Politechnika Śląska

Wydział Automatyki, Elektroniki i Informatyki

Uniwersalne Metody Projektowania Aplikacji na Urządzenia Mobilne i Wbudowane

Automatyczny system do nawadniania kwiatów

skład sekcji	Oliwia Gowor Przemysław Bryś
prowadzący	dr inż. Dariusz Marek
kierunek	Informatyka
wydział	AEI
semestr	6
grupa	ISMiP

Spis treści

1. Cel projektu

Celem projektu jest stworzenie systemu umożliwiającego automatyzację nawadniania kwiatów. Będzie on kontrolowany przez czujniki wilgotności gleby oraz natężenia światła.
W określonych warunkach ma uruchamiać się moduł nawadniający. Parametry systemu oraz komunikaty o warunkach panujących w pobliżu rośliny będą pokazywane za pomocą aplikacji Webowej, wysyłane w formie powiadomienia na pocztę e-mail oraz wyświetlane na wyświetlaczu dołączonym do układu. Jedyna wymagana ingerencja użytkownika ma polegać na uzupełnianiu zbiornika z wodą oraz ewentualnej zmianie domyślnych ustawień.

2. Wymagania funkcjonalne

Celem układu jest kontrolowanie wilgotności gleby w doniczce oraz natężenia światła w pobliżu rośliny za pomocą odpowiednich czujników. W przypadku, gdy wilgotność gleby spadnie poniżej określonego poziomu, uruchomiona zostanie mała pompka wodna. System zawiera także czujnik odległości. Umieszczony powinien być on nad zbiornikiem z wodą w celu określenia poziomu wody. Aktualne parametry oraz informacje o podlaniu rośliny wyświetlane są w aplikacji Webowej, na wyświetlaczu oraz wysyłane do użytkownika w postaci powiadomienia na skrzynkę mailową. Układ zbudowany jest na bazie mikroprocesora ESP32. Zasilany jest sieciowo, a w przypadku braku prądu, bateryjnie.

3. Porównanie elementów

3.1. Mikrokontroler

- Porównanie analizowanych elementów:

Model	**FireBeetle ESP32-E**	Raspberry Pi Pico W	Arduino Nano	Arduino Uno Rev3	Arduino Uno WiFi Rev2	ESP32-DevKitC
Układ	ESP-WROOM -32	RP2040	–	–	–	ESP-WROOM-32
Procesor	Tensilica LX6 dual-core	ARM	ATmega328	ATmega328	ATmega4809	Dual Core Tensilica LX6
WiFi	802.11 b/g/n/d/e/i/k/r	802.11 b/g/n	brak	brak	802.11b/g/n	802.11BGN HT40

Bluetooth	Bluetooth v4.2 / Bluetooth BLE	brak	brak	brak	Bluetooth v4.2 / Bluetooth BLE	Bluetooth v4.2 / Bluetooth BLE
Taktowanie procesora [MHz]	240	133	16	16	20	240
Pamięć Flash	4 MB	2MB	32 kB	32 kB	48 kB	4 MB
Pamięć SRAM [kB]	520	264	2	2	6	520
Piny analogowe [szt.]	5	4	8	6	6	2
Piny cyfrowe [szt.]	10	23	14	14	14	12
UART [szt.]	2	2	2	2	2	3
I2C [szt.]	2	2	2	2	2	2
SPI [szt.]	3	2	4	4	4	3
GDI [szt.]	1	brak	brak	brak	brak	brak
Napięcie zasilania [V]	3,3 / 5	5	6 – 20	6 – 20	6 – 20	5
Napięcie pracy [V]	3,3	3,3	5	5	5	3,3
Złącze	USB C	microUSB	miniUSB	DC	DC	microUSB
Zasilanie z akumulatora	Tak	Tak	Nie	Nie	Nie	Tak
Ładowanie akumulatora z USB	Tak	Nie	Nie	Nie	Nie	Nie
Temperatura robocza [°C]	-40 ~ 85	-20 ~ 70	-40 ~ 85	-40 ~ 85	-40 ~ 85	-40 ~ 125
Wymiary[mm]	25,4 x 60	51,3 x 21 x 3,9	45 x 18	68.6 x 53.4	69 x 53 x 14	55 x 28 x 8
Cena [zł]	62,90	35,90	129,00	119,00	249,00	49,90

