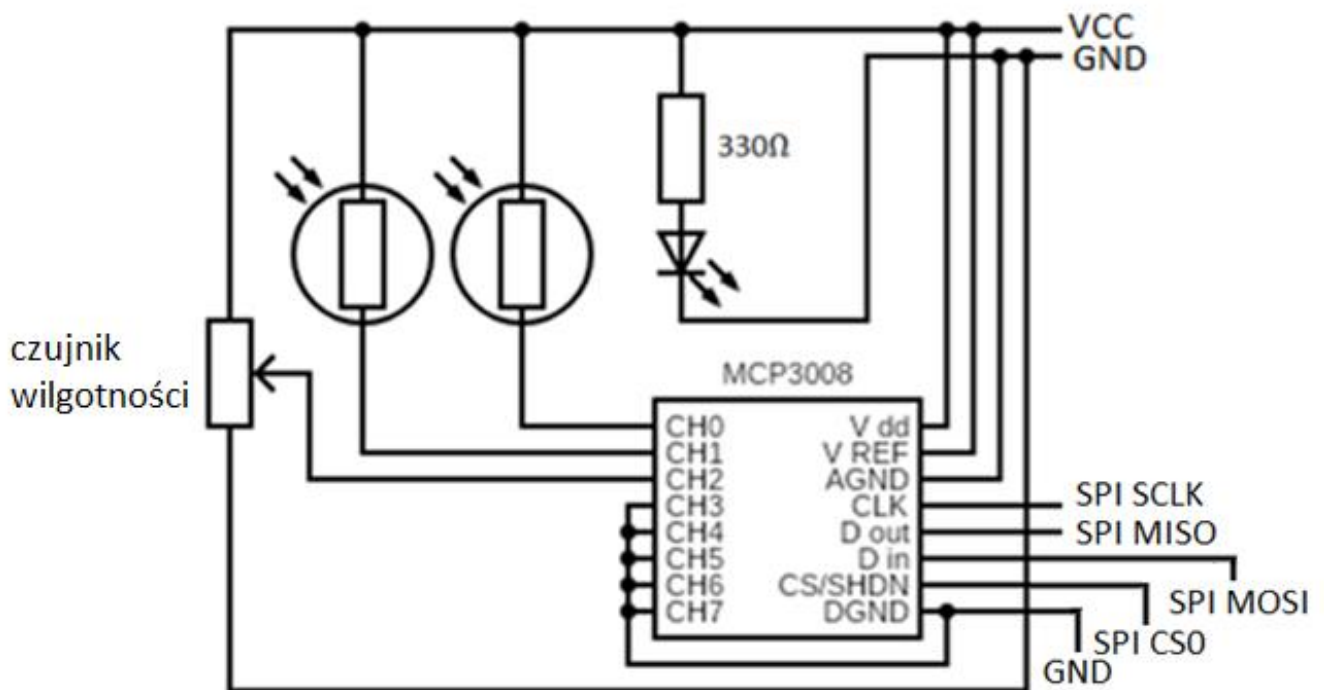
- Napięcie zasilania : od 3,3V do 5V
- Pobór prądu : 35mA
- Interface : analogowy
- Wymiary 76x20mm

2.1.5 Schemat układu



Rys. 17. Schemat układu [wykonanie własne]

Część układu z czujnikami



Rys. 18. Fragment schematu układu [wykonanie własne]

Opis wyjść MCP3008 :

- V_{DD} – zasilanie układu
- V_{REF} – zasilanie referencyjne, czyli maksymalna możliwa wartość, jaką może wystać układ
- A GND – uziemienie układu
- CLK – zegar synchronizujący przesył danych pomiędzy MCP 3008 a Raspberry Pi
- D out (serial data out) – z tego wyjścia można pobrać wartości z poszczególnych kanałów
- D in (serial data in)– na ten pin przesyłane są konfigurację kanałów
- CS/SHDN – to pin służący do inicjacji komunikacji z MCP3008, gdy na pinie tym jest 0, MCP3008 przechodzi w stan zmniejszonego poboru prądu.
- DGND – do tego pinu podłącza się nie używane kanały by nie dawały żadnych sygnałów, w innym wypadku kanały generują sygnał sinusoidalny.

