AKADEMIA GÓRNICZO-HUTNICZA

Raport z projektu

System nawadniania ogrodu

z przedmiotu

Sensory w Aplikacjach Wbudowanych

Elektronika i Telekomunikacja

Systemy Wbudowane

rok I studiów magisterskich

Paweł Majewski
Klaudia Litwin
Agata Krześniak
Jakub Żaba

15.06.2023

Spis Treści:

1. Opis i założenia projektu

Poniżej przedstawione zostały założenia projektowe oraz tabela z podziałem pracy w projekcie.

Założenia projektowe i wysokopoziomowy opis projektu

- Pomiar wilgotności gleby.
- Komunikacja dzięki WiFi.
- Przygotowanie Rest Api.
- Sterowanie systemem przy pomocy strony Internetowej (przetwarzanie przesyłanych danych).
- Danie możliwości użytkownikowi ustalania poziomu od którego zaczyna się nawadnianie.
- Dane przetwarzane w ESP32 na wartość procentową (odczytywanie sygnału analogowego i odpowiednia konwersja) i wysyłane do Rest Api.

Podział odpowiedzialności

Kamień milowy / funkcjonalność	Członek zespołu
Ogólne	
Ustalenie szczegółów projektu	Wszyscy
Raport / prezentacje	Wszyscy
Sprzęt	
Projekt obwodu elektronicznego	Paweł Majewski
Lutowanie obwodu na płytce uniwersalnej	Paweł Majewski
Embedded	
Obsługa czujnika	Paweł Majewski
Obsługa połączenia z Rest Api poprzez wifi	Paweł Majewski
Obsługa „imitacji" zraszacza poprzez obsługę servo mechanizm	Paweł Majewski
Obsługa diod LED	Paweł Majewski
Algorytm odpowiedzialny za wybrane działanie urządzeń	Paweł Majewski
Backend	
Budowa i uruchomienie Rest Api oraz przetestowanie połączenia poprzez wifi z innymi urządzeniami	Agata Krześniak
Przygotowanie bazy danych oraz integracja bazy danych z Rest Api (pobieranie / odczytywanie danych z bazy danych, obsługa podstawowych błędów związanych z bazą danych)	Agata Krześniak
Backend : • Możliwość pobierania danych z czujników • Możliwość wyświetlania danych • Możliwość dodawania danych z czujników • Możliwość ustawiania trybu pracy automatycznego / manualnego • Możliwość ustawiania progu wilogotności • Możliwość włączania / wyłączania zraszacza • Sterowanie procesem nawadniania	Agata Krześniak

• Obsługa niektórych błędów	
Testy jednostkowe aplikacji od strony backendu (m.in. testy pobierania danych z czujników, testy dodawania danych czujników, testy zmiany trybu pracy)	Agata Krześniak
Frontend	
Stworzenie layoutu aplikacji	Klaudia Litwin
Stworzenie logikii działania aplikacji: • Wyświetlanie danych z czujników dla użytkownika • Pobieranie danych od użytkownika • Stworzenie obsługi trybu automatycznego / manualnego • Stworzenie możliwości zadawania progu wilgotności • Stworzenie sygnalizacji czy dany czujnik jest uruchomiony • Stworzenie możliwości uruchomienia zraszaczy • Stworzenie komunikatu dla użytkownika o trającym nawadnianiu • Stworzenie blokad wprowadzania danych • Ładowanie strony	Klaudia Litwin
Połączenie frontendu z Rest Api	Klaudia Litwin
Testy połączenia frontendu z Rest Api	Klaudia Litwin
Testy manualne aplikacji	Klaudia Litwin

2. Sprzęt

Wybór komponentów

- Mikrokontroler Esp32 – posiada wbudowany moduł odpowiadający za obsługę wifi.
- Czujnik wilgotności gleby (Capacitlive Soli Moisture Sensor) – pojemnościowe czujniki wykazują się większą odpornością na korozję niż standardowe czujniki rezystancyjne.
- Servo mechanizm – był to najtańszy i najprostszy sposób imitacji zaworu wodnego.
- Diody LED, kabelki, płytki uniwersalne, goldpiny, powerbanki – były dostępne w osobistych zasobach.

Mikrokontroler i czujnik wilgotności gleby został również wybrany ze względu na chęć rozwijania swoich umiejętności, ponieważ był to pierwszy raz kiedy zostały użyte przez członków zespołu, drugim istotnym powodem były również konsultacje z prowadzącym zajęcia.

Układ mierzy poziom wilgotności gleby poprzez pomiar pojemności elektrycznej.

Kiedy gleba jest sucha, jej właściwości elektryczne się zmieniają, a pojemność kondensatora jest niska. W miarę wzrostu wilgotności gleby, zawartość wody w niej zwiększa jej przewodność elektryczną, co prowadzi do wzrostu pojemności kondensatora. Na podstawie tej zmiany pojemności można oszacować jaka jest wilgotność gleby.

Wybór czujnika

Wykorzystany w projekcie sensor to SZYTF SEN0193. Umożliwia on pomiar wilgotności gleby w sposób nieinwazyjny i niezawodny. Moduł ten, został wybrany ze względu na jego specyfikację

oraz korzyści wynikające z zastosowania technologii pojemnościowej. Moduł SZYTF SEN0193 jest wyposażony w wyprowadzenie analogowe. Dzięki temu, można odczytywać wilgotność gleby w czasie rzeczywistym i kontrolować jej poziom.

Specyfikacja czujnika:

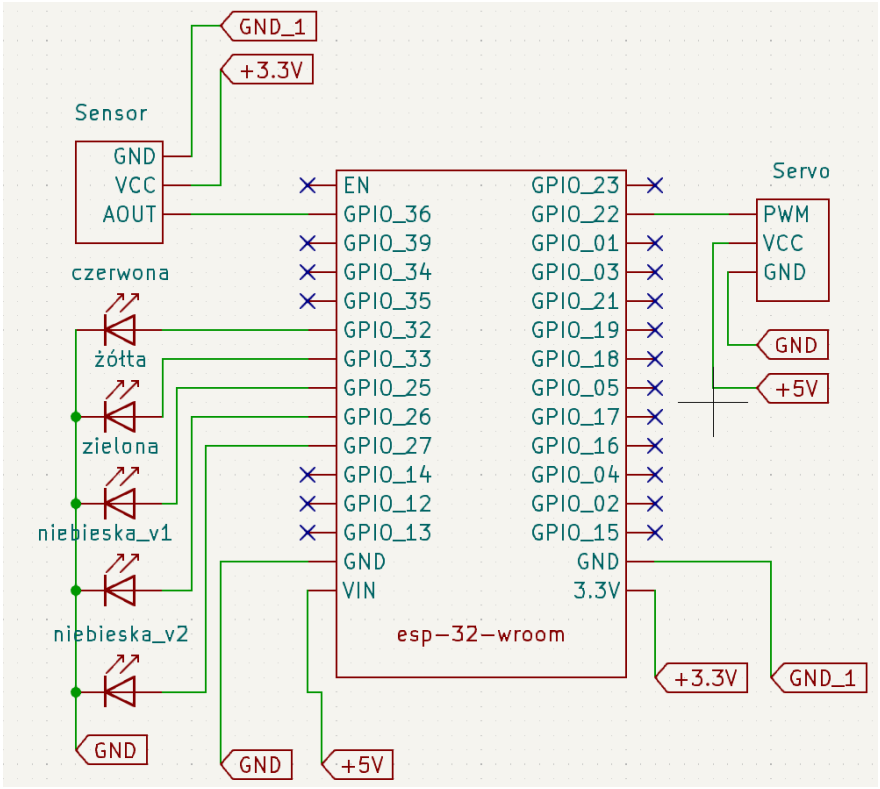
- Napięcie pracy : 3.3~5.5 VDC
- Napięcie wyjściowe: 0~3,0 VDC
- Wymiary: 98 x 23 mm

Czujnik pojemnościowy, taki jak czujnik wilgotności gleby SZYTF SEN0193, jest preferowany w porównaniu z czujnikiem rezystancyjnym ze względu na kilka istotnych czynników:

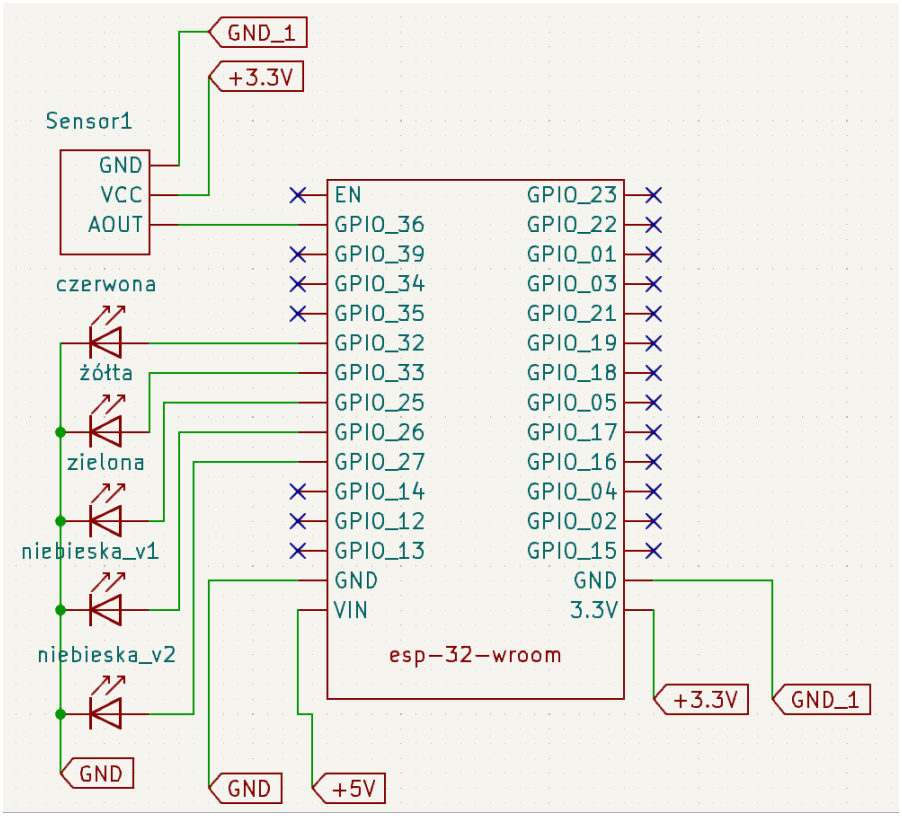
- Rezystancja gleby nie jest wiarygodnym wskaźnikiem wilgotności, dlatego lepiej jest mierzyć pojemność. Pomiar pojemności odnosi się do pojemności między elektrodami, które są umieszczone w glebie. W przypadku czujnika pojemnościowego, pojemność zależy od wielkości, powierzchni i odległości elektrod, a także od przenikalności dielektrycznej gleby.
- Przenikalność dielektryczna gleby jest bezpośrednio związana z jej wilgotnością. Powietrze, piasek i inne substancje obecne w glebie mają niską przenikalność dielektryczną, podczas gdy woda ma znacznie wyższą przenikalność dielektryczną. Nawet niewielkie zmiany zawartości wody w glebie powodują znaczne zmiany pojemności między elektrodami czujnika wilgotności gleby.
- Czujniki pojemnościowe są skuteczniejsze w wykrywaniu nawet niewielkich zmian zawartości wody w glebie. Dzięki temu można dokładniej monitorować poziom wilgotności gleby, co jest istotne w systemach nawadniania ogrodu.
- Mniej podatne na zakłócenia związane z innymi substancjami obecnymi w glebie (piasek czy inne substancje niezwiązane z wilgotnością) są właśnie czujniki pojemnościowe. Czujniki rezystancyjne mogą być bardziej podatne na interferencje, co może prowadzić do mniej dokładnych pomiarów.

Opis schematu

Na potrzebę projektu przygotowano przykładowe schamaty ideowe mające na celu ułatwienie przygotowania urządzenia na płytce uniwersalnej. (Na schamatach nie uwzględniono niezbędnych rezystancji.)

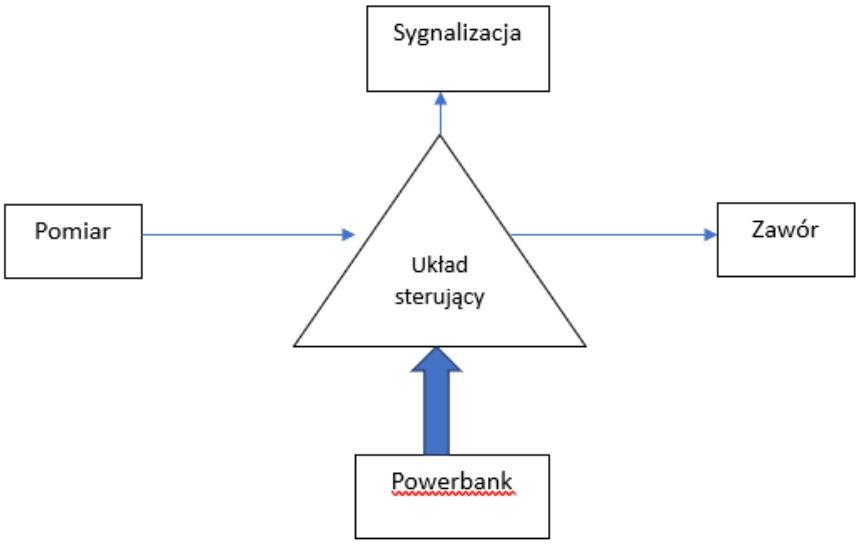


Rys.1 Schemat ideowy urządzenia pierwszego.



Rys.2 Schemat ideowy drugiego urządzenia.

Przygotowano dwa schematy różnią się jedynie tym, że na pierwszym jest dodatkowo umieszczony servo mechanizm. Dodatkowo podczas właściwego montażu dołożono niezbędne rezystory. Pierwszy schemat przedstawia urządzenie, do którego podłączone są diody LED, servo mechanizm i czujnik wilgotności gleby. Natomiast na drugim widzimy mikrokontroler, do którego podłączone są diody LED i czujnik. Powyższe układy można podzielić na bloki reprezentujące poszczególne elementy układu.



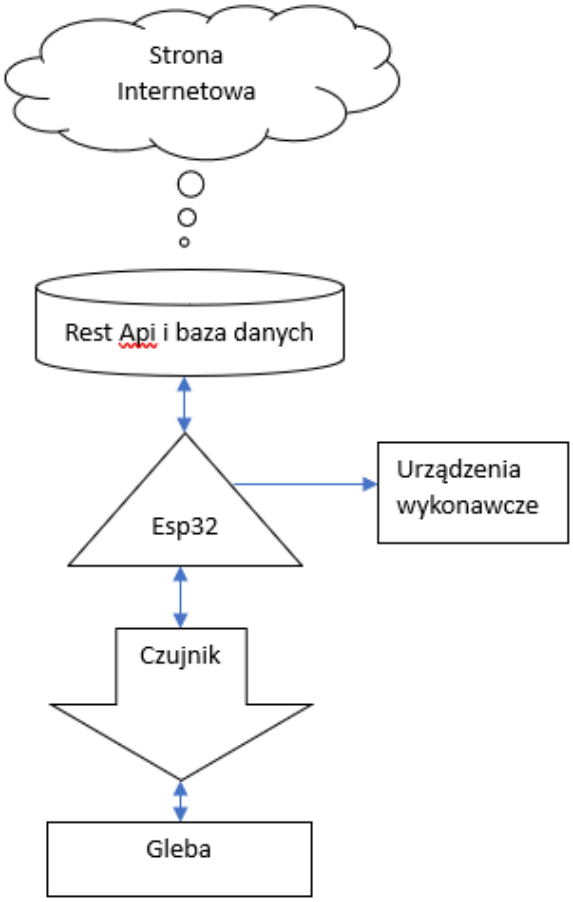
Rys.3 Schemat blokowy urządzenia.

Układ sterujący, czyli w tym przypadku ESP32 odpowiada za zarządzanie peryferiami i komunikację z Rest Api. Pomiar to blok w którym znajduje się czujnik wilgotności gleby. Zawór w tym przypadku imituje servo mechanizm. Sygnalizacja odpowiada za informowanie w jakim stanie jest urządzenie i jaki jest stan wilgotności gleby. Powerbank odpowiedzialny jest za dostarczenie energii niezbędnej do poprawnego działania układu.

3. Oprogramowanie

Architektura

Dane odczytywane z czujnika przekazywane są odpowiedniej obróbce opisanej w dalszej części raportu, następnie przekazywane są do Rest Api, które magazynuje dane i z którego korzysta strona internetowa, wszelkie dane przekazywane wprowadzane przez użytkownika na stronie Internetowej przekazywane są do Rest Api odpowiednio obsługiwane i na tej podstawie mikrokontroler po odczytaniu informacji zarządza podłączonymi urządzeniami wykonawczymi. Proces został przedstawiony na rysunku poniżej.