Jednym z najważniejszych aspektów była dla nas obecność modułu WiFi ze względu na wysyłanie powiadomień poprzez e-mail. Niestety Arduino Nano oraz Arduino Uno Rev3 go nie posiadają, przez co zostały przez nas odrzucone. W przypadku Arduino Uno WiFi Rev2 parametry procesora wypadają gorzej w porównaniu z pozostałymi mikrokontrolerami. Na korzyść mikrokontrolera FireBeetle ESP32-E przemówiła także obecność modułu umożliwiającego bezpośrednie podłączenie ogniwa oraz jego ładowanie podczas, gdy mikrokontroler zasilany jest przez USB.

3.2. Wyświetlacz

Wyświetlacz nie jest bardzo wymagającym komponentem. Ze względu na niewielką ilość danych prezentowanych na wyświetlaczu zdecydowaliśmy się na popularny wyświetlacz Philips PCF8574 LCD 2x16 z konwerterem I2C LCM1602 ze strony Botland w cenie około 25 zł.

3.3. Ogniwo

Ze względu na małe zużycie prądu przez procesor oraz założenie, że zasilanie bateryjne ma służyć tylko w przypadkach przerw w dostawie prądu zdecydowaliśmy się na ogniwo 18650 Li-Ion XTAR o pojemności 2200 mAh i napięciu nominalnym 3,7V. Ten akumulator litowo-jonowy posiada zabezpieczenia przed przeładowaniem, nadmiernym rozładowaniem, przeciążeniem i zwarciem.

3.4. Czujnik wilgotności gleby

- Porównanie analizowanych elementów:

Model	**MOD-01588**	Grove - czujnik / sonda do pomiaru wilgotności gleby v1.4	Maker Soil Moisture - pojemnościowy czujnik wilgotności gleby	Iduino ME110
Napięcie zasilania [V]	3,3 – 5	3,3 – 5	2,5 – 7	3,3 – 5
Prąd zasilania [mA]	35	25	5,4 – 5,6	25
Sygnał wyjściowy	analogowo-cyfrowy	analogowy	analogowy	analogowy
Kompatybilność	Arduino	Arduino/ RaspberryPi	Arduino	Arduino/RaspberryPi
Zakres	0 – 950	0 – 950	0 – 950	0 – 750
Cena [zł] (przybliżona)	5,5	17	30	4,9

Czujnik wilgotności gleby MOD-01588 wybraliśmy ze względu na cenę, dostępność oraz większy zakres pomiarowy od czujnika Iduino ME110.

3.5. Czujnik natężenia światła

- Porównanie analizowanych elementów:

Model	BH1750	Czujnik światła cyfrowy z regulacją - fotorezystor + potencjometr	**Czujnik światła LDR rezystancyjny – Okystar**	Iduino SE012
Napięcie zasilania [V]	3 – 5	3,3 – 5	3,3 – 5	3,3 – 5
Interfejs komunikacyjny	I2C	cyfrowy	cyfrowo - analogowy	analogowy
Zakres pomiarowy [lx]	1 – 65535	1 – 65528	1 – 65528	-------
Temperatura robocza [°C]	-40 ~ 85	-30 ~ 70	-30 ~ 70	-40 ~ 85
Wymiary [mm]	21 x 16 x 3,3	32 x 17	32 x 14 x 5	20 x 17
Cena [zł] (przybliżona)	15,90	4,90	4,90	6,40

O naszym wyborze czujnika zadecydował rodzaj komunikacji z mikrokontrolerem. Ze względu na to, że chcemy odczytywać z czujnika dokładną wartość oraz dostępne rodzaje oraz ilość pinów w mikrokontrolerze, zdecydowaliśmy się na czujnik analogowo-cyfrowy Czujnik światła LDR rezystancyjny – Okystar. Atutem były także łatwa dostępność oraz cena czujnika.