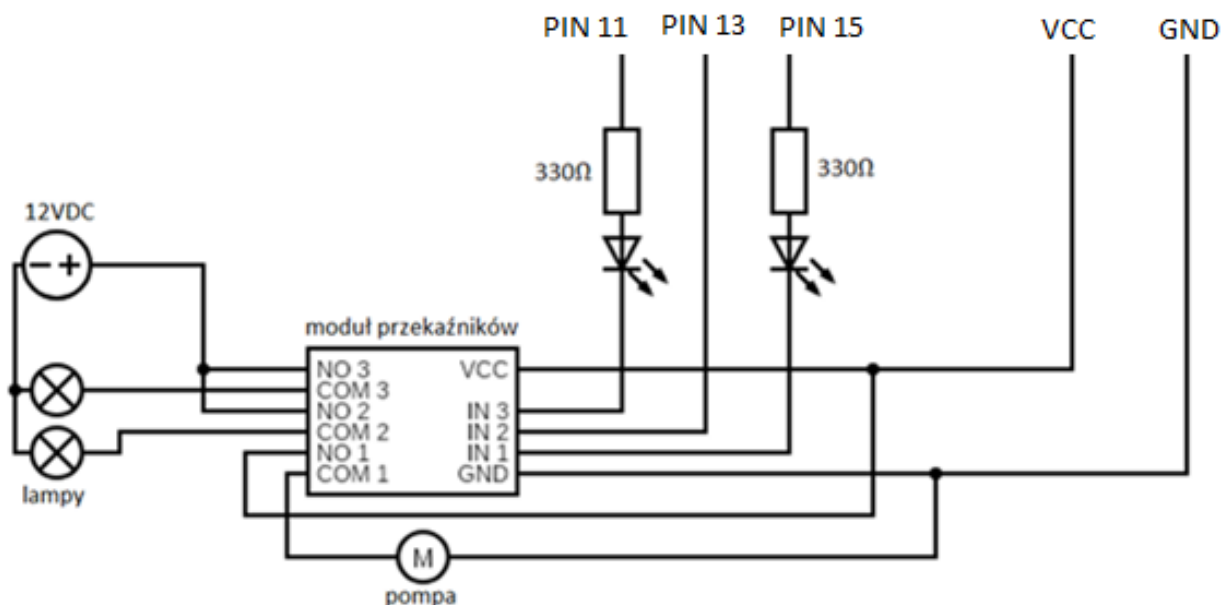
SPI

Raspberry Pi bez urządzeń zewnętrznych nie pozwala na odczyt danych analogowych, do tego potrzebujemy nie tylko konwertera ADC ale też połączenia przez konkretne piny przystosowane do komunikacji protokołem SPI.

Szeregowy interfejs urządzeń peryferyjnych (ang. 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.

Część układu z aktorami



Rys. 19. Fragment schematu układu [wykonanie własne]

Do rozdzielenia układu sterującego od aktorów, którzy wymagają większego zasilania użyto przekaźników. Przekaźniki pozwalają na użycie sygnału 3.3V do włączenia urządzeń używających 5V (pompa) i 12 V (lampy). PIN 11 i PIN 13 odpowiadają za włączenie lamp. PIN 15 jest odpowiedzialny za włączenie pompy. Sygnały stanu aktorów są wysyłane przez Raspberry PI. 1 – urządzenie włączone, 0 – wyłączone. Ponieważ każdy z aktorów jest podłączony do wejścia NO (normalny open).

2.2 Software

2.2.1 Konfiguracja serwera

Konfiguracja dostępu

Na początku pracy z Raspberry Pi należy zadbać by udostępniało one odpowiednie porty do połączenia się z nim, ale bez potrzeby podłączania peryferiów. Istnieje możliwość napisania całego projektu bezpośrednio na mikrokomputerze, ale jego rozmiary przekładają się na jego wydajność. Dla działania systemu wydajność urządzenia jest wystarczająca, jednak praca na nim może być niekomfortowa, gdy jesteśmy przyzwyczajeni do prędkości i funkcjonalności (np. kilka monitorów), które oferują standardowe stacje robocze.

Zaczynając od ustawienia statycznego adresu IP. Na wykonanie tego jest kilka możliwości, przy tym najlepiej zrobić to w pliku konfiguracyjnym na Raspberry Pi, ponieważ wtedy konfiguracja jest bezpośrednio na urządzeniu, co przekłada się na możliwość przeniesienia systemu do innej sieci przy niewielkim nakładzie pracy, w porównaniu z ustawieniem np. statycznego adresu IP w routerze. Plik znajduje się pod ścieżką „/etc/dhcp/conf”. Edytor pozwalający na edycje pliku można wywołać poleceniem :

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

Rys. 20. Fragment z terminalu linuxa [wykonanie własne]

W dolnej części pliku znajduje się konfiguracja interfejsu sieciowego 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
```

Rys. 21. Fragment z terminalu linuxa [wykonanie własne]

Wpisy pod „interface wlan0” to parametry interfejsu które można modyfikować. W miejsce `ip_address` należy wpisać preferowany statyczny adres IP. Warto pamiętać o tym, że adres musi znajdować się w tej samej podsieci co używana stacja robocza. By sprawdzić w jakiej podsieci znajduje się urządzenie należy użyć następującej komendy :

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

Rys. 22. Fragment z terminalu linuxa [wykonanie własne]

A następnie w zwróconej informacji wyszukać żądany interfejs.

```

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

```

Rys. 23. Fragment z terminalu linuxa [wykonanie własne]

Następnie należy włączyć serwer SSH. Standardowo Raspberry Pi jest wyposażone w taką funkcjonalność, ale jest ona nieaktywna. By włączyć serwer SSH należy wejść do programu konfiguracyjnego komendą :

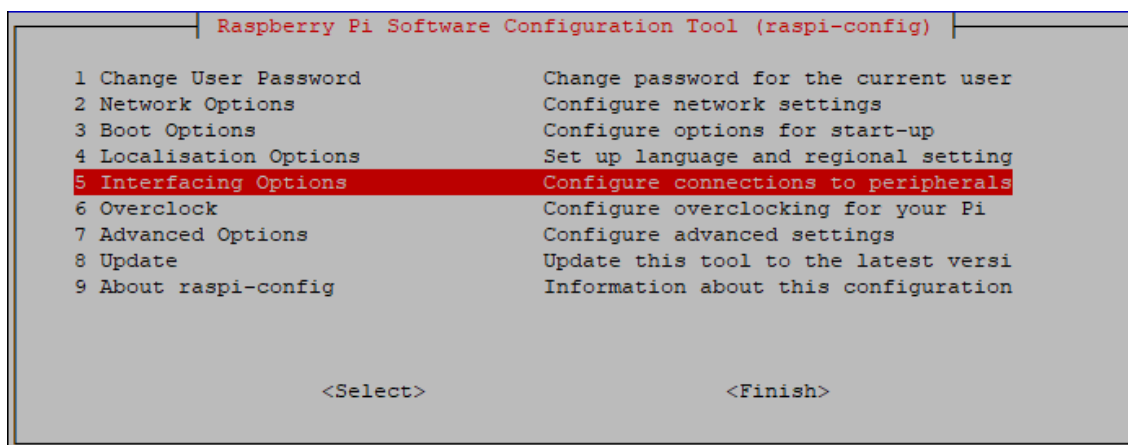
```