Rys.4 Przepływ danych

Poniżej znajduje się tabela z informacjami dotyczącymi częstotliwości akwizacji danych.

Zadanie	Co jaki czas [min]
Odczyt czujnika kiedy:	
Nie nawadniamy	30
Nawadniamy	18
Nawadniamy ręcznie	1
Inne:	
Czas aktualizacji danych ze strony	1/60
Odświeżanie widoku aplikacji	1

Na potrzebę odczytu danych z pinu analogowego skonfigurowano odpowiednio ESP32, ustawiając rozdzielczość ADC na 11 bitów. Zapewniło to większą precyzję. Ustawiono również wzmocnienie kanału ADC na 11 dB, co pozwoliło na zwiększenie czułości i pomiar sygnałów o niższym poziomie. Wzmocnienie 11 dB pozwala na mierzenie sygnałów, które mają umiarkowany zakres. Dodatkowo odczytywane dane są uśredniane poprzez przeprowadzenie 20 pomiarów, zsumowanie ich i podzielenie przez ich ilość, ma to na celu wyeliminowanie ewentualnych nieprawidłowych odczytów z czujnika, co mogłoby zaburzyć działanie całego systemu. Tak otrzymana

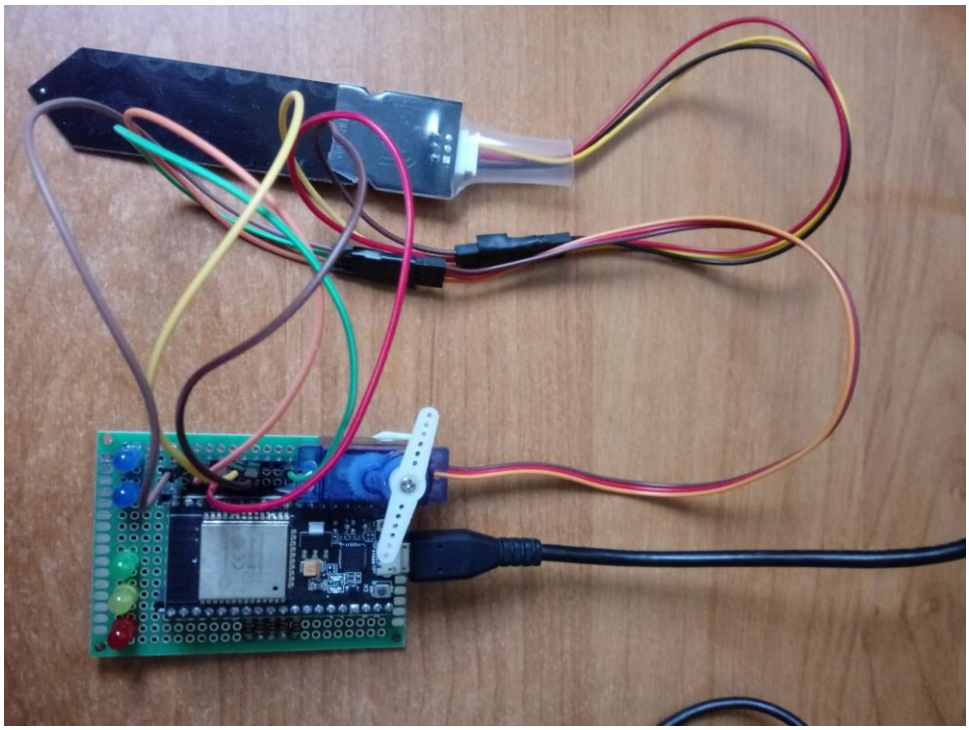
surowa wartość jest nastepnie konwertowana na procenty. Było to możliwe, dzięki wyznaczeniu skrajnych wartości odczytywanych z czujnika, gdy był całkiem suchy i gdy był zanurzony w wodzie.

Protokół HTTP jest używany do komunikacji między ESP32 a stroną Internetową, komunikacja odbywa się między serwerem, a klientem poprzez używanie odpowiednich metod używanych w żądaniach. Protokół HTTP (Hypertext Transfer Protocol) jest fundamentalnym protokołem komunikacyjnym, który umożliwia przesyłanie danych między klientem a serwerem w sieci. Jest on oparty na modelu żądanie-odpowiedź, gdzie klient wysyła żądania do serwera, a serwer udziela odpowiedzi na te żądania. Żądania HTTP są inicjowane przez klienta i zawierają informacje o tym, czego klient oczekuje od serwera. Odpowiedzi HTTP zawierają dane lub informacje zwracane przez serwer w odpowiedzi na żądanie klienta.