3.6. Czujnik odległości

- Porównanie analizowanych elementów:

Model	VL53L0XV2	**HC-SR04**	HY-SRF05
Napięcie zasilania [V]	2.6 – 5.5	3.3 – 5	5
Rodzaj czujnika	laserowy pomiar czasu lotu (TOF)	ultradźwiękowy	odbiciowy czujnik podczerwieni
Rozdzielczość [mm]	1	3	3

Zakres pomiarowy [m]	do 2	0,02 - 4,5	0,02-4
Dokładność	± 3% w najlepszym przypadku do ± 10% przy mniej optymalnych warunkach	± 5%	± 3%
Temperatura pracy	-20 ~ 70	-20 ~ 60	-25 ~ 70
Wymiary [mm]	4.40 x 2.40 x 1.00	45 x 20 x 15	45x20
Cena [zł] (przybliżona)	6,66	3,37	13,9

Czujnik odległości HC-SR04 został przez nas wybrany ze względu na największy zakres pomiarowy oraz pewność, że czujnik ultradźwiękowy będzie poprawnie działał z wodą w zbiorniku.

3.7. Pompa wodna

Ze względu na potrzebną niewielką moc do podlewania roślin oraz łatwe sterowanie pompą za pomocą mikrokontrolera zdecydowaliśmy się na małą, zanurzalną pompkę wodną zasilaną napięciem 3V – 6V.

3.8. Tranzystory

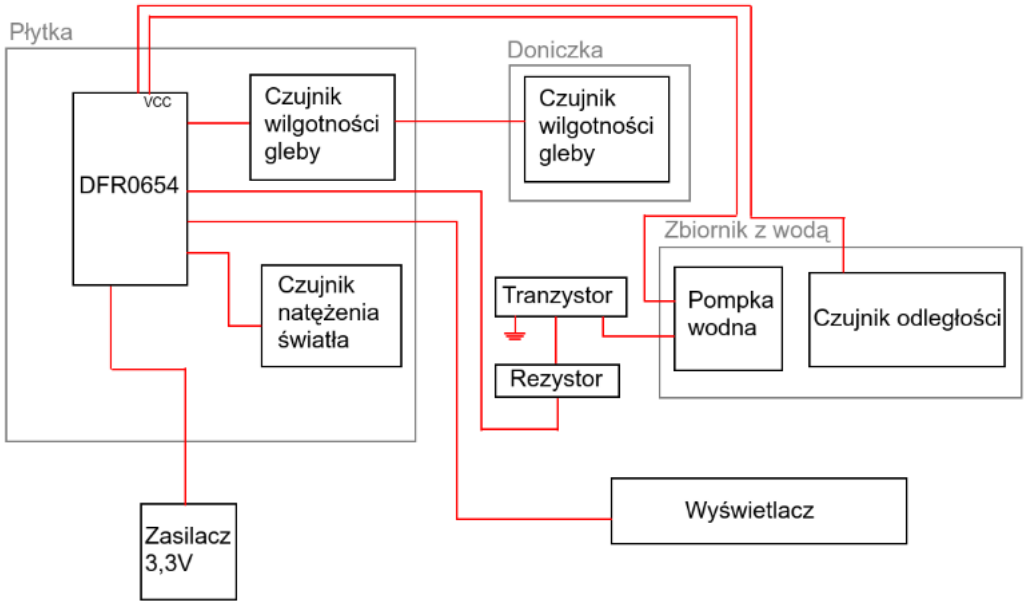
Ze względu na parametry mikrokontrolera oraz pompki wodnej wybraliśmy tranzystor bipolarny NPN 2N2222A, którego maksymalne napięcie z kolektora do emitera wynosi 75 V, a prąd kolektora to 600 mA. Moc pojedynczego tranzystora wynosi 625 mW.