pi@raspberrypi:~ $ sudo raspi-config

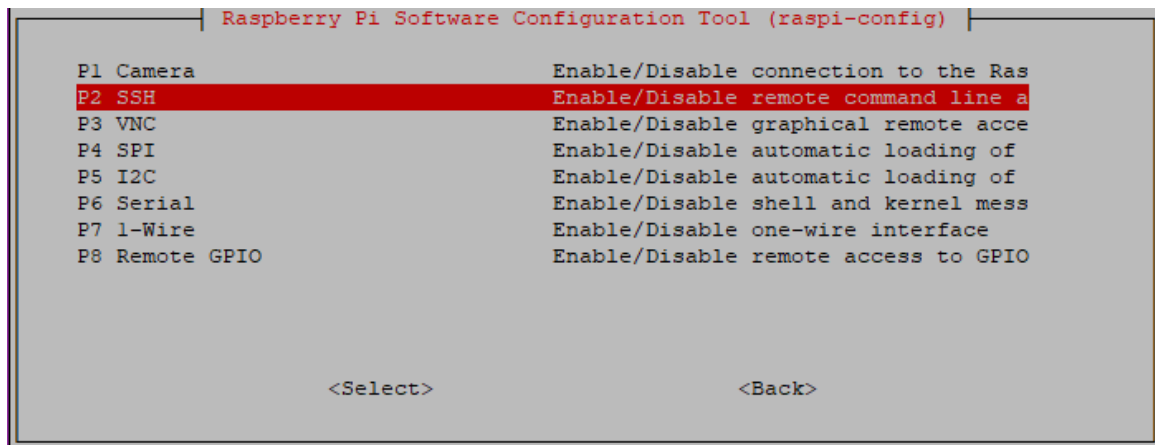
```

Rys. 24. Fragment z terminalu linuxa [wykonanie własne]

A następnie wybrać podkreślone na czerwono opcje :



Rys.25. Fragment z terminalu linuxa [wykonanie własne]



Rys. 26. Fragment z terminalu linuxa [wykonanie własne]

Kolejnym krokiem jest wyłączenie urządzenia, np. przez użycie komendy :

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

Rys. 27. Fragment z terminalu linuxa [wykonanie własne]

Teraz należy odłączyć peryferia od Raspberry Pi i po ponownym uruchomieniu jest możliwy dostęp do urządzenia z poziomu innej stacji roboczej w tej samej podsieci.

Włączenie aplikacji przy starcie systemu.

Żeby program włączał się od razu przy podłączeniu urządzenia do prądu (bez potrzeby logowania) należy użyć narzędzia zwanego Crontab.

Crontab to narzędzie programowe służące do planowania zadań w czasie w systemach Unix`owych. Pozwala na zaplanowanie wykonania skryptu, lub określonej operacji w wyznaczonym czasie, lub jakimś zdarzeniu (jak włączenie urządzenia).

By użyć Crontab należy wpisać komendę :

```
pi@raspberrypi:~ $ crontab -e
```

Rys. 28. Fragment z terminalu linuxa [wykonanie własne]

Co otworzy plik w którym można dodać ścieżkę do programu z oznaczeniem @reboot co sprawi że będzie on uruchamiany przy każdym starcie systemu.

```

GNU nano 2.7.4      Plik: /tmp/crontab.n0XQYT/crontab

# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
@reboot /home/pi/Farma/main.py

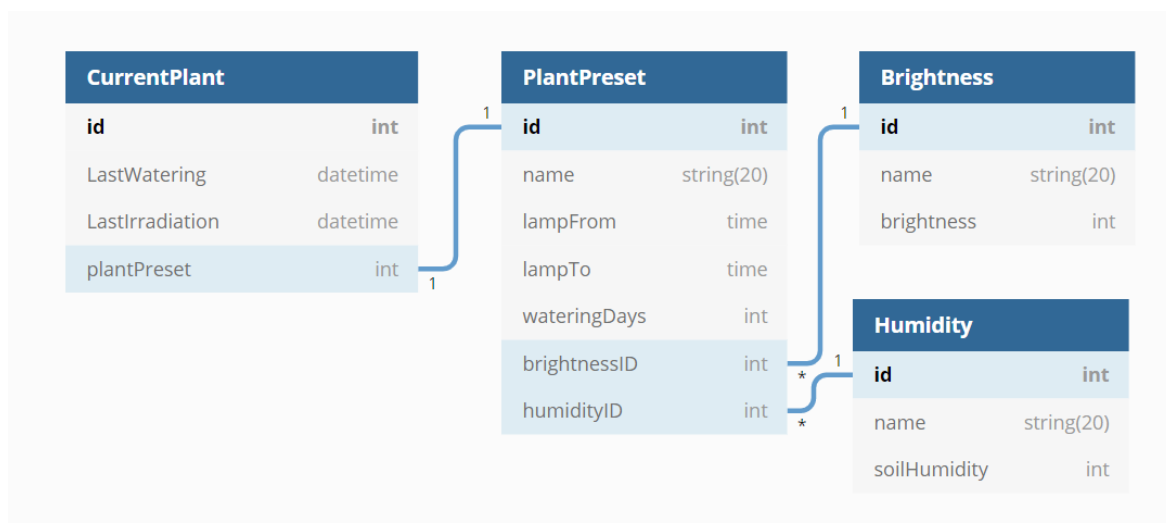
```

Rys. 29. Fragment z terminalu linuxa [wykonanie własne]

2.2.2 Baza danych

Baza danych pozwala na zarządzanie i gromadzenie danych. W projekcie użyto SQLite jako systemu zarządzania bazami danych (ang. Database management system) oraz SQLAlchemy jako ORM (ang. Object-relational mapping). Oba wspomniane narzędzia opisane są w wstępie pracy.

Schemat bazy danych



Rys. 30. Schemat bazy danych [wykonanie 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 nasłonecznienia w formacie zrozumiałym dla systemu oraz przyjazną dla użytkownika nazwę (np. „niewielkie nasłonecznienie”).

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

Dodanie bazy danych

Code first approach

W opisywanym podejściu zaczyna się od utworzenia modeli(klas) obiektów z których korzystać będzie aplikacja, następnie na ich podstawie konstruowana jest (zazwyczaj automatycznie) baza danych.

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)
```

