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

1 Wstęp

1.1 Wersja polska

1.1.1 Temat pracy

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.

1.1.2 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.

1.1.3 Streszczenie działów

- Teoria : Tutaj można znaleźć użyte przeze mnie programy oraz narzędzia które pozwoliły mi stworzenie software`owej części systemu.
- Implementacja systemu : dział opisujący system składający się z trzech części :
  - Wymagania : założenia co do funkcjonalności systemu.
  - 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.
- Dwa tygodnie działania systemu : sprawozdanie wraz z wnioskami z dwóch tygodni testowania działania systemu w praktyce.
- Plany na dalszy rozwój : spis rzeczy które planuje poprawić lub dodać.

1.2 Wersja angielska

1.2.1 Thesis topic

The purpose of the described system is to automate process of plant upkeep. System let`s 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. System is placed on the Raspberry Pi platform and the Flask miniframework. The system is intended for pot plants, which is 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.

1.2.2 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.

1.2.3 Summary of sections

- Theory: Here you can find the programs and tools I used that allowed me to create the software part of the system.
- System implementation: section describing the system consisting of three parts :
    - Requirements : assumptions as to the functionality of the system..
    - 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.
- Two weeks of system operation: a report with conclusions of two weeks of testing the system in practice.
- Plans for further development: list of things that it plans to improve or add.

2 Teoria

2.1 Korzyści hodowania roślin w domu

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.

2.2 Hodowanie roślin doniczkowych

Każdy gatunek roślin doniczkowych uprawianych w domu potrzebuje indywidulanego 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. Problematyczne 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 zły 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 złą lokalizacje rośliny. Pomagają z tym specjalne lampy przystosowane do imitowania promieni słonecznych, należy jednak pamiętać że za długie naświetlanie kończy się przemęczeniem 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 nas najlepiej rozwijają się przy dużej ilości światła, wysokiej temperaturze latem (pokojowej) i chłodnej 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 odpowiednie 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.

2.3 Kontrola urządzeń elektronicznych przez urządzenia mobilne

Najbardziej popularnym zastosowaniem komunikacji między urządzeniami elektronicznymi a urządzeniami mobilnymi jest smart home, nie mniej możliwość komunikacji powoli przechodzi na inne dziedziny. Widać to coraz częściej w przemyśle kulinarnym przy produktach takich jak Termomix (urządzenie wielofunkcyjne) czy maszyn do Sous Vide (wolnego gotowania). Urządzeń do zautomatyzowanej pielęgnacji roślin nie ma dużo (kilka produktów opisanych jest dalej) i wiele z nich posiada poważne wady.

Dużym plusem zastosowania urządzeń mobilnych do kontroli urządzeń jest ich popularność co sprawia że urządzenia sterujące są często aktualizowane. Ponadto pozwala to na zmniejszenie kosztów produktu (ponieważ nie musimy produkować pilota) oraz zdjęcia części odpowiedzialność z firmy produkującej.

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

2.4 Podobne produkty

Automated Indoor Gardener

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

Plusy:

- Urządzenie jest kompaktowe.
- Można dostosować nachylenie lampy.

Minusy:

- Nie ma żadnych mierników sprawdzających czy kwiat faktycznie potrzebuje nawodnienia czy nasłonecznienia.
- Brak inteface`u umożliwiającego konfiguracje ustawień, żeby zmienić godziny nasłonecznienia, bądź nawadniania potrzebujemy rozkręcić obudowę i podłączyć się bezpośrednio do Raspberry PI.
- Nie jest to produkt komercyjny, a otwarto źródłowy projekt co znaczy że nie możemy kupić gotowych komponentów, lecz budowa wszystkich jest opisana na stronie twórców.

FarmBot

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

Plusy:

- Dobrze zaprojektowany interface użytkownika.
- Duża ilość funkcjonalności.

Minusy:

- Wielkość.
- Cena.
- Nie nadaje się do użytku domowego.
- Tak jak poprzednie urządzenie nie jest to produkt komercyjny.
- Brak kontroli oświetlenia

AeroGarden Harvest wi-fi

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

Cena : 730zł

Plusy:

- Dostosowywana wysokość lampy.
- Informowanie użytkownika.