3.9. Rezystory

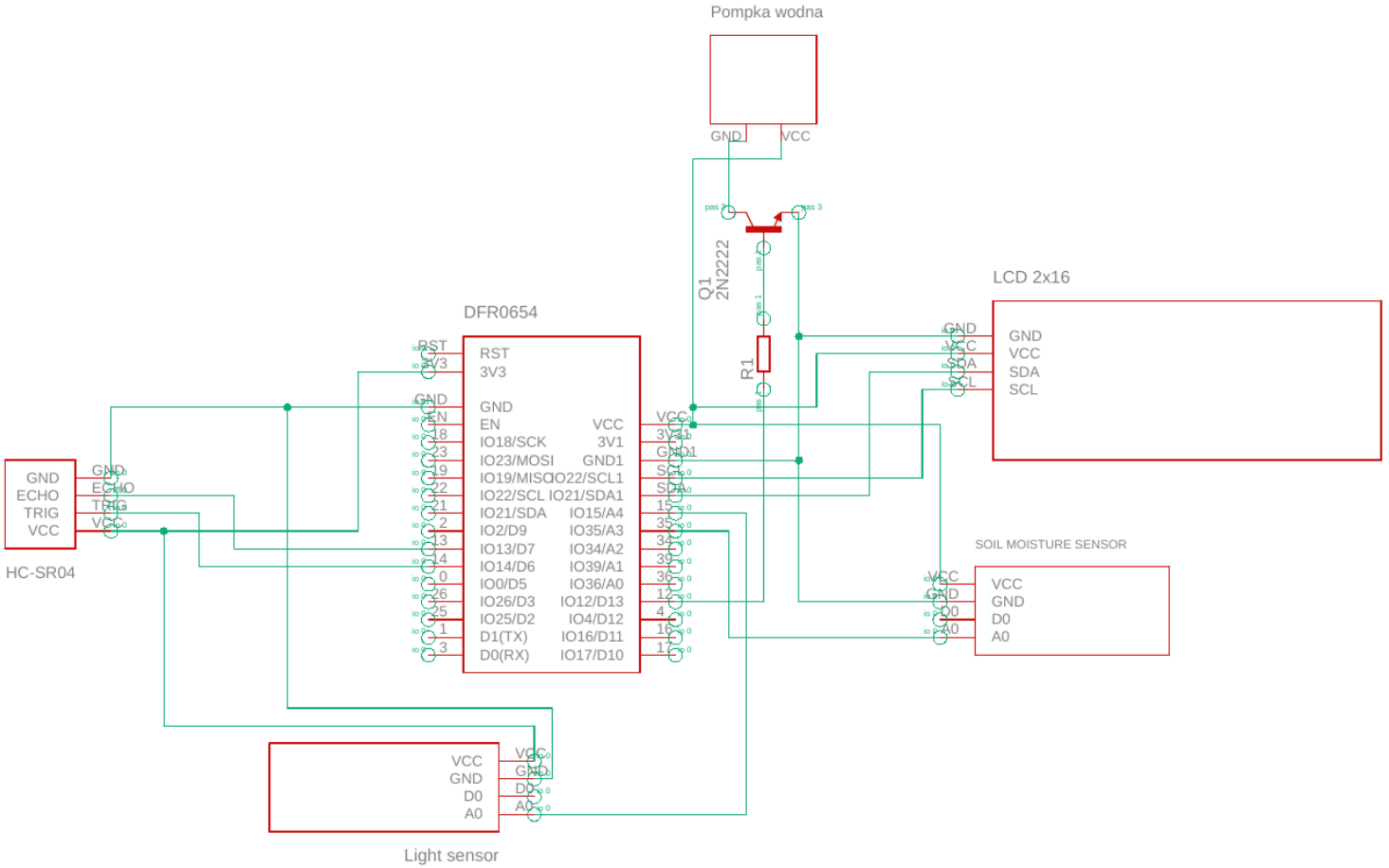
Wybrano zestaw rezystorów CF THT 1/4W. W skład zestawu wchodzi po 10 sztuk elementów o następujących rezystancjach:10 Ω, 22 Ω, 47 Ω, 100 Ω, 220 Ω, 470 Ω, 1 kΩ, 2.2 kΩ, 4.7 kΩ, 10 kΩ, 22 kΩ, 47 kΩ, 100 kΩ, 220 kΩ, 470 kΩ, 1 MΩ.

4. Schematy

4.1. Schemat blokowy układu:



4.2. Schemat (SCH) projektu:



5. Oprogramowanie

Kod programu napisano w języku C++ oraz przy pomocy HTML, CSS i Javascript w przypadku części Webowej i podzielony jest na 6 plików:

5.1. Gsender.h

Jest to plik nagłówkowy, który zawiera deklarację klasy Gsender.