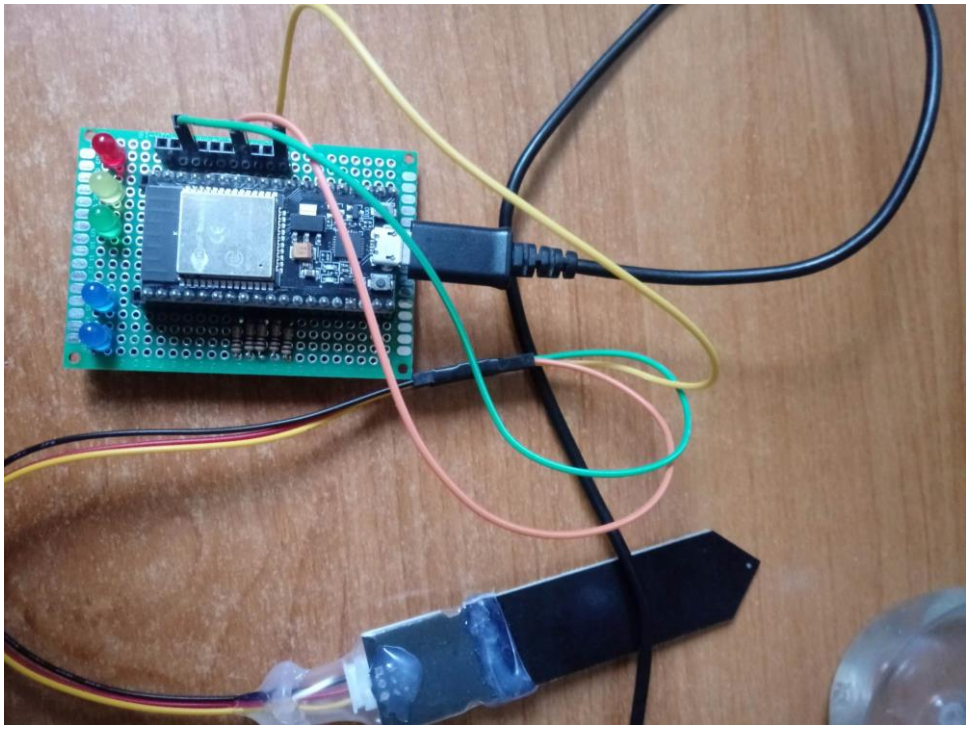
5. Wykonanie

Część sprzętowa

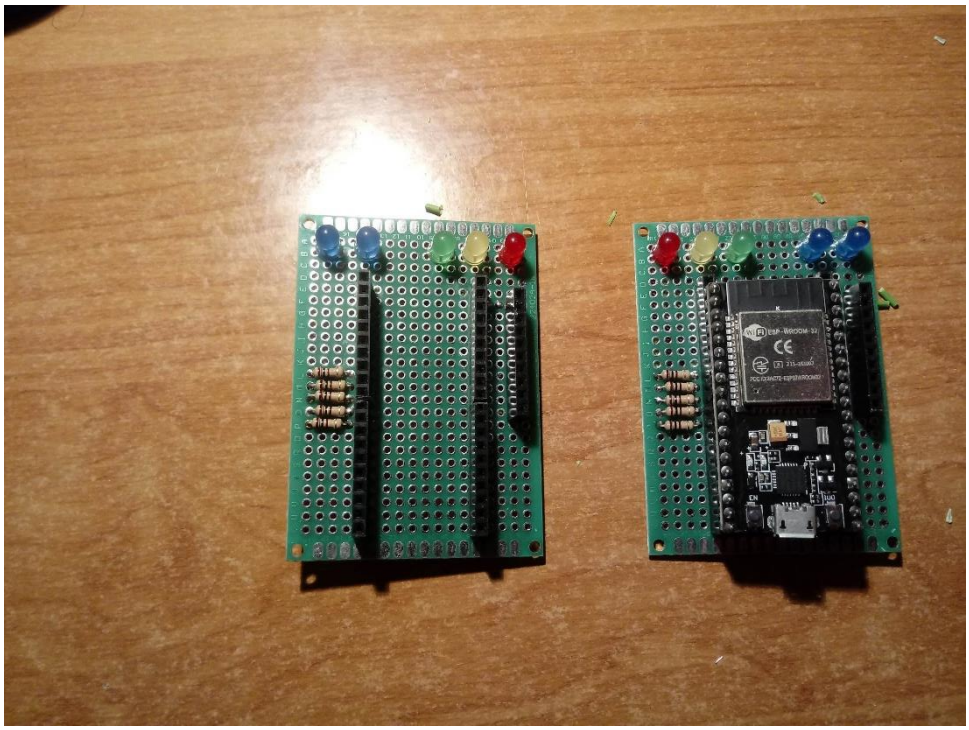
Na podstawie sporządzonego schamatu elektronicznego sporządzono układ. Poszczególne elementy zostały przylutowane na płytce uniwersalnej. Dzięki gold-pinom można w prosty sposób podłączyć/odłączyć poszczególne elementy układu.



Rys. 5 Płytka A(z servo mechanizmem)



Rys. 6 Płytka B



Rys. 7 Płytki bez podłączonych elementów.

Oprogramowanie na mikrokontroler

Na potrzebę projektu zostało napisane oprogramowanie na mokrokontroler ESP32. W tym celu wykorzystane visual studio code jak również rozszerzenie ESP-IDF Explorer. Zapewnia ono odpowiedni odczyt danych z czujnika, zarządza również innymi peryferiamii i odpowiada za komunikację bezprzewodową z Rest Api.

Cały projekt podzielono na poszczególne moduły ze względu na funkcjonalność jaką zapewniają.

- Leds - obsługa diod LED.
- Main - główny plik programu.
- Sensor - obsługa czujnika.
- Servo - obsługa servo mechanizmu.
- Task - zarządzanie zadaniami.
- Wifi_api – rozszerza funkcjonalność wybranych operacji bezprzewodowych.
- WiFi – podstawowa obsługa komunikacji WiFi

a) Ledy

Pliki zawierają funkcje do inicjalizacji odpowiednich pinów do których podpięte są diody LED, a także funkcje do sterowania diodami.

LED	Funkcja (co sygnalizują)
Czerwona (PIN_32)	Słabe nawodnienie (0-25%)
Żółta (PIN_33)	Średnie nawodnienie (25-75%)
Zielona (PIN_25)	Dobre nawodnienie (75-100%)
Niebieska (PIN_26)	Zawór otwarty/zamknięty
Niebieska (PIN_27)	Połączenie wifi

b) Main

Zawiera funkcję app_main() w której inicjalizowane są poszczególne peryferia podłączone do ESP32, a także tworzy taski odpowiedzialne za działanie mikrokontrolera i przypisuje je do wybranych rdzeni.

c) Sensor

Główna część projektu odpowiedzialna za właściwą obsługę czujnika. Zawiera struktury przechowujące odczytywane dane z czujnika (sensorData). Posiada funkcje do inicjalizacji czujnika, w której konfigurowana jest rozdzielczość i wzmocnienie kanału ADC. Posiada również funkcję, która zwraca przez wskaźnik aktualną wartość pomiaru. Surowe dane są odpowiednio uśredniane i przekształcane na wartość procentową.

d) Servo

Dla servo mechanizmu również napisano funkcję inicjalizującą, która odpowiednio kształtuje sygnał sterujący. Dzięki napisanym funkcją można zmieniać stan i ustawiać odpowiedni kąt.

e) Task

Mikrokontroler realizuje trzy zadania pierwsze z nich odpowiada za odczyt danych z czujnika przekazanie ich do Rest Api, jak również zapalenie odpowiedniej diody LED sygnalizujacej aktualne nawodnienie gleby. Częstotliwość z jaką aktualizowane są dane jest uzależniona od tego czy aktualnie nawadniamy ręcznie wtedy odczyt jest co minutę. Gdy trwa proces nawadniania dane aktualizowane są po zakończeniu tego procesu. Podczas normalnej pracy dane są uaktualniane co pół godziny. Kolejne zadanie dotyczy pobierania danych ze strony Internetowej, jest to dokonywane co sekundę dzięki temu system reaguje praktycznie od razu na żądanie ze strony użytkownika. Istnieje również jeszcze jedno zadanie, które realizuje tylko mikrokontroler posiadający servo mechanizm. Działanie polega na tym, że gdy użytkownik uruchomi ręcznie zraszacze zmieniany jest stan servo mechanizmu imitując otworzenie się zaworu z wodą. W przypadku gdy uruchomi sie automatyczne nawadnianie, urządzenie przechodzi przez kolejne kroki zapewniając odpowiednie nawodnienie gleby. Pierwszy z nich to

uruchomienie nawadniania na czas dwóch minut, kolejny to odczekanie minuty, następnie powtórzenie nawadniania przez kolejną minutę, a na samym końcu odczekanie pięciu minut. Dzięki temu woda ze zraszaczy powinna się wchłonąć, a odczyt po tym procesie powinien być prawdziwy.

f) wifi_api

Pliki zawierają funkcje odpowiedzialne, za odpowiednie formatowanie widomości w formacie JSON. Dzięki odpowiedniemu parsowaniu otrzymanych danych z Rest Api istnieje możliwość ich poprawnego przeanalizowania i podjęcia odpowiednich działań przez mikrokontroler. Dane przed wysłaniem do Rest Api muszą być również odpowiednio przekształcone do formatu JSON.

g) wifi

Pliki zawierają inicjalizację modułu wifi. Posiadają również funkcje obsługujące żądania GET i POST. W każdej funkcji zawarta jest obsługa błędów, które mogą wynikać z błędów podczas połączenia. Zawiera również funkcje odpowiedzialne za wysyłane i odbierania danych z REST API.

Cały program skupia się na odpowiednim odczycie danych z czujnika i przekazaniu ich do REST API. Zapewnia również odpowiednią reakcję na otrzymywane dane poprzez sterowanie urządzeniami wyjściowymi.

Aplikacja

Aby zapewnić użytkownikom wygodną obsługę urządzenia, opracowano dedykowaną aplikację REST API. Ta zaawansowana technologia umożliwia komunikację i wymianę danych między użytkownikiem a urządzeniem w sposób intuicyjny. Dzięki temu użytkownicy mogą korzystać z urządzenia za pomocą interfejsu, który jest dostosowany do ich potrzeb i preferencji. Aplikacja REST API działa jako pośrednik, zapewniając prosty dostęp do funkcji i zasobów urządzenia, takich jak odczyt i zapis danych, kontrola parametrów czy wykonywanie określonych czynności.

Backend

Opracowana aplikacja została napisana przy użyciu frameworka Flask. Flask jest elastycznym frameworkiem do tworzenia aplikacji webowych w języku Python (do realizacji projektu wykorzystana jest wersja 3.9.7). Wybór tego frameworka do implementacji aplikacji był związany z jego skalowalnością i obszerną dokumentacją. Flask umożliwia definiowanie różnych endpointów, które reagują na żądania HTTP, takie jak GET lub POST. Każdy endpoint jest obsługiwany przez odpowiednią funkcję, która wykonuje konkretne operacje, takie jak pobieranie danych z bazy danych, przetwarzanie ich i zwracanie odpowiedzi w formacie JSON.

Kod odpowiedzialny za backand Api znajduje się w pliku swaw_api.py. Na początku należało przygotować środowisko, w tym celu w pliku swaw_api.py znajduje się importowanie niezbędnych modułów, definiowana jest klasa przechowująca dane z czujnika wilgotności, funkcje pomocnicze oraz inicjalizacja niektórych zmiennych, które używane są w dalszej części kodu. W celu możliwości wglądu w logi aplikacji, została dodana konfiguracja logowania, dzięki której logi Api zapisywane są do pliku “swaw_api.log”. Poniżej opisane zostały główne funkcjonalności napisane od strony backendowej.

