**System automatyki szklarni z wykorzystaniem sieci bezprzewodowej**

1. Wstęp
2. Cel i zakres pracy
3. Mikroklimat
4. Przegląd rynku rozwiązań komercyjnych
   1. Rozwiązania dla rolników
   2. Rozwiązania hobbystyczne
5. Technologie wykorzystywane w czujnikach
   1. Przegląd układów mikroprocesorowych
   2. Przegląd zastosowanych technologii bezprzewodowych
   3. Przegląd czujników
6. Budowa systemu
   1. Wprowadzenie
   2. Założenia projektu
   3. Część hardware-owa projektu
      1. wprowadzenie
      2. Mikrokontroler
      3. Czujniki
      4. pytka
   4. Część software-owa projektu
      1. Wprowadzenie
      2. Omówienie komunikacji na linii mikrokontroler - serwer
      3. Oprogramowanie mikrokontrolera
         1. Założenia
         2. Schematy blokowe
         3. Omówienie działania programu
      4. Oprogramowanie serwerowe
         1. Założenia
         2. Schematy blokowe
         3. Omówienie działania programu
7. Efekt badania prototypu urządzenia
8. Podsumowanie i wnioski
9. Literatura
10. Streszczenie
11. Abstract

**Wstęp**

**Cel i zakres pracy**

Celem pracy jest zaprojektowanie, wykonanie prototypu oraz oprogramowanie systemu monitorowania warunków panujących w szklarni (wilgotności powietrza, temperatury oraz wilgotności gleby). Urządzenie będzie umożliwiało zdalne monitorowanie parametrów przy pomocy interfejsu przeglądarkowego. Projekt ma powstać z wykorzystaniem systemu mikroprocesorowego z interfejsem bezprzewodowym oraz dedykowanych czujników na zaprojektowanym obwodzie PCB.

Zakres prac:

* Zbudowanie i oprogramowanie prototypu urządzenia pomiarowego bazującego na mikrokontrolerze ESP32, czujniku temperatury i wilgotności powietrza SHT30 oraz pojemnościowym czujniku wilgotności gleby
* Dobór protokołu komunikacyjnego zapewniającego stabilną wymianę informacji pomiędzy rozproszonymi urządzeniami pomiarowymi a serwerem
* Stworzenie aplikacji serwerowej odpowiadającej za zbieranie i przetwarzanie informacji z urządzeń pomiarowych
* Zaimplementowanie prostego interfejsu pozwalającego na odczyt aktualnych wartości przekazanych przez urządzenia pomiarowe oraz dostęp do danych archiwalnych