Minusy:

- Mały pojemnik na wodę.
- Za mocne światło wysuszające roślinę.
- Brak automatyzacji.
- Zwykła lampa.
- Powiadomienia przychodzą niezależne od godziny co sprawiało że użytkownicy często dostawali powiadomienia w godzinach nocnych

SmallGarden 2

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

Plusy:

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

Minusy:

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

Smart Garden 9

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 : 764 zł

Plusy:

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

Minusy:

- Możliwość sadzenie jedynie roślin od producenta.

Wnioski :

Oświetlenie

Pierwszą rzeczą którą zamierzam poprawić jest oświetlenie, zamierzam 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 zamierzam zadbać by roślina dostawało dodatkowe oświetlenie wyłączę w godzinach których go potrzebuje oraz

wtedy gdy światło słoneczne nie będzie dla danej rośliny wystarczające co nie było zaadresowane w żadnym z przeglądanych przeze mnie produktów.

Nawodnienie

Musi odbywać się automatycznie oraz wtedy gdy roślina tego potrzebuje.

Budowa urządzenia

Więkość produktów daje gotowe obudowy co ograniczona wielkość roślin. Chciałbym żeby moje urządzenie było 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, oraz komputerów. Komunikaty nie mogą być natarczywe. Informacje przekazywane użytkownikowi powinny być pogrupowane by użytkownik nie czuł natłoku informacji.

2.5 Uzasadnienie wyboru tematu pracy

Po przejrzeniu dostępnych produktów uznałem że jest dużo miejsca na poprawę tego typu urządzeń. Chciałbym bardziej zagłębić się zarówno w automatyzacje codziennych procesów jak i komunikacje między urządzeniami. Jest to bardzo inspirujący temat, oraz z aktualnym rozwojem IOT (ang. Internet of things) oraz cyfryzacji domów, może stać się w najbliższym czasie bardzo dochodowy.

2.6 Wymagania systemu

2.6.1 Wymagania funkcjonalne

- Aplikacja do kontroli urządzenia musi być dostępna przez przeglądarki internetowe.
- Aplikacja musi skalować się urządzenia mobile.
- Aplikacja pozwala na komunikacje z system przez Web API.
- Użytkownik może zapisać ustawienie rośliny.
- System musi zapewniać roślinie podane przez użytkownika parametry.
- System wykonuje polecone 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.

2.6.2 Wymagania niefunkcjonalne

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

2.7 Użyte Programy

Visual Studio Code

Jest edytorem kodów źródłowych stworzonym przez firmę Microsoft. Podstawowa wersja zajmuje niewiele miejsca na dysku, dzieje tak dlatego 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 ale 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 to 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 połącznia 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 reakcje 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.

2.8 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 zwierająca 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 składnie. 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 nieżelaznego od platformy, wykonywalnego kodu bajtowego. Python z założenia jest modularny, kernel jest bardzo mały 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 przyśpieszające 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. Dla zainteresowanych bardziej kompleksowym frameworkiem do API w Pythonie zachęcam do poczytania o Django. 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 komplikowało by 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 kopi zapasowych.

Ważne cechy SQLite:

- Wieloplatformowość : można użyć go do aplikacji desktopwej, 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.

3 Implementacja systemu

3.1 Hardware

3.1.1 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 wilgoci gleby	8 zł
Taśma LED 300 GROW 1m	50 zł
Pompa 5V	20 zł
razem	317 zł

3.1.2 Opis komponentów

3.1.2.1 Raspberry Pi 3 B+

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 jest bazowany na Debianie (dystrybucja linuxa). Dwie Główne wyróżniające cechy Raspberry Pi to jedynie 5V wymaganego zasilania oraz GPIO (ang. general prupose 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.

3.1.2.2 Moduł przekaźników

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

3.1.2.3 ADC (konwerter analogowo cyfrowy)

Jako że Raspiberry 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 stała częstotliwością. Dokładność konwersji zależy głownie 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.