Rys. 31. Fragment pliku „models.py” [wykonanie własne]

Każdy model widziany przez SQLAlchemy musi dziedziczyć po klasie „Model” oraz zawierać kolumny opisane w następującym formacie :

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

Rys. 32. Przykład dodania kolumny w SQLAlchemy [wykonanie własne]

Wspomnianymi wyżej dodatkowymi parametrami jest np. auto-inkrementacja, indeksowanie itp. Wartość „__tablename__” nie jest konieczna, ale pozwala na nadanie dowolnej nazwy dla tabeli niezależnie od nazwy klasy modelu, domyślną nazwą dla PlantPreset byłaby nazwa plant_preset. Kolejnym krokiem jest użycie komendy, które na podstawie modeli generuje bazę danych:

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

Rys. 33. Fragment pliku „baseDBSetup.py” [wykonanie własne]

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)
```

Rys. 34. Przykładowy kod łączący SQLite, Flask i SQLAlchemy [wykonanie własne]

W tym przykładzie można zobaczyć, dlaczego język Python cieszy się tak dużą popularnością (trzecie miejsce w rankingu Tiobe w czasie pisania pracy), zwłaszcza przy tworzeniu małych aplikacji. Kilka linii kodu wystarczy do połączenia bazy danych z aplikacją. Kodu co prawda jest niewiele, ale są tutaj integrowane aż trzy komponenty:

- SQLite
- SQLAlchemy
- Flask

CRUD (ang. 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()
```

Rys. 35. Fragment pliku „baseDBSetup.py” [wykonanie własne]

- Odczyt z bazy (Read)

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

Rys. 36. Fragment pliku „main.py” [wykonanie własne]

- Edytowanie danych w bazie (Update)

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

Rys. 37. Fragment pliku „main.py” [wykonanie własne]

- Usuwanie z bazy (Delete)

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

Rys. 38. Przykładowy kod usuwający wpis w bazie danych [wykonanie własne]

2.2.3 API

Interfejs programowania aplikacji (ang. Application Programming Interface) – jest to zbiór reguł takich jak funkcje i struktury danych, które pozwalają na komunikację pomiędzy aplikacjami.

W projekcie użyto konkretnego rodzaju API, tak zwanego Web API, które pozwala na komunikację przez protokół http. Z perspektywy użytkownika pozwala to między innymi na komunikację z systemem przez przeglądarkę internetową, choć komunikację może rozpocząć też inna aplikacja wysyłająca żądanie HTTP.

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)
```

Rys. 39. Przykładowy kod bazowej aplikacji wykorzystującej Flask [wykonanie własne]

Na przykładzie powyżej można zobaczyć jak wygląda podstawowe API w frameworku Flask, linii 6-7 dodaje bazowy endpoint (nasz adres bez żadnej podstrony) który zwróci „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 podstawowym porcie http
- Debug = True – mówi o tym, 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'])
```

Rys. 40. Fragment pliku „main.py” [wykonanie własne]

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 przypisane kilka endpointów w zależności od użytej metody, warto pamiętać że domyślnie przeglądarka internetowa wysyła żądzenie GET.

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

Rys. 41. Fragment pliku „main.py” [wykonanie własne]

Flask pozwala 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, ze względu na łatwość odczytu oraz możliwość szyfrowania.

Dla standardowego żądania po pozytywnym rozpatrzeniu wysyłany jest kod statusu 200 (kod sukcesu) oraz w ciele odpowiedzi żądane przez użytkownika dane, które w architekturze REST są zazwyczaj w formacie JSON, choć w przypadku opisywanego systemu w większości przypadków będzie to strona internetowa.

Nie dla każdego sukcesywnie rozpatrzonego żądania zwracany jest kod 200, są odstępstwa od tej reguły. W Przypadku chęci usunięcia czegoś z bazy (metoda DELETE) standardową praktyką jest zwrócenie kodu 204 (brak zawartości), lub w przypadku przekierowania na inną stronę zwraca się kod 302.

Przekierowanie do innego endpointa wygląda w następujący sposób :

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

Rys. 42. Fragment pliku „main.py” [wykonanie własne]

2.2.4 Widoki

Widok – generowana strona internetowa która pozwala użytkownikowi na interakcje z systemem.

Widok taki jest generowany na podstawie adresu pod który zostało wysłane żądanie (tak zwany endpoint), danych wysłanych razem z żądaniem, oraz danymi w bazie danych. Narzędziem do obsługi żądania, jak i generowania widoku jest Flask (opisany w wstępie pracy).