**Przegląd rynku rozwiązań komercyjnych**

**Technologie wykorzystywane w czujnikach**

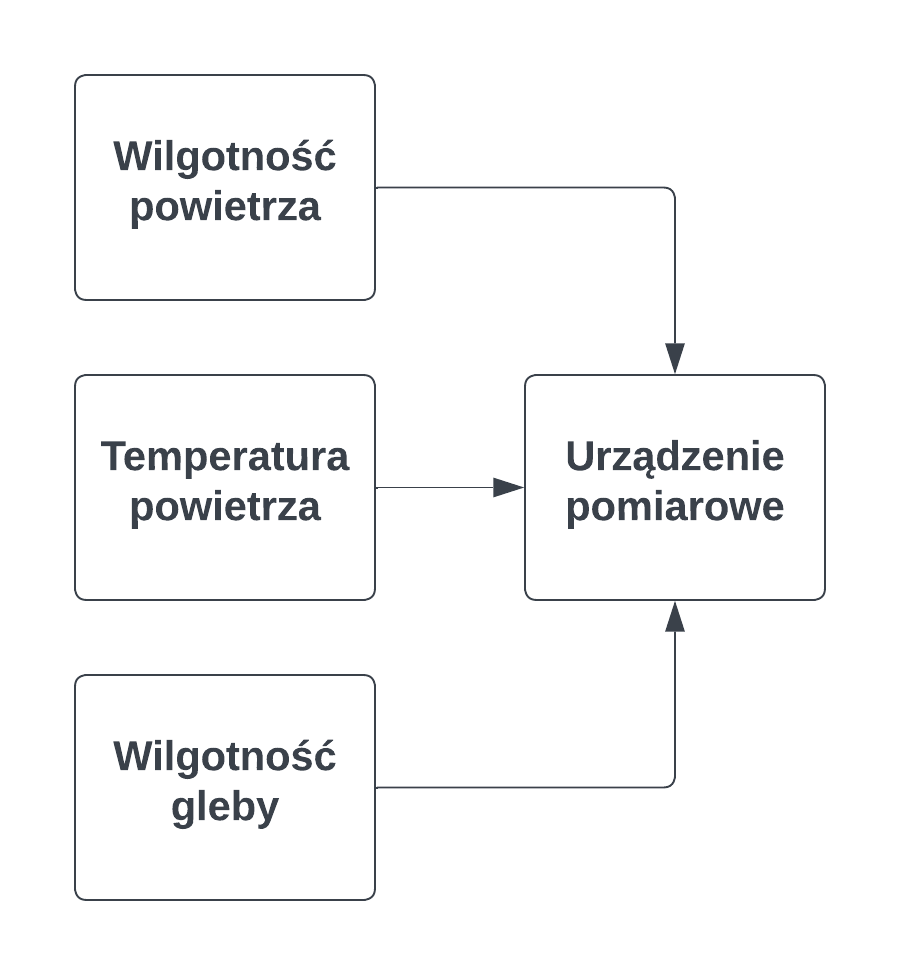
**Budowa systemu**

**Wprowadzenie**

**Założenia projektu**

**HW**

Częścią sprzętową projektu jest stworzenie urządzenia pomiarowego pozwalającego na wykonywanie zdalnych pomiarów temperatury i wilgotności powietrza oraz wilgotności gleby w szklarni. W tej sekcji pracy opisane zostały zastosowane rozwiązania sprzętowe realizujące w praktyce założenia ze schematu ideowego.



**Mirkokontroler**

W projekcie został zastosowany mikrokontroler ESP32 (ESP-WROOM-32). Jest to moduł wyposażony w dwurdzeniowy mikroprocesor Tensilica LX6 240 Mhz, 520KB SRAM, 4MB pamięci flash oraz obsługę sieci Wi-Fi oraz Bluetooth.