3.1.2.4 Fotorezystor

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

Użyty w systemie jest fotorezystor GL5537-1

Specyfikacja :

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

3.1.2.5 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

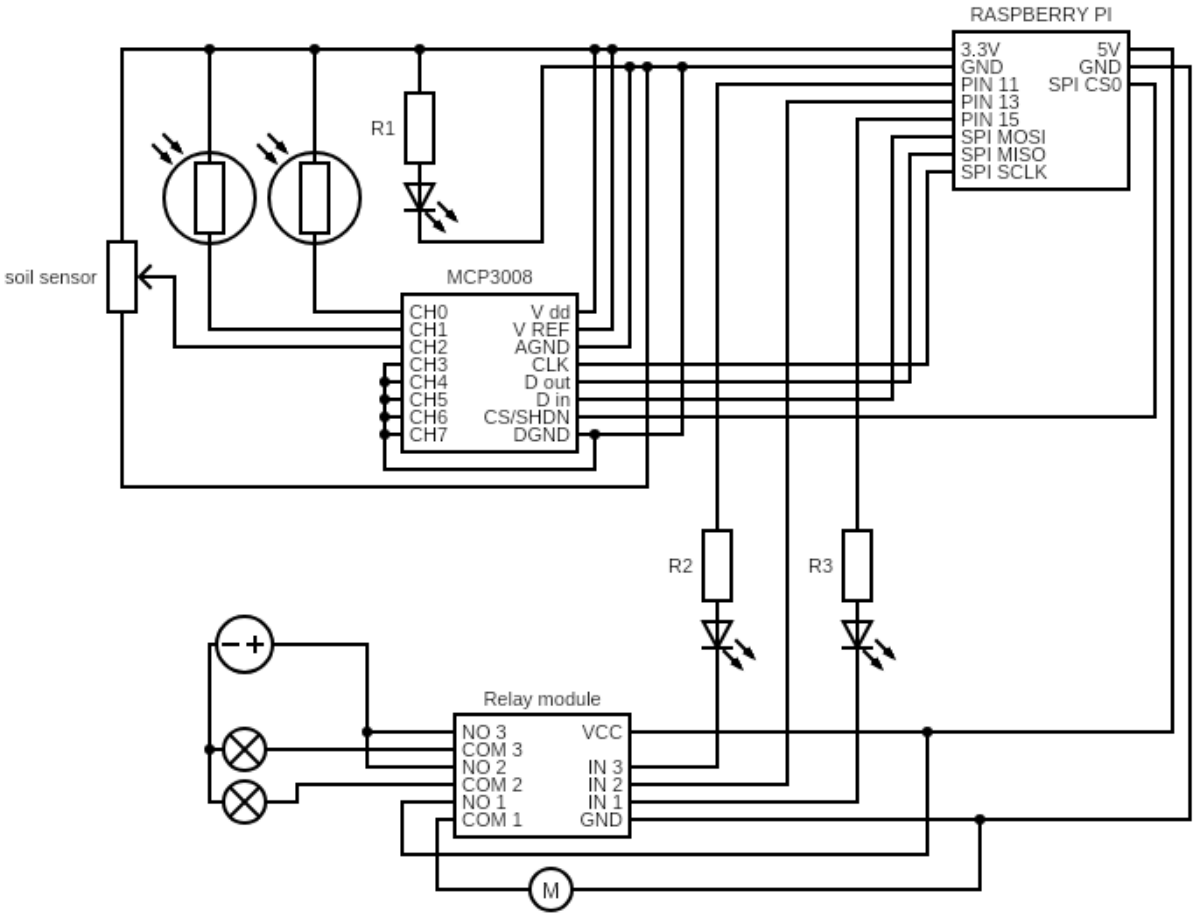
odczytując wyniki za pomocą ADC, działa to ponieważ im większa wilgoć tym mniejsza rezystancja gleby.

Użyty w systemie jest czujnik Iduino ME110

Specyfikacja :

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

3.1.3 Schemat układu



Rysunek 1 schemat opisywanego układu (wykonanie własne)

3.1.4 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 (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.