Pobieranie danych z czujników

Endpoint: GET *“.../mainview”*

Aby zrealizować funkcjonalność pobierania danych z czujnika wilgotności, została

zaimplementowana funkcja *get_sensor_data()*, która przypisana jest do endpointa *.../mainview"* oraz obsługuje żądanie GET. Po otrzymaniu żądania HTTP typu GET na adres *" .../mainview"*, aplikacja nawiązuje połączenie z bazą danych przy użyciu modułu *mysql.connector*. Następnie wykonuje zapytanie SQL, które pobiera najnowsze dane z tabeli sensor_data dla czujników wilgotności. Otrzymane dane są przechowywane w postaci obiektów klasy SensorData, które zawierają informacje o identyfikatorze czujnika, wilgotności, stanie czujnika oraz stanie zaworu. Po pobraniu danych, aplikacja analizuje je w celu określenia procesu podlewania i ewentualnych błędów. Jeśli tryb pracy systemu jest ustawiony na automatyczny (zmienna *isAutomaticMode = true*), aplikacja sprawdza wilgotność dla obu czujników. Jeśli wilgotność w obu czujnikach jest poniżej 40%, ustawiana jest wartość flagi *watering_process_value* na 1, co oznacza potrzebę podlewania. Ponadto, jeśli przynajmniej jeden z czujników ma wilgotność poniżej 30%, również ustawiana jest wartość watering_process na 1. W przypadku, gdy czujnik z *sensor_id* = 1, zostanie wyłączony , a *sensor_id* = 2 będzie wskazywać potrzebę podlewnia – system nie włączy procesu nawadniania. Natomiast kiedy *sensor_id* = 2 będzie nieaktywny, a *sensor_id* = 1 będzie wskazywać wartość wilgotności poniżej progu – system włączy proces nawadniania. Ta sama logika zaimplementowana jest dla trybu manualnego, gdzie próg nawadniania podaje użytkownik. Dodatkowo kiedy są włączone oba czujniki, a jeden z nich pokazuje wartość wilgotności powyżej progu, natomiast drugi wskazuje, że wilgotność gleby przekroczyła zadany próg o 10% - zostanie aktywowany proces podlewania. Aplikacja obsługuje przypadki braku danych dotyczących wilgotności. Jeśli brakuje danych dla obu czujników, generowany jest komunikat błędu informujący o braku tych danych przesyłany przy pomocy flagi *error_message*. W przypadku braku danych dla jednego z czujników, generowany jest odpowiedni komunikat informujący o braku danych dla konkretnego czujnika. Jeśli wilgotność w jednym z czujników jest poniżej 10, również generowany jest odpowiedni komunikat informujący o zbyt niskiej wartości wilgotności. Dane odpowiedzi są formatowane w formacie JSON, zawierającym informacje o danych z czujników (*sensor_data* , o stanie procesu podlewania (*watering_process*) , stanie zaworu (*sprinkle_state*) oraz ewentualnym komunikacie błędu (*error_message*). Odpowiedź w formacie JSON jest zwracana jako odpowiedź HTTP.

Dodawanie danych z czujnika

Endpoint: POST *" .../mainview"*

Aby zrealizować tą funkcjonalność, zaimplementowana została funkcja *add_sensor_data()*, która przypisana jest do endpointa *" .../mainview"* oraz obsługuje żądanie POST. Ta funkcjonalność umożliwia dodawanie danych z czujnika wilgotności do bazy danych. Dane przekazane w żądaniu w formacie JSON, zawierające - identyfikator czujnika (*sensor_id*) , wilgotność (*humidity*) oraz informacje czy czujnik jest włączony (*is_sensor_on*) są pobierane. Następnie wykorzystując moduł *mysql.connector* nawiązane jest połączenie z bazą danych. Po nawiązaniu połączenia zaimplementowane zostało zapytanie INSERT w celu dodania danych do tabeli "*sensor_data*". Po wykonaniu zapytania, wprowadzone zmiany są zapisane poprzez wywołanie metody *commit()* na obiekcie połączenia. Na koniec połączenie jest zamykane aby utrzymać efektywność działania. Dane są formatowane przez moduł *jsonpickle* do formatu JSON i zwracane jako odpowiedź HTTP z kodem 201, który informuje o sukcesie żądania.

Ustawianie trybu pracy

Endpoint: POST *" .../mode"*

Ta funkcjonalność umożliwia zmianę trybu pracy – na automatyczny lub manualny, dla systemu. Aby tego dokonać zaimplementowana została funkcja *set_mode()* która przypisana jest do endpointu *" .../mode"* i obsługuje żądanie POST. W pierwszym kroku, wartość trybu automatycznego przekazana w żądaniu w formacie JSON jest pobierana za pomocą *request.get_json()*. Następnie, wartość ta jest przypisywana do zmiennej globalnej *isAutomaticMode* (True / False), umożliwiając globalne przechowywanie informacji o trybie automatycznym. Ostatecznie, jest zwracana odpowiedź w formacie JSON, informująca o sukcesie operacji.

Endpoint: GET *"/...mode"*

Funkcja została zaimplementowana w celu przetestowania możliwości zmiany trybu pracy. Aby tego dokonać zaimplementowana została funkcja *get_mode()* która przypisana jest do endpointu *" .../mode"* i obsługuje żądanie GET. Funkcja *get_mode()* wykorzystuje zmienną globalną *isAutomaticMode*, która przechowuje informację o bieżącym trybie pracy. Funkcja tworzy słownik *return_data* zawierający klucz *AutomaticMode* i wartość *isAutomaticMode* (*True / False*). Następnie słownik ten jest zamieniany na format JSON za pomocą funkcji *jsonify()*. Ostatecznie funkcja zwraca JSON jako odpowiedź.

Ustawianie progu wilgotności

Endpoint: POST *".../threshold"*

Funkcja *set_threshold()* obsługuje żądanie POST na ścieżce *".../threshold"*. Jej funkcjonalność polega na ustawieniu wartości progu nawadniania dla trybu manualnego. Funkcja pobiera wartość progu z żądania POST, korzystając z *request.json['threshold']*, gdzie *threshold* jest kluczem w danych przesłanych w żądaniu. Odczytana wartość progu jest przypisywana do zmiennej *manual_threshold*. Kolejno funkcja zwraca komunikat zawierający ustawioną wartość progu w formacie tekstowym.

Endpoint: GET *".../threshold"*

Funkcja została zaimplementowana w celu przetestowania możliwości ustawiania progu nawadniania. Funkcja *get_threshold()* obsługuje żądanie GET na ścieżce *".../threshold"*. Jej funkcjonalność polega na zwróceniu aktualnie ustawionej wartości progu nawadniania dla trybu manualnego. Na początku funkcja odczytuje wartość zmiennej manual_threshold. Następnie funkcja tworzy odpowiedź tekstową, która zawiera aktualnie ustawioną wartość progu. Wartość ta jest wstawiana w odpowiednie miejsce w tekście za pomocą metody *format()*. Dzięki temu wartość progu nawadniania zwracana jest w formacie tekstowym.

Włączanie / wyłączanie zraszacza

Endpoint: POST *".../sprinkle"*

Funkcja *sprinkler_toggle()* obsługuje żądanie POST na ścieżce *".../sprinkle"*. Jej funkcjonalność polega na przełączaniu stanu zraszacza na podstawie otrzymanych danych. Na początku funkcja pobiera dane z żądania w formacie JSON za pomocą metody *get_json()*. Następnie sprawdzane jest, czy w otrzymanych danych istnieje klucz 'sprinkler_on'. Jeśli klucz nie istnieje, funkcja zwraca odpowiedź z błędem i kodem 400, informując o brakujących parametrach. Po pomyślnym sprawdzeniu danych, funkcja nawiązuje połączenie z bazą danych, używając funkcji *get_connection_()*, następnie wykonuje zapytanie SQL, które aktualizuje stan zraszacza w tabeli sensor_data. W zapytaniu używane są otrzymane dane *sprinkler_on* jako wartość, która zostanie ustawiona. Kolejno, zmiany są zatwierdzane w bazie danych przez wykonanie metody *commit()* na obiekcie połączenia. Po

zakończeniu operacji na bazie danych, połączenie jest zamykane. Na koniec funkcja zwraca odpowiedź z informacją o sukcesie.