5.2. Gsender.cpp

Ten kod definiuje klasę Gsender, która umożliwia wysyłanie wiadomości e-mail za pomocą protokołu SMTP. Oto najważniejsze fragmenty kodu:

- Metoda statyczna *Instance()* tworzy instancję klasy Gsender i zwraca wskaźnik do niej. Wykorzystuje wzorzec Singleton, aby upewnić się, że istnieje tylko jedna instancja klasy Gsender.
- Metoda *Subject(const char* subject)* ustawia temat wiadomości. Usuwa poprzedni temat (jeśli istniał) i alokuje pamięć dla nowego tematu.
- Metoda *AwaitSMTPResponse(WiFiClientSecure &client, const String &resp, uint16_t timeOut)* oczekuje na odpowiedź serwera SMTP. Sprawdza, czy klient otrzymuje odpowiedź od serwera w określonym czasie. Jeśli czas przekroczy limit, zwraca wartość false.

```
bool Gsender::AwaitSMTPResponse(WiFiClientSecure &client, const String &resp,
uint16_t timeOut)
{
  uint32_t ts = millis();
  while (!client.available())
  {
    if(millis() > (ts + timeOut)) {
      _error = "SMTP Response TIMEOUT!";
      return false;
    }
  }
  _serverResponce = client.readStringUntil('\n');
#if defined(GS_SERIAL_LOG_1) || defined(GS_SERIAL_LOG_2)
  Serial.println(_serverResponce);
#endif
  if (resp && _serverResponce.indexOf(resp) == -1) return false;
  return true;
}
```

- Metoda *getLastResponce()* zwraca ostatnią otrzymaną odpowiedź serwera SMTP.

- Metoda *getError()* zwraca bieżący błąd (jeśli wystąpił) jako wskaźnik na stałą tablicę znaków.
- Metoda *Send(const String &to, const String &message)* jest główną metodą do wysyłania wiadomości e-mail. Tworzy połączenie z serwerem SMTP, autoryzuje się, ustawia nagłówki wiadomości, wysyła treść wiadomości i zamyka połączenie.

```
bool Gsender::Send(const String &to, const String &message)
{
  WiFiClientSecure client;
  client.setInsecure();
#if defined(GS_SERIAL_LOG_2)
  Serial.print("Connecting to :");
  Serial.println(SMTP_SERVER);
#endif
  if(!client.connect(SMTP_SERVER, SMTP_PORT)) {
    _error = "Could not connect to mail server";
    return false;
  }
  if(!AwaitSMTPResponse(client, "220")) {
    _error = "Connection Error";
    return false;
  }

#if defined(GS_SERIAL_LOG_2)
  Serial.println("HELO friend:");
#endif
  client.println("HELO friend");
  if(!AwaitSMTPResponse(client, "250")){
    _error = "identification error";
    return false;
  }


#if defined(GS_SERIAL_LOG_2)
  Serial.println("AUTH LOGIN:");
#endif
  client.println("AUTH LOGIN");
  AwaitSMTPResponse(client);


Serial.println("hey2222");


#if defined(GS_SERIAL_LOG_2)
  Serial.println("EMAILBASE64_LOGIN:");
#endif
  client.println(EMAILBASE64_LOGIN);
  AwaitSMTPResponse(client);

#if defined(GS_SERIAL_LOG_2)
  Serial.println("EMAILBASE64_PASSWORD:");
#endif
  client.println(EMAILBASE64_PASSWORD);
  if (!AwaitSMTPResponse(client, "235")) {
```

```
    _error = "SMTP AUTH error";
    return false;
  }

  String mailFrom = "MAIL FROM: <" + String(FROM) + '>';
#if defined(GS_SERIAL_LOG_2)
  Serial.println(mailFrom);
#endif
  client.println(mailFrom);
  AwaitSMTPResponse(client);

  String rcpt = "RCPT TO: <" + to + '>';
#if defined(GS_SERIAL_LOG_2)
  Serial.println(rcpt);
#endif
  client.println(rcpt);
  AwaitSMTPResponse(client);

#if defined(GS_SERIAL_LOG_2)
  Serial.println("DATA:");
#endif
  client.println("DATA");
  if(!AwaitSMTPResponse(client, "354")) {
    _error = "SMTP DATA error";
    return false;
  }

  client.println("From: <" + String(FROM) + '>');
  client.println("To: <" + to + '>');

  client.print("Subject: ");
  client.println(_subject);

  client.println("Mime-Version: 1.0");
  client.println("Content-Type: text/html; charset=\"UTF-8\"");
  client.println("Content-Transfer-Encoding: 7bit");
  client.println();
  String body = "<!DOCTYPE html><html lang=\"en\">" + message + "</html>";
  client.println(body);
  client.println(".");
  if (!AwaitSMTPResponse(client, "250")) {
    _error = "Sending message error";
    return false;
  }
  client.println("QUIT");
  if (!AwaitSMTPResponse(client, "221")) {
    _error = "SMTP QUIT error";
    return false;
  }
  return true;
}
```