3.2 Software

3.2.1 Konfiguracja serwera

3.2.1.1 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 przyzwyczajenia 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 konfiguracje 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/dhcpcd.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. Przypomi 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
```

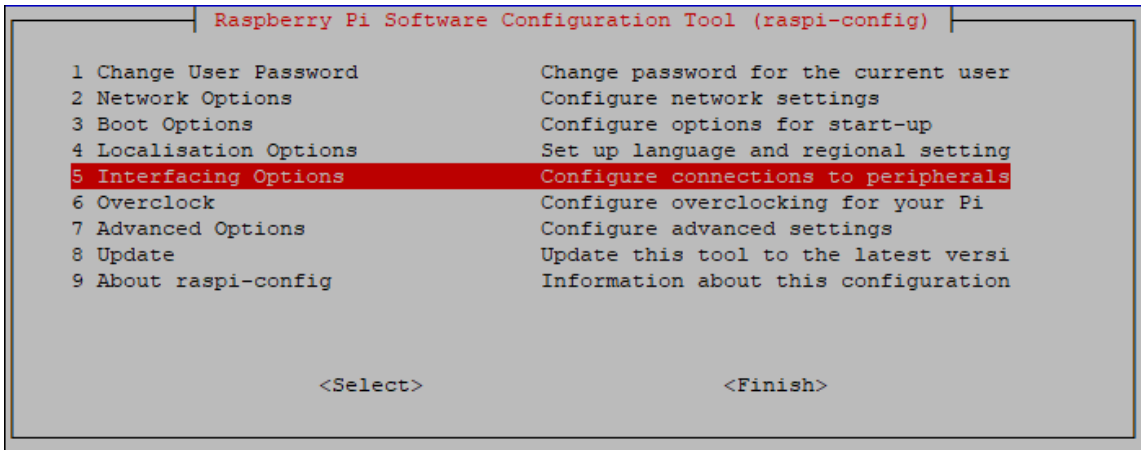
Rysunek 5 fragment konsoli

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

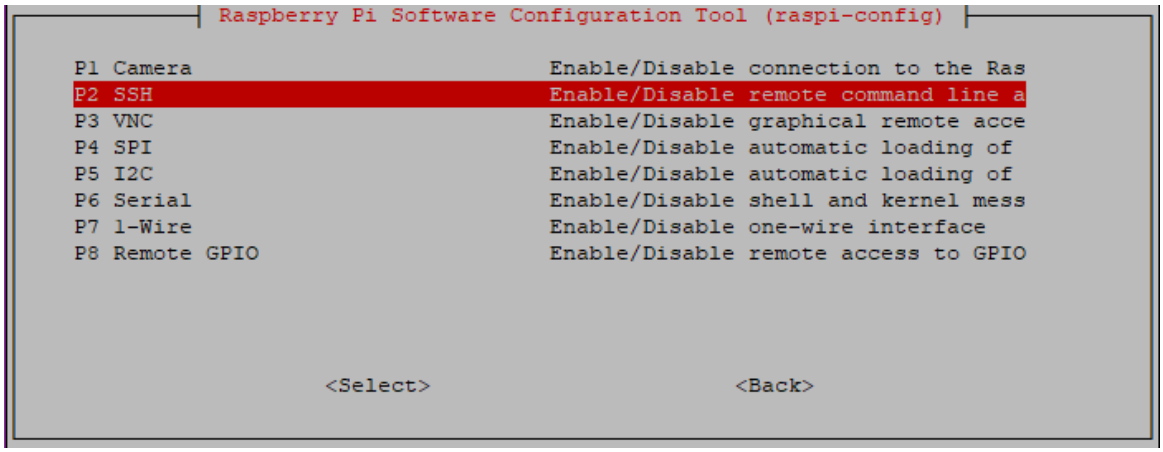
```
pi@raspberrypi:~ $ sudo raspi-config
```

Rysunek 6 fragment konsoli

A następnie wybrać podkreślone na czerwono opcje :



Rysunek 7 fragment konsoli