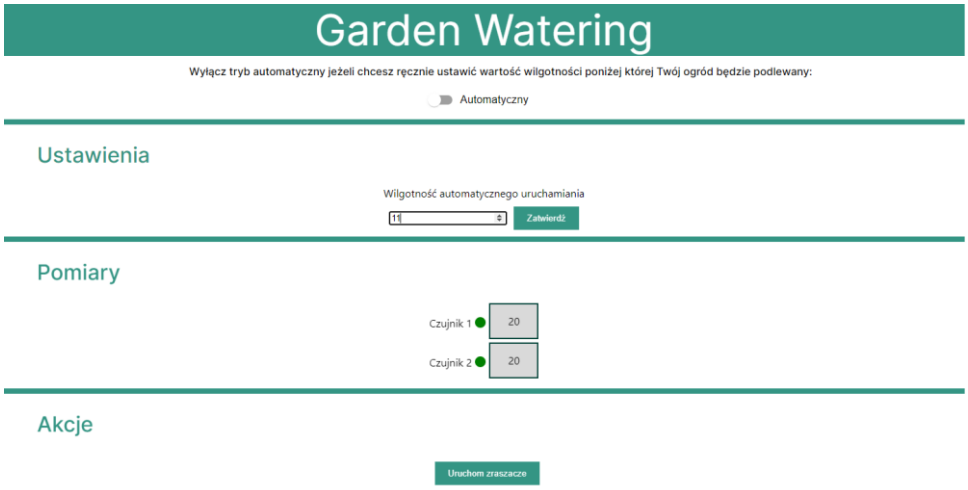
Frontend

W celu umożliwienia użytkownikowi odczytu wyników oraz modyfikacji ustawień do Rest Api został dopisany frontend. Został on napisany przy użyciu CSS, React'a i Typescript'a. Cały wygląd strony został zapisany w odpowiednich klasach, które są umieszczone w plikach CSS. Poszczególne pliki zostały umieszczone w folderach komponentów, których dotyczą. W tych folderach znajdują się również odpowiadające im pliki "index.tsx". Kod zawarty w tych plikach odpowiada za prawidłowe wyświetlanie i działanie poszczególnych komponentów a także całego frontendu aplikacji. Dodatkowo ze względu na powtarzające się style został stworzony plik "styleguide.css" zawierający zdefiniowane, niektóre wykorzystywane style, które powtarzają się w aplikacji.

Najważniejsze człony komunikacji z API zostały zawarte w folderze services. Zawiera on pliki "baseService.ts", w którym zdefiniowane są metody do wysyłania żądań HTTP oraz "APIService.ts" zawierający metody do interakcji z API.

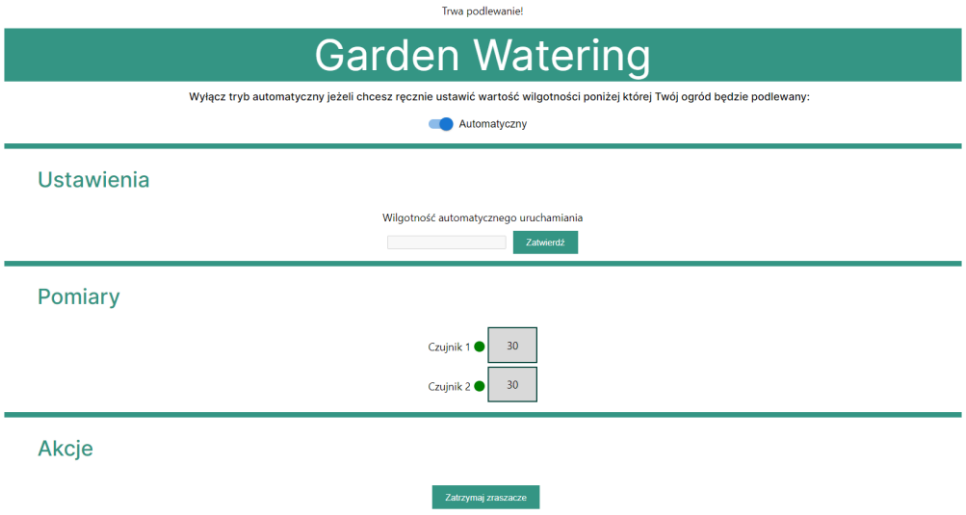
Dane, które są wyświetlane dla użytkownika są odświeżane na stronie co minutę oraz przy ręcznym przeładowaniu strony. Informacja o tym jest zawarta w zmiennej w pliku "index.tsx" dla Layoutu. Przy ładowaniu się strony została dodana ikona kręcącego się kółka informująca użytkownika, że ładowanie właśnie trwa.

Napisana aplikacja składa się z czterech sekcji, tak jak zostało to przedstawione na Rys.8



Rys.8 Podstawowy wygląd aplikacji

Pierwsza sekcja widniejąca w kodzie jako folder "TopSection" zawiera nazwę strony "Garden Watering" umieszczoną na pasku tytułowym oraz przycisk pozwalający na włączenie i wyłączenie trybu Automatycznego, wraz z powyżej umieszczonym opisem jego działania. W sekcji tej, powyżej tytułu pojawia się również informacja o tym, że w danym momencie trwa podlewanie. Zostało to zrealizowane tak by informacja pojawiała się zarówno, gdy podlewanie uruchomiło się automatycznie jak i gdy zostało uruchomione przyciskiem z aplikacji. Opisywana informacja o podlewania wygląda tak jak zostało to pokazane na Rys. 9.

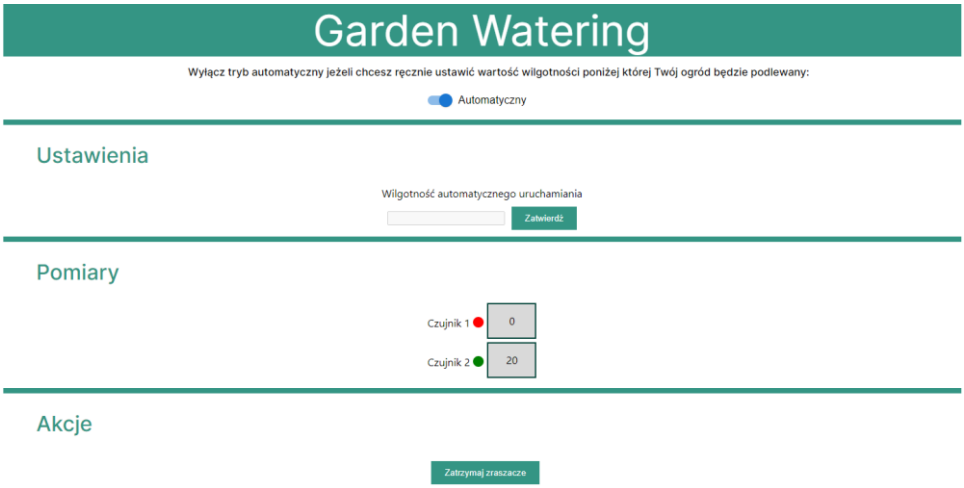


Rys.9 Widok strony, gdy trwa podlewanie

Dostępny w tej sekcji przycisk został zrealizowany jako Toggle Switch. Wartość ustawiona na nim przez użytkownika zostaje przekazana do Rest Api i określa tryb pracy naszego systemu nawadniania. W trybie automatycznym użytkownik ma możliwość ustawienia wartości wilgotności automatycznego podlewania w sekcji Ustawienia (“SettingSection”). Gdy przestawimy aplikację na tryb Automatyczny możliwość tej zmiany zostanie zablokowana, a pole zostanie wyszarzone tak jak zostało to zaprezentowane na Rys. 10.

Rys.10 Blokada w trybie Automatycznym

Przy polu odpowiadającym za pobranie wartości od użytkownika został umieszczony przycisk zatwierdzający wpisaną wartość i przesyłający ją dopiero po jego naciśnięciu, w celu zabezpieczenia aplikacji przed przesłaniem niepełnej bądź niepoprawnej wartości. Trzecią sekcją są pomiary (“MeasurementsSection”). Są w niej wyświetlane dane z dwóch czujników. Oprócz wartości liczbowej została również umieszczona dioda sygnalizująca stan czujnika. Gdy odbierana jest informacja, że oba czujniki są włączone obie diody mają kolor zielony, tak jak było to pokazane na Rys. 8. Gdy zostanie przekazana wartość informująca o tym, że któryś czujnik nie działa dioda zmieni kolor na czerwony, tak jak to zostało zaprezentowane na Rys. 11. Dioda została zrealizowana jako kropka, której kolor jest zależny od wartości parametru “isSensorOn”, który określa czy dany czujnik jest w tym momencie włączony.