Kod wykorzystuje również dyrektywy preprocesora, takie jak #ifdef, #if defined, aby włączyć lub wyłączyć logowanie na Serial Monitor w zależności od zdefiniowanych makr, takich jak

GS_SERIAL_LOG_1 i GS_SERIAL_LOG_2. W logach wyświetlane są informacje dotyczące połączenia z serwerem SMTP oraz wysyłanych i otrzymywanych odpowiedzi.

5.3. Message.h

Jest to plik nagłówkowy, który zawiera deklaracje funkcji *sendInfo()* i *sendInfoPump()*.

5.4. Message.cpp

Ten kod zawiera dwie funkcje, *sendInfo()* i *sendInfoPump()*, które wysyłają informacje przez e-mail. Oto opis najważniejszych fragmentów kodu:

- Funkcja *sendInfo()* przyjmuje jako parametry adres e-mail, wilgotność, poziom oświetlenia i odległość od czujnika. Tworzony jest ciąg znaków message, do którego dodawane są informacje o wilgotności, poziomie oświetlenia i odległości. Następnie tworzony jest obiekt klasy Gsender i wywoływana jest funkcja *Send()*, która wysyła wiadomość e-mail z danymi.
- Funkcja *sendInfoPump()* przyjmuje jako parametry adres e-mail, stan pompy wodnej (włączona/wyłączona) i wilgotność. Podobnie jak w funkcji *sendInfo()*, tworzony jest ciąg znaków message, do którego dodawane są informacje o stanie pompy wodnej i wilgotności. Następnie tworzony jest obiekt klasy Gsender i wywoływana jest funkcja *Send()*, która wysyła wiadomość e-mail z danymi.

Obie funkcje korzystają z klasy Gsender, której instancja jest pobierana za pomocą metody *Instance()*. Klasa Gsender prawdopodobnie implementuje funkcjonalność wysyłania wiadomości e-mail.

5.5. index.h

Jest to plik nagłówkowy zawierający deklarację strony internetowej, napisanej przy pomocy HTML, CSS oraz JavaScript jako tablica znaków typu char. Zawiera treść strony internetowej w formie tzw. "raw literal" (znak *R"rawliteral(`* na początku i *`)rawliteral"* na końcu). Dzięki temu wszystkie znaki wewnątrz ciągu są traktowane dosłownie, bez potrzeby ucieczek (backslashes) dla znaków specjalnych.

Istnieje kilka funkcji JavaScript zdefiniowanych w tagu <script>, które wykonują się po stronie przeglądarki:

- t*ogglePump(element)* oraz *toggleAwayMode(element)* to funkcje, które wykorzystują obiekt XMLHttpRequest do wysłania żądania HTTP GET na odpowiednie ścieżki (/pump?state=1 lub /pump?state=0, /awayMode?state=1 lub /awayMode?state=0) w zależności od stanu przekazanego elementu (np. checkboxa). Wysyłając takie żądania, można włączać i wyłączać pompę lub tryb "away mode".
- *setAwayModeInterval()* oraz *setMinHumidity()* funkcje te wykorzystują XMLHttpRequest do wysłania żądania HTTP GET na odpowiednią ścieżkę (/setHumidity?value=<wartość>) z wartością odczytaną z pewnych elementów na stronie. W ten sposób można ustawić pewne parametry, takie jak interwał trybu "away mode" czy minimalną wilgotność.
- Trzy interwały (*setInterval(...)*) to fragmenty kodu JavaScript, które regularnie (co 2000 ms) wykonują żądania HTTP GET na różne ścieżki (/lightLvl, /humidity, /distance) i aktualizują zawartość odpowiednich elementów na stronie, takich jak poziom światła, wilgotność czy odległość.