Rysunek 8 fragment konsoli

Teraz zostało wpisać w konsoli :

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.

3.2.1.2 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 planowana zadań w czasie w systemach Unix`owych. Pozwala na zaplanowanie wykonania skryptu w określonym czasie, bądź bo określonej operacji.

By użyć Crontab należy wpisać komendę :

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

Rysunek 10 fragment konsoli

Co otworzy nam plik w którym możemy dodać nasz plik z oznaczeniem @reboot co sprawi że będzie on wykonywany przy każdym starcie systemu.

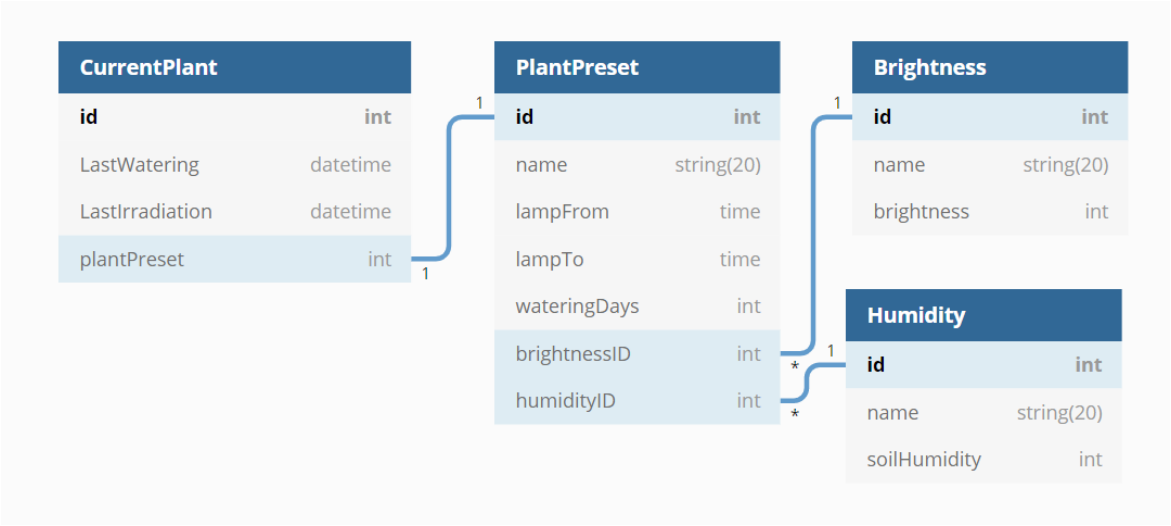
```
  GNU nano 2.7.4         Plik: /tmp/crontab.n0XQYT/crontab