Rys.11 Blokada w trybie Automatycznym

W ostatniej sekcji, Akcje (“ActionSection”) umieszczony został przycisk pozwalający na ręczne uruchomienie i zatrzymanie podlewania. Napis na przycisku w momencie, gdy nie trwa podlewanie to “Uruchom zraszacze”, ale gdy odbierana jest informacja, że podlewanie w danym momencie trwa zmienia się on na “Zatrzymaj zraszacze”, co można zaobserwować na Rys. 9.

6. Testy

W tej sekcji opisane zostały testy wykonane w ramach projektu. Zostały one zrealizowane w celu sprawdzenia poprawności działania programów, a także doboru odpowiednich ustawień dla urządzenia.

Testy Api

Poniżej przedstawione zostały wybrane testy jednostkowe wykonane w ramach przetestowania REST API.

1. Dla funkcji *get_sensor_data()*

Test case 1 Sprawdzenie poprawności działania w przypadku poprawnych danych z czujników.	
Dane wejściowe	Sensor 1 { "humidity": 65, "is_sensor_on": 1, "sensor_id": 1 } Sensor 2 { "humidity": 21, "is_sensor_on": 1, "sensor_id": 2 }
Oczekiwany rezultat	W odpowiedzi na endpoint ***GET”*** *http://localhost:5000/mainview”* otrzymano JSON z danymi o sensorach oraz flaga “watering_process” ustawiona na 1.

	Kod : HTTP/1.0 200 OK
Aktualny rezultat	Otrzymano oczekiwany rezultat
Test case 2 Sprawdzenie poprawności działania w przypadku, gdy jeden z czujników nie jest włączony.	
Dane wejściowe	Sensor 1 { "humidity": 70, "is_sensor_on": 1, "sensor_id": 1 } Sensor 2 { "humidity": 10, "is_sensor_on": 0, "sensor_id": 2 }
Oczekiwany rezultat	W odpowiedzi na endpoint ***GET"*** *http://localhost:5000/mainview"* otrzymano JSON z danymi o sensorach oraz flaga "watering_process" ustawiona na 0. Kod : HTTP/1.0 200 OK
Aktualny rezultat	Otrzymano oczekiwany rezultat

2. Dla funkcji *add_sensor_data()*

Test case 1 Sprawdzenie poprawności dodawania danych z czujnika.	
Dane wejściowe	Sensor 1 { "humidity": 65, "is_sensor_on": 1, "sensor_id": 1 }
Oczekiwany rezultat	Kod : HTTP/1.0 201 CREATED
Aktualny rezultat	Otrzymano oczekiwany rezultat

Test case 2 Sprawdzenie obsługi brakujących parametrów.	
Dane wejściowe	Sensor 1 { "is_sensor_on": 1, "sensor_id": 1 }
Oczekiwany rezultat	"error": "Brak niektórych parametrów" Kod : HTTP/1.0 400 BAD REQUEST
Aktualny rezultat	Otrzymano oczekiwany rezultat

3. Dla funkcji *set_mode()*

Test case 1 Sprawdzenie ustawiania trybu manualnego.	
Dane wejściowe	***POST*** *http://localhost:5000//mode* { "isAutomaticMode": false }
Oczekiwany rezultat	Kod : HTTP/1.0 200 OK Dodatkowo po przesłaniu żądania na : ***GET*** *http://localhost:5000//mode* w rezultacie : "results": Mode is set to false oraz kod : HTTP/1.0 200 OK
Aktualny rezultat	Otrzymano oczekiwany rezultat

Test case 2 Sprawdzenie ustawiania trybu automatycznego.	
Dane wejściowe	Należy upewnić się, że aktualnie włączony jest tryb manualny. Następnie przesłać żądanie: ***POST*** *http://localhost:5000//mode* { "isAutomaticMode": true }
Oczekiwany rezultat	Kod : HTTP/1.0 200 OK Dodatkowo po przesłaniu żądania na : ***GET*** *http://localhost:5000//mode* w rezultacie : "results": Mode is set to true oraz kod : HTTP/1.0 200 OK
Aktualny rezultat	Otrzymano oczekiwany rezultat

4. Dla funkcji *set_threshold()*

Test case 1 Sprawdzenie obsługi braku przekazanego progu wilgotności.	
Dane wejściowe	Należy upewnić się, że próg wilgotności nie jest ustawiony oraz włączony jest tryb manualny. Następnie przesłać żądanie: ***GET*** *http:/localhost:5000/mainview*
Oczekiwany rezultat	"error": "Wprowadz prog nawadniania" Kod : HTTP/1.0 400 BAD REQUEST
Aktualny rezultat	Otrzymano oczekiwnay rezultat
Test case 2 Sprawdzenie ustawiania progu wilgotności.	
Dane wejściowe	Należy upewnić się, że włączony jest tryb manualny. Następnie przesłać żądanie: ***POST*** *http://localhost:5000/threshold* { "threshold": 60 }
Oczekiwany rezultat	Manual threshold : 60 Kod : HTTP/1.0 200 OK Dodatkowo upewnić się wysyłając żądanie: ***GET*** *http://localhost:5000/threshold*

	Oczekiwany wynik Manual threshold : 60 Kod : HTTP/1.0 200 OK
Aktualny rezultat	Otrzymano oczekiwany rezultat

5. Dla funkcji *sprinkler_toggler()*

Test case 1 Sprawdzenie zmiany stanu zraszacza.	
Dane wejściowe	***POST*** *http:/localhost:5000/sprinkler* { "sprinkler_on": 1 }
Oczekiwany rezultat	"success": "true" Kod : HTTP/1.0 200 OK Następnie przesłać żądanie: ***GET*** *http:/localhost:5000/mainview* Flaga “sprinkle_state” powinna być ustawiona na 1.
Aktualny rezultat	Otrzymano oczekiwany rezultat

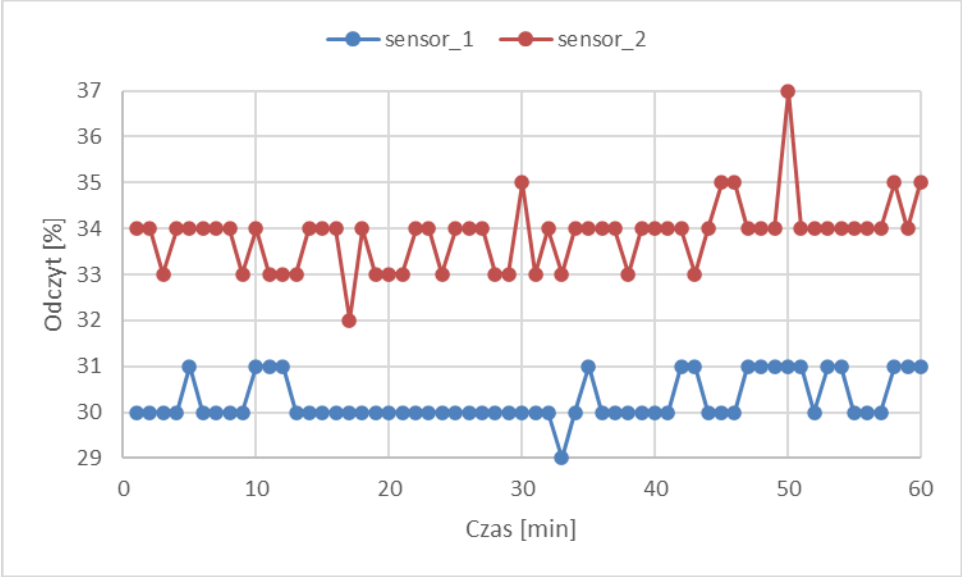
Przykłady żądań i JSONów znajdują się w pliku tests.http. Umożliwiają one przeprowadzenie testów jednostkowych dla poszczególnych funkcji i sprawdzenie ich poprawności dla różnych przypadków. Dzięki temu można zweryfikować, czy funkcje działają zgodnie z oczekiwaniami dla różnych zestawów danych wejściowych.

W ramach testowania aplikacji realizowane były też testy manualne. Sprawdzana była między innymi poprawność komunikacji między frontendem a Rest Api. Poprawność wyświetlania i odbierania danych, działania przycisków i ogólnego działania strony. W trakcie tych testów wykrytych zostało kilka błędów i możliwości usprawnienia działania względem wersji pierwotnej. Kwestie te udało się w aktualnej, finalnej wersji rozwiązać.