5.6. smartPot.ino

Ten kod jest głównym programem dla mikrokontrolera ESP. Oto najważniejsze fragmenty kodu:

- Zdefiniowana stała SOUND_SPEED reprezentuje prędkość dźwięku w centymetrach na mikrosekundę.
- Zdefiniowane zmienne globalne, takie jak *ssid* i *password*, które przechowują nazwę sieci WiFi i hasło, *connection_state* przechowuje stan połączenia, a *reconnect_interval* określa czas oczekiwania przed kolejną próbą połączenia z siecią WiFi. Innymi zmiennymi globalnymi są prędkość dźwięku (*SOUND_SPEED*), parametry wejściowe (*PARAM_INPUT_VALUE*, *PARAM_INPUT_STATE*), minimalna wilgotność (*minHumidity*), czas trwania (w mikrosekundach) sygnału ultradźwiękowego w czujniku odległości (*duration*), interwały czasowe w programie (*counter*) zmienna (*awayMode*), która określa, czy tryb "away mode" jest aktywowany, interwał (w milisekundach) dla trybu "away mode" (*awayModeInterval*)
- Zainicjalizowany obiekt klasy *LiquidCrystal_I2C* do obsługi wyświetlacza LCD.
- Zainicjalizowany obiekt klasy *AsyncWebServer* o nazwie *server* nasłuchujący na porcie 80, co umożliwia obsługę żądań HTTP od przeglądarek internetowych.

- Funkcja *checkHumidity()* sprawdza poziom wilgotności. Jeśli jest zbyt niski, włącza pompę wodną i wyświetla informacje na wyświetlaczu LCD. Co jakiś czas wysyła również informację o pompie wodnej.
- Funkcja *getLight()* odczytuje poziom oświetlenia i zwraca odpowiednią wartość (ciemno, średnio, jasno).
- Funkcja *getDistance()* mierzy odległość przy użyciu czujników ultradźwiękowych i zwraca wynik w centymetrach.
- Funkcja *displayLight()* wyświetla poziom oświetlenia na wyświetlaczu LCD i zwraca wartość jako ciąg znaków.
- Funkcja *processor()* zastępuje miejsca zastępcze w kodzie HTML strony internetowej. Wstawia dane dynamiczne, takie jak stan trybu "away mode", poziom światła, wilgotność, odległość, stan pompki wodnej czy aktualny minimalny poziom wilgotności.
- Funkcja *handleAwayMode()* obsługuje tryb "away mode" poprzez wysyłanie powiadomień na podstawie ustawionego interwału.
- Funkcja *WiFiConnect()* nawiązuje połączenie z siecią WiFi na podstawie podanych nazwy SSID i hasła. Jeśli nie zostaną podane, używa domyślnych wartości.
- Funkcja *Awaits()* oczekuje na połączenie z siecią WiFi, jeśli nie jest już nawiązane, w pętli próbuje ponownie połączyć się co pewien czas.
- W funkcji *setup()* inicjalizowane są różne ustawienia, takie jak komunikacja szeregowa, piny wejściowe/wyjściowe, połączenie WiFi. Następnie tworzony jest obiekt Gsender (klasy do wysyłania wiadomości e-mail) i wysyłana jest wiadomość testowa. Konfiguruje ona także endpointy obsługujące różne żądania HTTP.
- W funkcji *loop()* odbywa się główna pętla programu. Sprawdzane są parametry roślin (wilgotność, poziom oświetlenia, odległość od czujnika) i wyświetlane na wyświetlaczu LCD. Odbywa się także obsługa trybu “away mode”.

```
void loop() {
  checkHumidity();

  //First part of informations displayed on LCD
  lcd.setCursor(0, 0);
  lcd.print("Swiatlo:");
  lcd.setCursor(8, 0);
  String lightLvl = getLight();
  lcd.print(lightLvl);

  lcd.setCursor(0, 1);
  lcd.print("Wilgotnosc:");
  lcd.setCursor(11, 1);
  float humidity = ((4095 - analogRead(A2)) / 28);
  lcd.print(humidity);
```

```
  delay(2000);

  //Second part of informations displayed on LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Woda:");
  lcd.setCursor(5, 0);
  float distanceCm = getDistance();
  lcd.print(distanceCm);
  lcd.setCursor(11, 0);
  lcd.print("cm");
  lcd.setCursor(0, 1);
  lcd.print("(od czujnika)");

  handleAwayMode(humidity, lightLvl, distanceCm);

  delay(2000);
}
```

Kod korzysta również z funkcji pomocniczych, takich jak *sendInfoPump()* i *sendInfo()*, które wysyłają wiadomości e-mail z informacjami o pompie wodnej oraz parametrach roślin.