# 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
```

Rysunek 11 fragment konsoli

3.2.2 Utworzenie bazy danych

3.2.2.1 Schemat bazy danych



Rysunek 12 schemat bazy danych (opracowanie własne)

CurrentPlant – tabela która przechowuje informacje odniesienie aktualnie wybranych ustawień 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”).

3.2.2.2 Dodanie bazy danych

3.2.2.2.1 Code first aproach

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

3.2.2.2.2 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 yśli auto inkrementacje, indeksowanie itp. Zmienna __tablee__ nie jest konieczna ale pozwala na 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”

3.2.2.3 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

3.2.2.4 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()
```

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

3.2.3 API

3.2.3.1 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 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 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 dowalonych 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 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 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 odpowiedz 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"

3.2.4 Widoki

3.2.4.1 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

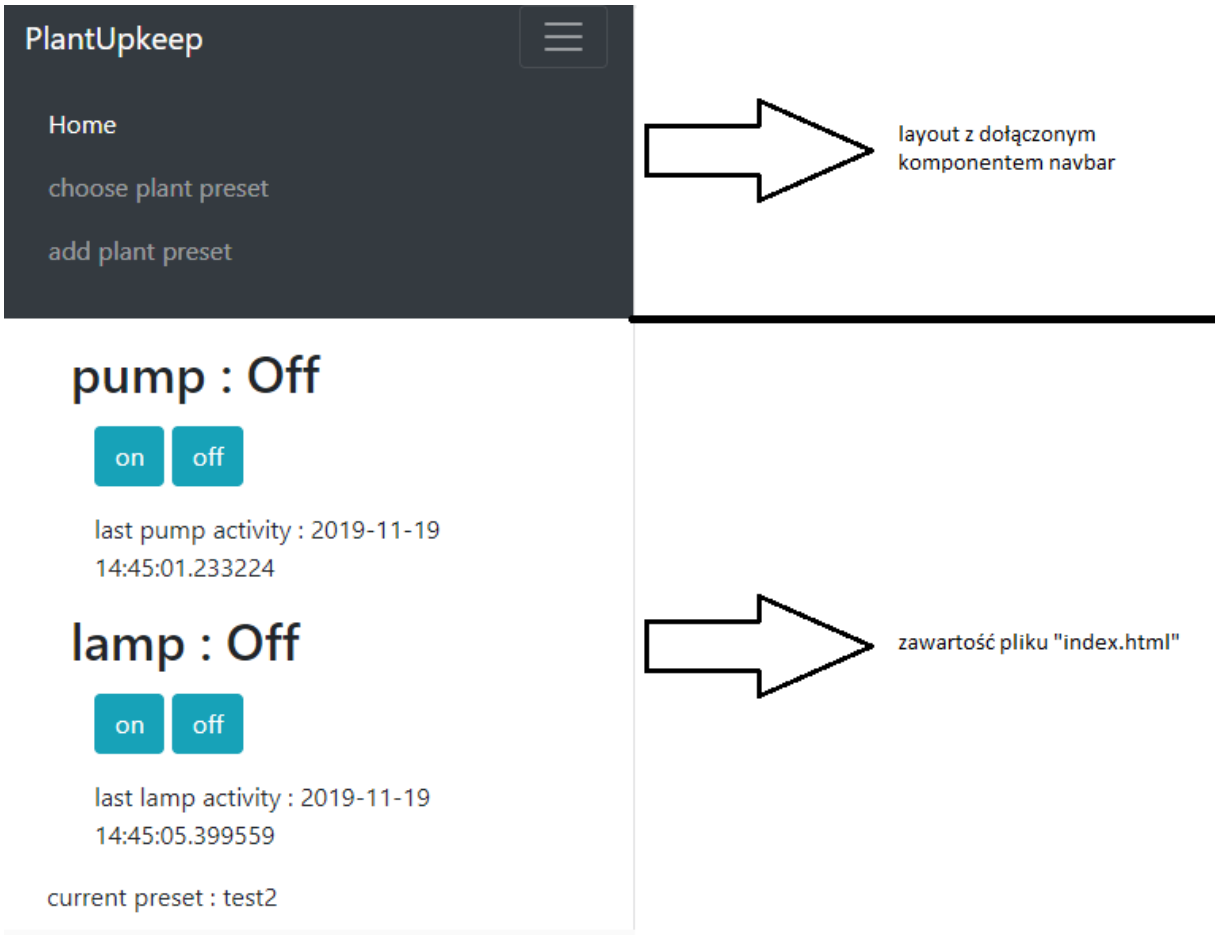
```
{% extends 'layout.html' %}
```

Który dołącza 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 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.

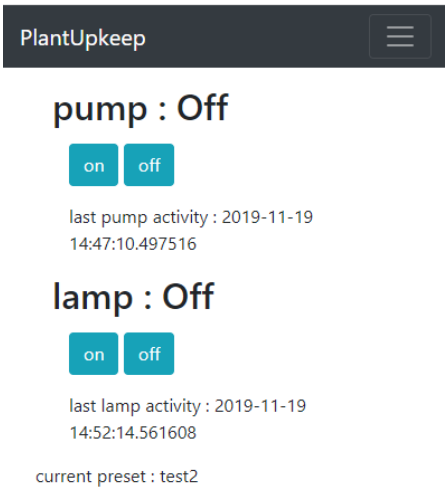
Wynik :



Rysunek 26 wygenerowany widok „index.html”