Generowanie strony

Gdy endpoint kończy się w ten sposób

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

Rys 43. Fragment pliku „main.py” [wykonanie własne]

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 zaczyna się od :

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

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

{%block body%}
<div id="main">
  <div>
    <h2>pump : {{pumpStatus}}</h2>
    <div>
      <a href="/pump/on" class="btn btn-info" role="button">on</a>
      <a href="/pump/off" class="btn btn-info" role="button">off</a>
    </div>
    <div><a>last pump activity :  {{lastWatering}}</a></div>
    <div><a>current humidity level :  {{currentHumidity}}</a> </div>
  </div>
</div>
```

Rys. 44. Przykład użycia rozrzedzenia [wykonanie własne]

który używa funkcjonalności rozrzedzenia pliku (w tym wypadku „layout.html”), co w praktyce dołącza do strony część wspólną dla wszystkich podstron, następnie piszemy standardowy kod HTML. Możemy też doładować w dowolnym miejscu komponent w następujący sposób :

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

```

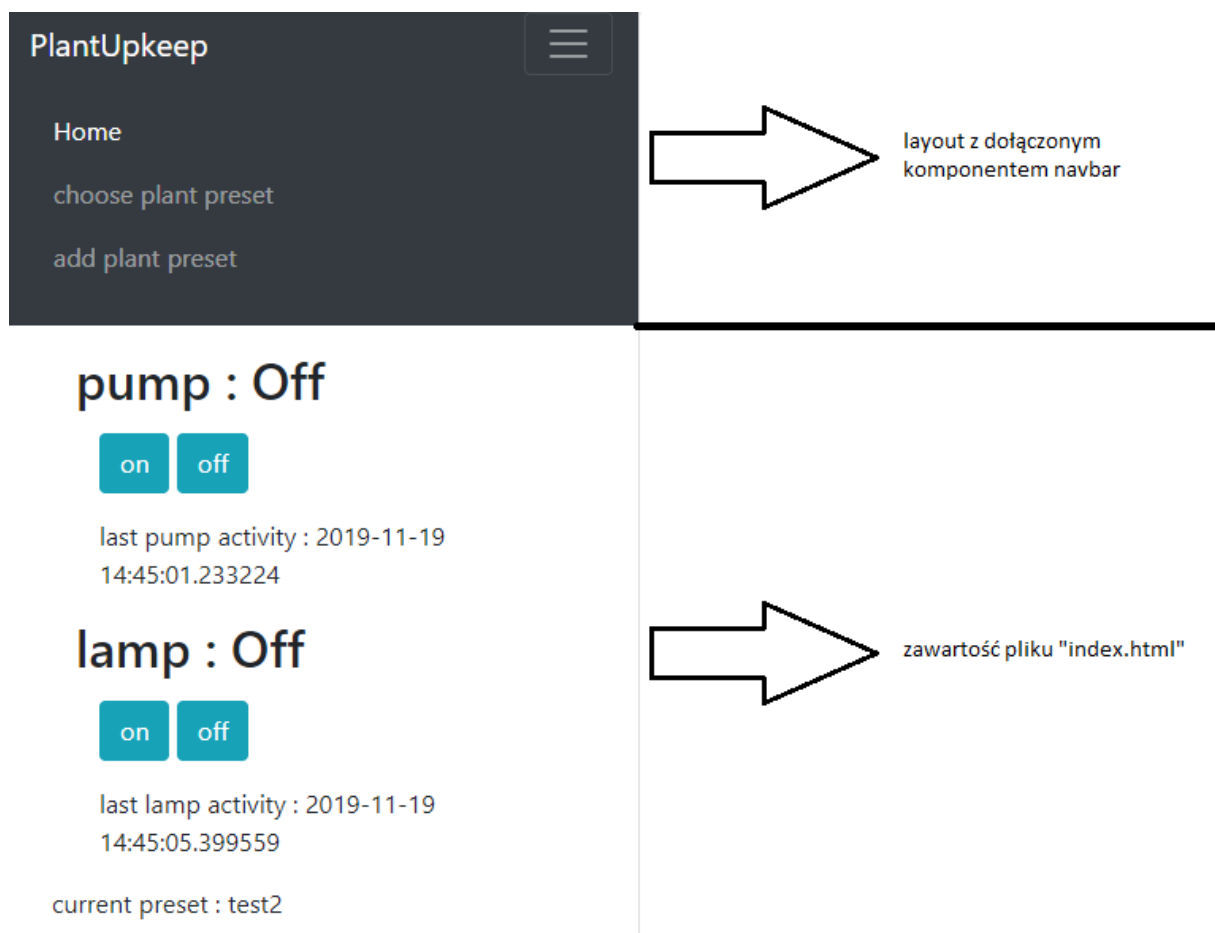
</head>
<body>
  {% include 'includes/navBar.html' %}
  <div>
    {%block body%}{%endblock%}
  </div>

```

Rys. 45. Przykład dodania komponentu [wykonanie własne]

co załaduje cały HTML z danego pliku. W tym przypadku layout zawierający metadane doładowuje navbar.html (menu strony) ponieważ jest to jedna z rzeczy która będzie znajdować się na każdej podstronie.

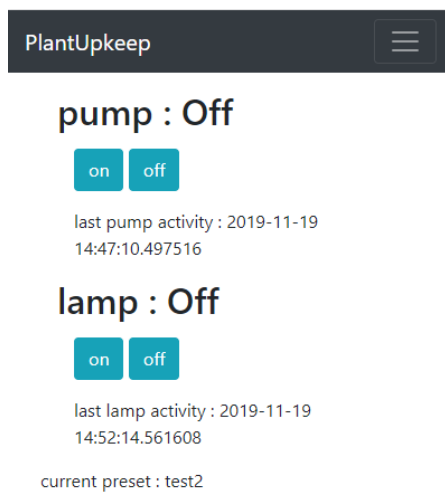
Wynik :



Rys. 46. Wygenerowany widok „index.html” [wykonanie własne]

Widoki w systemie

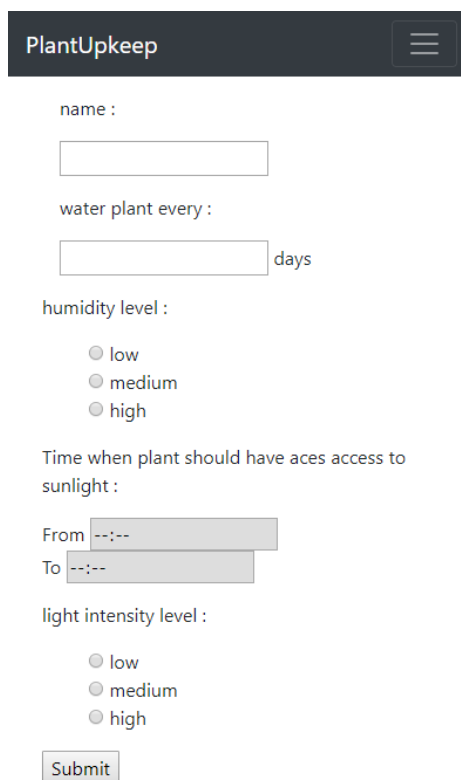
System zawiera 3 widoki



The screenshot shows the 'PlantUpkeep' application header. Below it, there are two sections: 'pump : Off' and 'lamp : Off'. Each section has 'on' and 'off' buttons. Under the pump section, it says 'last pump activity : 2019-11-19 14:47:10.497516'. Under the lamp section, it says 'last lamp activity : 2019-11-19 14:52:14.561608' and 'current preset : test2'.

Home – widok w którym można zobaczyć dane o aktualnym stanie aktorów, oraz manualnie włączyć lub wyłączyć poszczególne urządzenie. Ponadto możemy zobaczyć czas ostatniego ich użycia oraz aktualnie włączone ustawienie.

Rys. 47. Wygenerowany widok „index.html”
[wykonanie własne]



The screenshot shows the 'PlantUpkeep' application header. Below it is a form titled 'Add new preset'. The form contains the following fields and controls: a 'name' text input; a 'water plant every' text input followed by 'days'; a 'humidity level' section with three radio buttons labeled 'low', 'medium', and 'high'; a 'Time when plant should have access to sunlight' section with 'From' and 'To' time pickers; a 'light intensity level' section with three radio buttons labeled 'low', 'medium', and 'high'; and a 'Submit' button at the bottom.

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

Rys. 48. Wygenerowany widok
„addPreset.html” [wykonanie własne]

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 wszystkie wcześniej utworzone ustawienia i pozwala na wybranie aktywnego.

Rys. 49. Wygenerowany widok „chosePreset.html” [wykonanie własne]

2.2.5 Algorytmy użycia aktorów

Samo korzystanie z aktorów jest dość proste: z bazy danych pobierana jest docelową wartość parametru (nawodnienie lub nasłonecznienie) i jeśli sensor zwróci wartość poniżej wartości docelowej przypisany aktor jest włączany. Podstawową trudność polega na wyznaczaniu przez program nadzorujący momentu wywołania wyżej wymienionej operacji. Wygląda to następująco :

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')
```