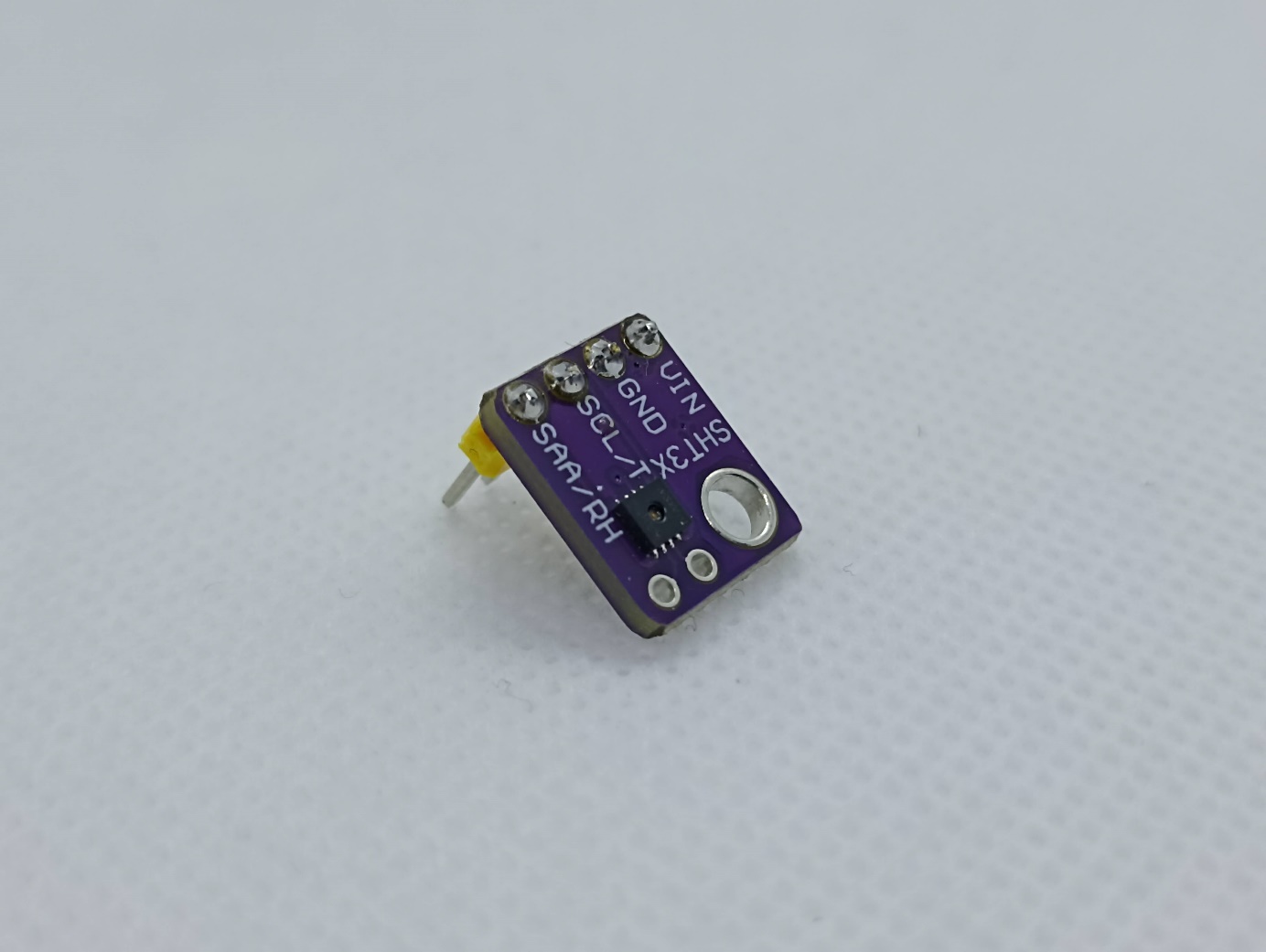


Parametry jakimi cechuje się wybrany mikrokontroler są tylko częściowo wykorzystywane w projekcie. Wybór modułu został podyktowany przede wszystkim jego szeroką dostępnością, stosunkowo niską ceną, wbudowaną obsługą sieci Wi-Fi oraz wsparciem dla frameworka Arduino.

**Czujniki**

Głównym założeniem projektu jest automatyzacja pomiarów wilgotności i temperatury powietrza oraz wilgotności gleby w szklarni. Aby umożliwić mikrokontrolerowi zbieranie tych danych zostały zastosowane poniższe czujniki:

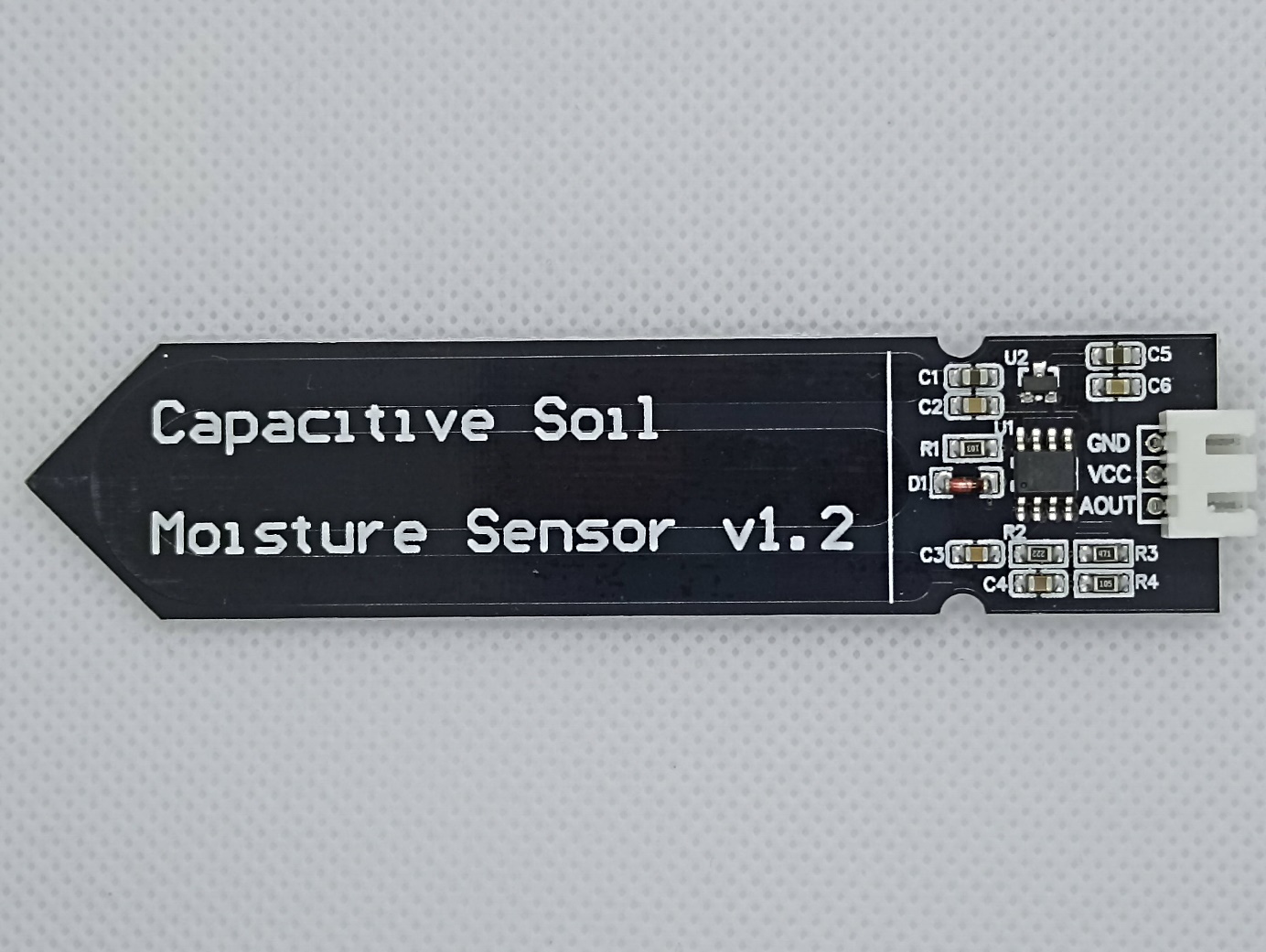
* SHT30 – pomiar wilgotności i temperatury powietrza
* Pojemnościowy czujnik wilgotności gleby