6. Obsługa

6.1. Urządzenie

System jest prosty w obsłudze i może pracować bez większej ingerencji ze strony użytkownika przez okres czasu ograniczony przez pojemność używanego naczynia z wodą oraz, w przypadku braku prądu, pojemność i poziom naładowania ogniwa.
Aby układ działał prawidłowo, należy upewnić się czy wszystkie części są podłączone prawidłowo. Następnie czujnik wilgotności oraz silikonową rurkę od pompki należy wbić do gleby w doniczce z rośliną, a samą pompkę zanurzyć w naczyniu z wodą do podlewania kwiatów. Bezpośrednio nad naczyniem z wodą należy zamontować czujnik odległości. Czujnik natężenia światła ułożyć w takiej pozycji, aby bez zakłóceń monitorował stopień nasłonecznienia rośliny (nie układać pod przedmiotami, które zaburzają dostęp światła do czujnika np. pod parapetem). W celu uzyskania dostępu do aplikacji webowej, pozwalającej zdalnie sterować urządzeniem oraz w celu otrzymywania powiadomień e-mail niezbędne jest także przygotowanie urządzenia zapewniającego połączenie internetowe poprzez Wi-Fi. Nazwę sieci, hasło oraz dane skrzynki mailowej należy uzupełnić w kodzie programu. Następnie należy podłączyć urządzenie do prądu oraz koszyka z ogniwem. Jeśli wszystko zostało podłączone prawidłowo zaświeci się wyświetlacz LCD, pokazujący najpierw stan połączenia z internetem oraz adres IP, który należy wpisać w przeglądarce internetowej, a następnie aktualne parametry. Na e-mail zostanie wysłane powiadomienie o uruchomieniu urządzenia, a następnie co ustalony czas wysyłane będą powiadomienia z aktualnymi parametrami lub powiadomienia o włączeniu pompy wodnej.

6.2. Aplikacja Webowa

Aplikacja Webowa umożliwia sterowanie urządzeniem oraz monitorowanie jego aktualnego stanu z dowolnej przeglądarki internetowej. Warunkiem działania tej funkcji jest podłączenie urządzenia do tej samej sieci Wi-Fi co mikroprocesor. Adresem storny jest IP urządzenia, wyświetlane na ekranie LCD podczas jego uruchamiania.

Za pomocą aplikacji użytkownik może włączyć lub wyłączyć “tryb wyjazdowy (“away mode”), który odpowiada za wysyłanie powiadomień o aktualnym stanie rośliny za pomocą wiadomości e-mail. Istnieje możliwość ustawienia ich częstotliwości. Kolejną funkcją jest możliwość manualnego włączania oraz wyłączania pompki wodnej. Aby przełączyć się na manualne sterowanie pompką wodna należy ustawić minimalny pozom wilgotności na 0. W aplikacji wyświetlane parametry czujników w czasie rzeczywistym. Został zaimplementowany suwak do ustawiania minimalnego poziomu wilgotności gleby w doniczce, przy którym uruchamia się moduł nawadniający.

7. Bibliografia

- https://github.com/montotof123/Esp8266-12/tree/master
- https://lastminuteengineers.com/i2c-lcd-arduino-tutorial/
- https://howtomechatronics.com/tutorials/arduino/ultrasonic-sensor-hc-sr04/
- https://botland.com.pl/content/150-arduino-w-polaczeniu-z-czujnikiem-wilgotnosci
- https://randomnerdtutorials.com/esp32-async-web-server-espasyncwebserver-library/
- https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-web-server-arduino-ide/
- https://randomnerdtutorials.com/esp8266-esp-now-wi-fi-web-server/
- https://randomnerdtutorials.com/esp32-esp8266-input-data-html-form/?fbclid=IwAR0q15v_AwVq77jp5DJo3nrgi6txMUTVXtYtwVLlIQIwaE7LebCDNULapHI