Rys. 50. Fragment pliku „loop.py” [wykonanie własne]

Powyżej widać 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.

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')
```

Rys. 51. Fragment pliku „loop.py” [wykonanie własne]

W tym kodzie można zobaczyć podobne funkcje, ustawiającą (setupPump) oraz trzy zdarzenia (firstWatering, shedulePump oraz cancelShedulePump). Tutaj funkcja ustawiająca ma za zadanie ustawienie tylko pierwszego zdarzenia (firstWatering) ponieważ nawodnienie ustawiane jest na x („x” pobrane z bazy danych) dni od poprzedniego nawodnienia a nie na x dni od aktualnej daty. Dopiero zdarzenie pierwszego nawodnienia ustawia regularne podlewanie rośliny co x dni (w zależności od ustawień rośliny). Sprawdzanie czujnika wilgotności 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.

2.2.6 Wykonywanie funkcji w zaplanowanym czasie

Do wykonywania zadań w danym okresie czasu wykorzystana została biblioteka „schedule” stworzona przez Daniela Badera. Biblioteka ta używa wzorca budowniczego do planowania zadań w czasie, co pozwala zapisywać do planera wszelkie funkcje, bądź inne wywoływalne obiekty (callable) by wykonywały się w wybranym przez użytkownika czasie.

Wzorzec „Budowniczy” (ang. 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 większą kontrolę i przejrzystość obiektu.

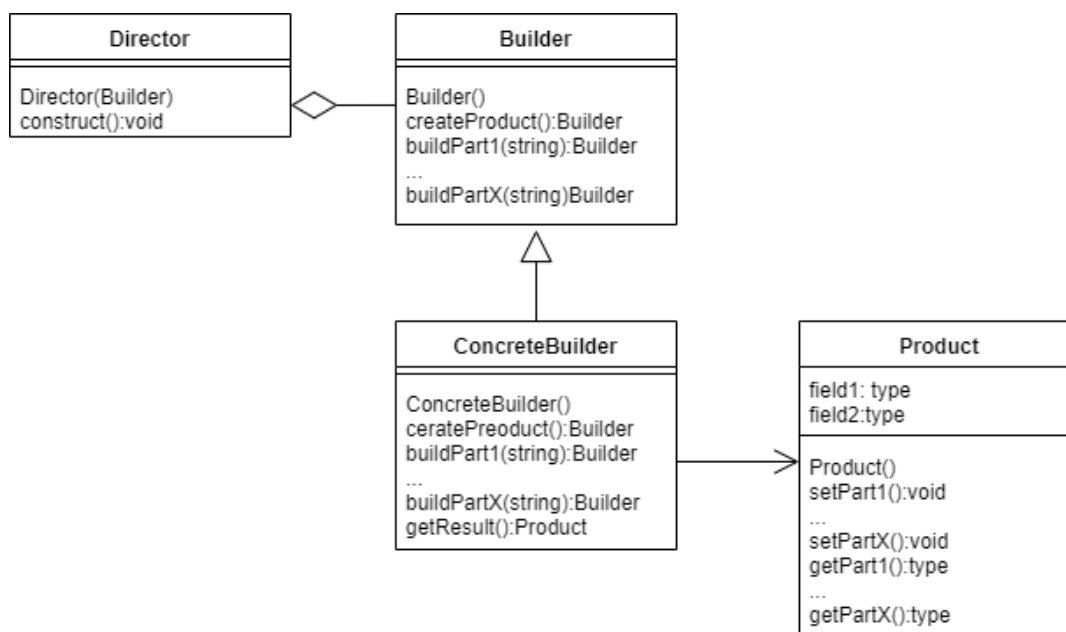
Zalety :

- Daje 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 :



Rys. 52. Diagram UML [wykonanie własne]

By aktywować zadania zapisane w planerze musimy wykonać kod :

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

Rys. 53. Przykład kodu [wykonanie własne]

co blokowało by serwer przed odpowiedziami na żądania, więc wystąpiła potrzeba ustawienia pętli nadzorującej zadania jako osobny wątek. Do tego użyty został moduł „threading”, który udostępnia proste funkcje korzystające z bardziej zaawansowanego modułu „thread”. Moduł „threading” został użyty nie tylko przez jego prostotę, ale też ze względu na zmniejszenie ilości kodu, łatwiejsze utrzymanie kodu, oraz mniejsze prawdopodobieństwo wystąpienia błędów (funkcje są bardziej ograniczone, przez co jest mniejsza szansa na popełnienie błędu).

Dla wygody została stworzona globalna flaga, 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)
```