Testy działania

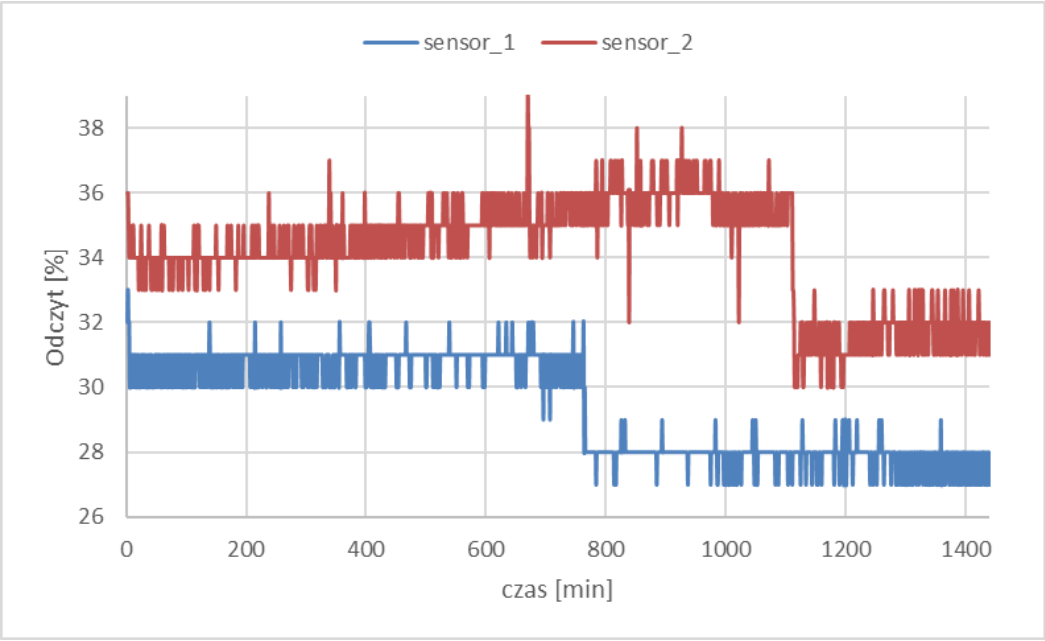
Przeprowadzono również testy weryfikujące czy układ działa poprawnie w środowisku docelowym. Na tą potrzebę czujniki umieszczono w doniczce z glebą, która uprzednio była lekko nasączona wodą. W doniczce nie znajdowały się rośliny, a sama doniczka znajdowała się w pomieszczeniu, w którym jest słabe nasłonecznienie. Badania przeprowadzono w dwóch fazach. Pierwsza z nich polegała na magazynowaniu danych przez godzinę i obserwacji jak zmienia się w tym czasie wilgotność gleby. Druga próba była analogiczna jednak próba trwała 24 h. Na potrzeby testów dane z sensorów odczytywane były co minutę w celu zaobserwowania czy występują szybkie zmiany, które wpłyną na właściwe ustawienia urządzenia.

Pierwszy wykres prezentuje wyniki dla testu, który trwał godzinę.



Rys.12 Wykres prezentujący pomiary dla godzinnego testu

Drugi wykres prezentuje wyniki dla testu, który trwał 24 h.



Rys.13 Wykres prezentujący pomiary dla 24 h testu

Na podstawie przeprowadzonych badań można zauważyć, że zmiany wilgotności gleby są niewielkie. Jednak podczas ostatecznego doboru właściwych ustawień urządzenia wzięto również pod uwagę tryb pracy w jakim jest urządzenie, ale również inne czynniki jakie mogą wpłynąć na wilgotność gleby, czyli wysoka temperatura i obecność roślin.

7. Podsumowanie

Stworzona płytka uniwersalna i program działają poprawnie ze względu na staranne wykonanie i wiele godzin pracy. Kod może zawierać niewielkie błędy które nie zostały wykryte podczas testowania układu. Układ daje możliwość dalszej rozbudowy. Po zakończeniu prac układ wystarczy uruchomić zgodnie z zamieszczonymi niżej informacjami. Kod jest poprawny ze względu na to, że spełnia nasze założenia projektowe i został zweryfikowany na podstawie przeprowadzonych testów.

8. Dalszy rozwój i ulepszenia

Poniżej przedstawione zostały możliwości dalszego rozwoju aplikacji:

- Dodanie możliwości dołączenia większej liczby czujników.
- Dodanie możliwości ustawienia tygodniowego harmonogramu nawadniania.
- Dodanie nowego widoku / zakładkę w aplikacji umożliwiającą monitorowanie statystyk nawadniania terenu w danym okresie.
- Dodanie możliwości wglądu do zużycia wody.
- Testy wydajnościowe.
- Naprawa błędu związanego z nieprawidłowym odczytem danych ze strony.

9. Instrukcja

Ogólne:

Urządzenia należy podłączyć do zasilania poprzez kabel USB. Następnie należy uruchomić odpowiednie aplikacje na komputerze dzięki czemu istnieje możliwość odczytu danych z urządzenia i sterowania nim.

Mikrokontroler:

Należy sklonować repozytorium, Następnie uruchomić Visual Studio Code. Jeśli nie posiadamy rozszerzenia ESP-IDF należy je pobrać. Następnie przechodzimy do ESP-IDF SDK Configuration Editor (menuconfig), gdzie podajemy nazwę i hasło do sieci z którą się będziemy łaczyć. Kod należy zbudować i wgrać na mikrokontroler. Należy pamiętać, że jeśli wgrywamy kod na płytke, która posiada servo mechanizm należy ustawić w kodzie flage BOARD na 0, dzieki temu dodane zostaną odpowiednie elementy zapewniające obsługe mechanizmu, natomiast gdy wgrywamy kod na urzadzenie które posiada drugi czujnik ustawiamy tą flagę na wartość 1. Po wgraniu kodu na urządzenia i odpowiednim podłączeniu peryferii możemy podłączyć urządzenie do zasilania.

Rest Api i Baza danych:

Na tym etapie, w pierwszej kolejności należy przygotować bazę danych. Powinna istnieć baza danych “swawapi”, gdzie użytkownik to “root”, hasło to “localhost”. Należy dodać tabele sensor_data, a w niej zaimplementować kolumny takie jak:

“humidity” - FLOAT (Not null),

“is_sensor_on” - TINYINT ,

“created_at” - TIMESTAMP (Default – CURRENT_TIMESTAMP),

“sensor_id” - TINYINT (Not null),

“sprinkle_state” - TINYINT.

Po przygotowaniu bazy, można zweryfikować czy dane do połączenia z bazą zgadzają się z wprowadzonymi danymi w pliku swaw_api.py w funkcji get_connection_(). Jeśli wszystko się zgadza, aplikacja może zostać uruchomiona. Aby uruchomić aplikacje z linii komend należy wpisać polecenie python swaw_api.py

Frontend:

Do w pełni poprawnego uruchomienia aplikacji wraz z frontendem na aktualnym etapie potrzebne jest zainstalowanie na komputerze środowiska Node.js (wersja 20.3.0), a następnie zainstalowanie w edytorze kodu Node Package Manager’a. W Visual Studio Code można to zrobić za pomocą komendy “npm install”. Następnie będąc w folderze, w którym zostały zapisane pliki z kodem należy wywołać w terminalu komendę “npm start” uruchamiającą aplikację. Należy też pamiętać, że do poprawnego działania konieczne jest by było uruchomione Rest Api.

Bibliografia

Servo Motor SG90 Data Sheet - Imperial College London, www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf. Accessed 18 June 2023.

"ESP32 Pinout Reference." *Last Minute Engineers*, 2 Mar. 2023, lastminuteengineers.com/esp32-pinout-reference/.

"Technical Documents: Espressif Systems." *Technical Documents | Espressif Systems*, www.espressif.com/en/support/documents/technical-documents. Accessed 18 June 2023.

"Esp32 Esp-IDF." *ESP32 ESP-IDF*, 8 Feb. 2023, esp32tutorials.com/.

Górecki, Piotr. "Czujniki Wilgotności Gleby – Dlaczego Są Tak Problematyczne?" *FORBOT*, 21 Feb. 2023, forbot.pl/blog/czujniki-wilgotnosci-gleby-dlaczego-sa-tak-problematyczne-id52948?fbclid=IwAR0QPhm9rb95N-NO6qstLOBIneAarod-2nHD0T_1Ur4opfj8_et-6C3wiiI.

Majewski, Paweł, et al. "Garden_system." *GitHub*, github.com/PawelMajew/garden_system.git. Accessed 18 June 2023.