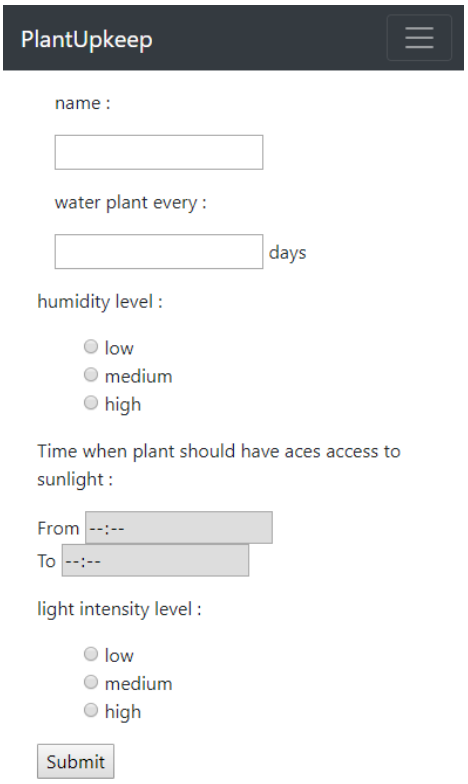
3.2.4.2 *Widoki w systemie*

System zawiera 3 widoki



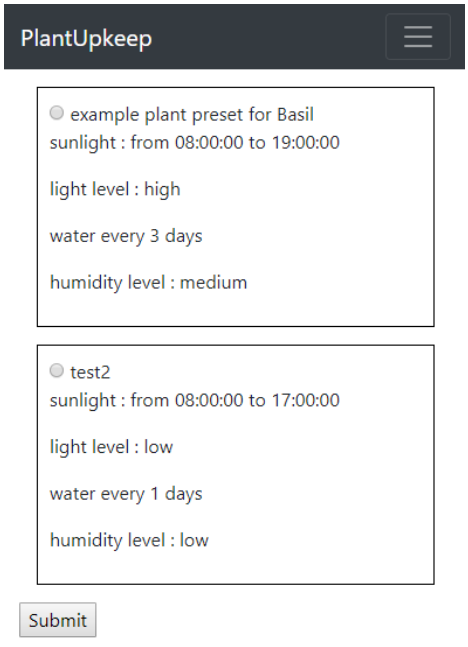
Rysunek 27 wygenerowany widok „index.html”

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 28 wygenerowany widok „addPreset.html”

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



Rysunek 29 wygenerowany widok „chosePreset.html”

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

3.2.5 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 :

3.2.5.1 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 funkcje 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.

3.2.5.2 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 funkcje 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 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.

3.2.6 Wykonywanie funkcji w zaplonowanym 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 do planera wszelkie funkcje, bądź inne wywoływalne obiekty (callable) by wykonywały się w wybranych przez nas okresach czasu.

3.2.6.1 *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 kontrole i przejrzystość obiektu.

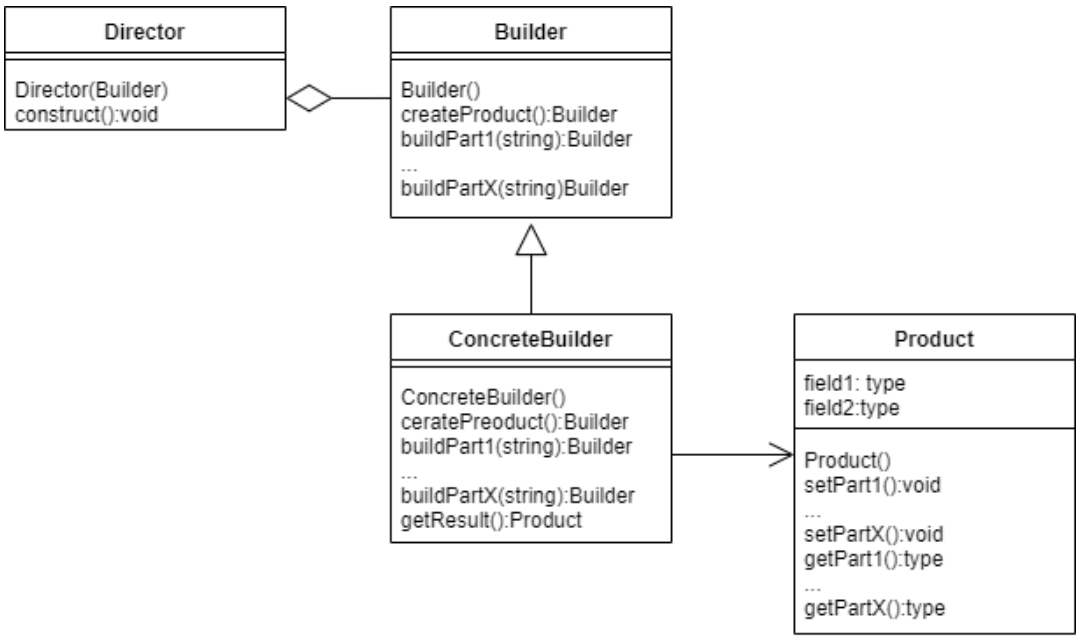
Zalety :

- 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 Reposibility Principle) : pozwala na izolacje 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 rządzenia, więc trzeba ustawić pętle nadzorującą zadania jako osobny wątek, do tego użyłem modułu „threading" który pozwala na tworzenie interface`ów wątków ponad nisko poziomowym 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 pętle nadzorującą, do tego skorzystałem z modułu „singal" który pozwala na przechwycenie sygnału i wyłączenie wszystkiego w wybranej przez programistę kolejności.