Rys. 54. Fragment pliku „loop.py” [wykonanie własne]

Do wyłączenia serwera wykorzystać można sygnał SIGINT (ctrl+c) więc trzeba też zadbać o to by sygnał wyłączał cały serwer w tym też pętli nadzorującą. Do tego użyto 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)
```

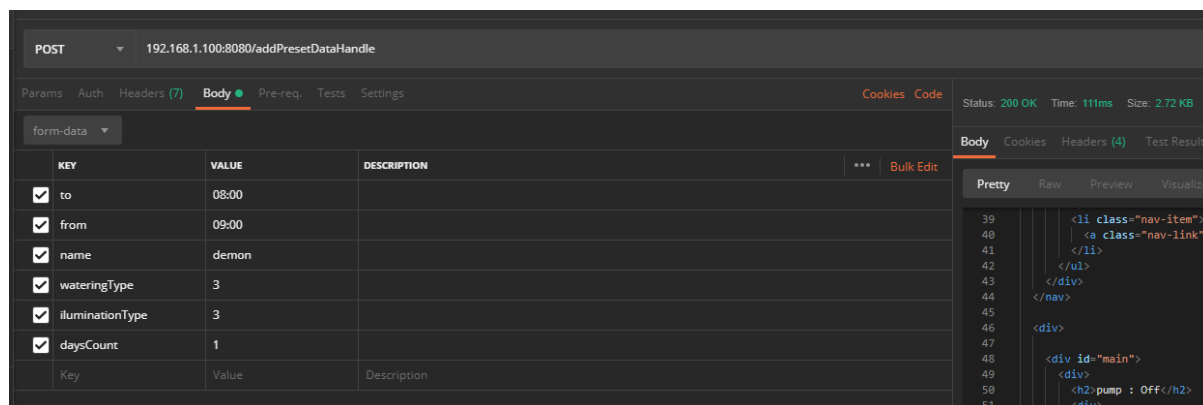
Rys. 55. Fragment pliku „main.py” [wykonanie własne]

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

2.2.7 Komunikacja z systemem bez użycia strony internetowej

Przykład

Z systemem można komunikować się też przy pomocy samych żądań HTTP pozwala to na stworzenie innych programów zarządzających systemem bez potrzeby znajomości wewnętrznych mechanizmów.



Rys. 56. Fragment programu Postman pokazujący wysłane żądanie [wykonanie własne]

Jako przykład użyje żądania wysłanego z programu Postman który pozwala przetestować poszczególne endpointy.

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

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 oświetlenia 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

2.2.8 Testy

Testy API

Do stworzenia automatycznych testów API w Postman`ie tworzy się tak zwane 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 wiadomo, 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 na zapewnienie że API spełnia wymagania funkcjonalne jak i nie funkcjonalne.

3 Podumowanie

Aktualnie kod nadzorujący jest integralną częścią serwera. W dalszym rozwoju systemu planowane jest 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 liczby roślin przez jeden serwer, aktualnie serwer może obsłużyć do 5 roślin ze względu na ograniczoną liczbę pinów w Raspberry PI.

Kolejnym planem jest dodanie możliwości kontroli większej liczby parametrów takich jak : wilgotność powietrza, skład powietrza. Ale wymaga to hermetycznego środowiska oraz więcej danych o określonej roślinie.

Ważne jest by przedłużyć żywotność czujnika wilgotności 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. Kolejnym pomysłem rozwiązania problemu krótkiej żywotności czujnika wilgotności jest automatyczne wyciąganie go z gleby gdy system nie wykonuje pomiaru, co jest opcją nieco bardziej skomplikowaną, ale też potencjalnie przedłuży życie sensora o wiele bardziej.

W przyszłości system będzie korzystać z połączenia HTTPS i odpowiednio zabezpieczonej strony internetowej oraz bazy danych.

System będzie informować użytkownika o problemach takich jak brak wody w zbiorniku, za niska wilgotność powietrza, zła temperatura, niezależnie od tego czy sam system jest w stanie wpłynąć na te parametry. Z większą liczbą funkcjonalności wiąże się większa liczba czujników, co znaczy że zmieni się też interface użytkownika, tak by użytkownik mógł pobrać interesujące go informacje.

4 Literatura

1. Strony www

- 1.1 Oficjalna strona SQLAlchemy - <https://www.sqlalchemy.org/>
- 1.2 Strona z otwartoźrudłowym projektami - <https://www.hackster.io/>
- 1.3 Strona z artykułami o tematyce hardware - <https://hackaday.com/>
- 1.4 Oficjalna strona produktu AeroGarden - <https://www.aerogarden.com/harvest-base.html>
- 1.5 Oficjalna strona produktu SmallGarden - <https://www.edntech.com/pages/smallgarden>
- 1.6 Oficjalna strona produktu SmartGarden - <https://sklep.smart-garden.pl/>
- 1.7 Oficjalna strona Raspberry PI - <https://www.raspberrypi.org/>
- 1.8 Sklep z częściami elektronicznymi - <https://arduino-pl.com/>
- 1.9 Sklep z częściami elektronicznymi - <https://www.generationrobots.com/>
- 1.10 Poradniki i projekty firmy AdaFruit - <https://learn.adafruit.com/>