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

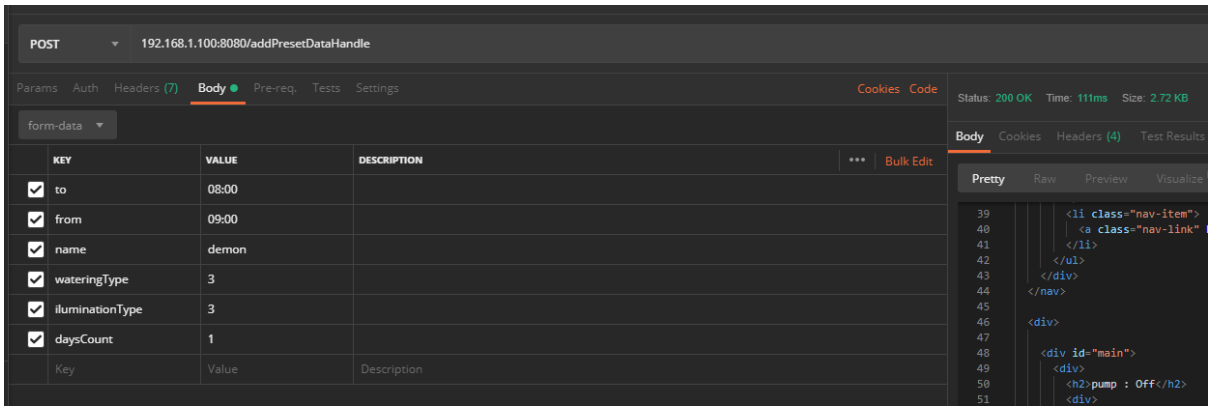
Rysunek 35 fragment pliku „main.py"

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

3.2.7 Komunikacja z systemem bez użycia strony internetowej

3.2.7.1 Przykład

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

3.2.7.2 *Spis* endpointów

<table>
<tr><td>Endpoint</td><td colspan="3">addPresetDataHandle</td></tr>
<tr><td>Metoda</td><td colspan="3">POST</td></tr>
<tr><td rowspan="8">Ciało żądania</td><td colspan="3">Formularz z danymi :</td></tr>
<tr><td>Nazwa</td><td>Typ danych</td><td>Opis</td></tr>
<tr><td>e</td><td>string</td><td>Nazwa ustawienia</td></tr>
<tr><td>from</td><td>time</td><td>Czas od którego roślina będzie naświetlana</td></tr>
<tr><td>to</td><td>time</td><td>Czas do którego roślina będzie naświetlana</td></tr>
<tr><td>dayCount</td><td>int</td><td>Co ile dni będzie naświetlana roślina</td></tr>
<tr><td>iluminationType</td><td>int</td><td>Id ustawień światła</td></tr>
<tr><td>wateringType</td><td>int</td><td>Id ustawień wilgotności gleby</td></tr>
<tr><td>Działanie</td><td colspan="3">Zapisuje nowe ustawienie rośliny</td></tr>
</table>

Endpoint	iluminationtypes
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

3.2.8 Testy

3.2.8.1 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 odpowiedz 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 na zapewnienie że API spełnia wymakania funkcjonalne jak i nie funkcjonalne.

4 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 w Raspberry PI.

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

W przyszłości system będzie korzystać z połączenia HTTPS oraz odpowiednio zabezpieczonego logowania.

System będzie informować użytkownika o problemach takich jak brak wody w zbiorniku, za niska wilgotność powietrza, zła tempereatura niezależnie od tego czy sam system jest w stanie wpłynąć na te parametry.

Zamierzam także zwiększyć ilość informacji jakie może zobaczyć użytkownik, przez dodanie większej liczby czujników.

5 Publikacje

5.1 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/

https://muratordom.pl/ogrod/rosliny/uprawa-roslin-doniczkowych-w-domu-ogolne-zasady-aa-vRbS-Wrc1-4pEP.html

https://poradnikogrodniczy.pl/

5.2 Książki

Samouczek HTML – Karol Wierzchołowski

Learning Python 5ed – Mark Lutz

Mastering the Raspberry Pi - Warren Gay

5.3 Artykuły

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