- 1.11 Strona do nauki SQLite - <https://www.sqlitetutorial.net>
- 1.12 Dokumentacja MCP 3008 - https://botland.com.pl/index.php?controller=attachment&id_attachment=673
- 1.13 Sklep z częściami elektronicznymi - <https://botland.com.pl/>
- 1.16 Strona do nauki SQLite - <https://www.sqlitetutorial.net>
- 1.17 Dokumentacja do użytej w projekcie biblioteki shedule - <https://schedule.readthedocs.io/en/stable/>
- 1.18 Strona z opisem użycia SPI na Raspberry PI - <https://raspberrypi-aa.github.io/session3/spi.html>
- 1.19 Opisy wzorców projektowych - <https://refactoring.guru/design-patterns/>
- 1.20 Ogólny opis wymagań roślin doniczkowych - <https://muratordom.pl/ogrod/rosliny/uprawa-roslin-doniczkowych-w-domu-ogolne-zasady-aa-vRbS-Wrc1-4pEP.html>
- 1.21 Opis hodowania bazylii w warunkach domowych - <http://www.przesadzilam.pl/moj-balkon/uprawa-bazylii-w-doniczce/>

1.22 Baza wiedzy o bazach danych - <https://www.oracle.com/pl/database/>

2. Książki

2.1 Jon Duckett – „HTML i CSS. Zaprojektuj i zbuduj witrynę WWW. Podręcznik Front-End Developera”, Helion, 2018

2.2 Mark Lutz - „Learning Python 5ed”, O’Railly ,2013

2.3 Warren Gay – „Mastering the Raspberry Pi”, Apress, 2013

5 Spis ilustracji

Rys. 1. Zdjęcie poglądowe urządzenia Automated Indoor Gardener	8
Rys. 2. Zdjęcie poglądowe urządzenia FarmBot	9
Rys. 3. Zdjęcie poglądowe urządzenia Aerogarden Harvester	11
Rys. 4. Zdjęcie poglądowe urządzenia Smallgarden 2	12
Rys. 5. Zdjęcie poglądowe urządzenia Smart Garden 9	13
Rys. 6. Schemat funkcji	19
Rys. 7. Schemat modułów funkcjonalnych	21
Rys. 8. Zdjęcie poglądowe Raspberry Pi.....	23
Rys. 9. Zdjęcie poglądowe modułu przekaźników	24
Rys. 10. Przykładowy obwód przekaźnika	25
Rys. 11. Zdjęcie poglądowe MCP 3008	25
Rys. 12. Schemat MCP 3008	26
Rys. 13. Diagram bloków funkcyjnych	27
Rys. 14. Schemat blokowy ADC	27
Rys. 15. Zdjęcie poglądowe fotorezystora	28
Rys. 16. Zdjęcie poglądowe czujnika Iduino ME110	29
Rys. 17. Schemat układu	30
Rys. 18. Fragment schematu układu	31
Rys. 19. Fragment schematu układu	32
Rys. 20. Fragment z terminalu linuxa	33
Rys. 21. Fragment z terminalu linuxa	33
Rys. 22. Fragment z terminalu linuxa	33
Rys. 23. Fragment z terminalu linuxa	34
Rys. 24. Fragment z terminalu linuxa	34
Rys.25. Fragment z terminalu linuxa	34
Rys. 26. Fragment z terminalu linuxa	35
Rys. 27. Fragment z terminalu linuxa	35
Rys. 28. Fragment z terminalu linuxa	35
Rys. 29. Fragment z terminalu linuxa	36
Rys. 30. Schemat bazy danych	36
Rys. 31. Fragment pliku „models.py”	37
Rys. 32. Przykład dodania kolumny w SQLAlchemy	37
Rys. 33. Fragment pliku „baseDBSetup.py”	37
Rys. 34. Przykładowy kod łączący SQLite, Flask i SQLALCHEMY	38
Rys. 35. Fragment pliku „baseDBSetup.py”	38
Rys. 36. Fragment pliku „main.py”	38
Rys. 37. Fragment pliku „main.py”	39
Rys. 38. Przykładowy kod usuwający wpis w bazie danych	39
Rys. 39. Przykładowy kod bazowej aplikacji wykorzystującej Flask	39
Rys. 40. Fragment pliku „main.py”	40
Rys. 41. Fragment pliku „main.py”	40
Rys. 42. Fragment pliku „main.py”	40
Rys. 43. Fragment pliku „main.py”	41
Rys. 44. Przykład użycia rozrzedzenia	41
Rys. 45. Przykład dodania komponentu	42
Rys. 46. Wygenerowany widok „index.html”	42

Rys. 47. Wygenerowany widok „index.html”	43
Rys. 48. Wygenerowany widok „addPreset.html”	43
Rys. 49. Wygenerowany widok „chosePreset.html”	44
Rys. 50. Fragment pliku „loop.py”	44
Rys. 51. Fragment pliku „loop.py”	45
Rys. 52. Diagram UML	46
Rys. 53. Przykład kodu	47
Rys. 54. Fragment pliku „loop.py”	47
Rys. 55. Fragment pliku „main.py”	47
Rys. 56. Fragment programu Postman pokazujący wysłane żądanie	48