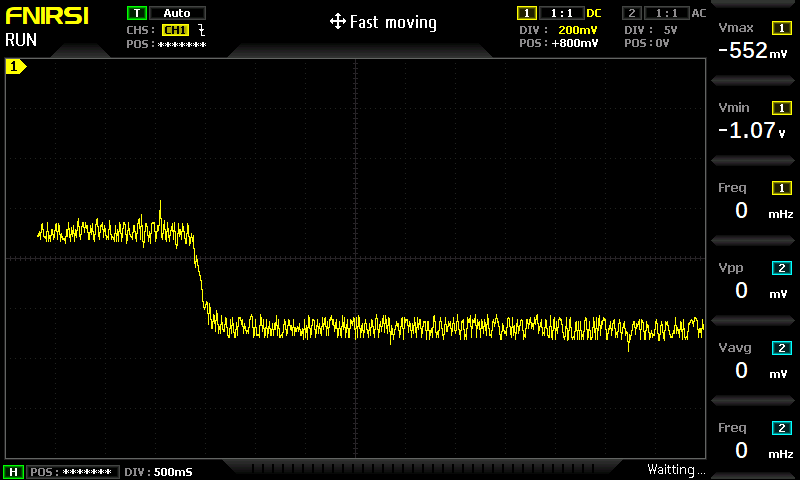
(**DO WERYFIKACJI**) SHT30 jest sensorem o dokładności rzędu 0.2°C w pomiarze temperatury oraz 2 punkty procentowe w pomiarze wilgotności. Obsługuje zakres [T]0-65°C i [H]10-90%. Jest to wystarczająca dokładność jak i rozpiętość pomiarów w tym zastosowaniu.

Sensor podłączany jest do mikrokontrolera za pomocą magistrali I2C. W zastosowanym układzie linia danych (SDL) podłączona jest do pinu D21, a linia zegara (SCL) do pinu D22.

Zastosowanie czujnika z komunikacją opartą na magistrali I2C pozwala na ewentualne rozszerzenie urządzenia pomiarowego o dodatkowy sensor, dla zwiększenia niezawodności pomiarów przy braku konieczności wykorzystywania dodatkowych złącz GPIO mikrokontrolera. Jest to możliwe, ponieważ moduł mimo domyślnie ustawionego adresu I2C na 0x44 pozwala go zmienić na 0x45 po fizycznej zmianie konfiguracji przez zwarcie odpowiednich pinów.

****

Czujnik wilgotności gleby jest prostym urządzeniem, którego działanie odbywa się na zasadzie pomiaru zmiany pojemności medium w którym się znajduje. Odczyty realizowane są przez interpretacje wartości na wyjściu analogowym sensora.



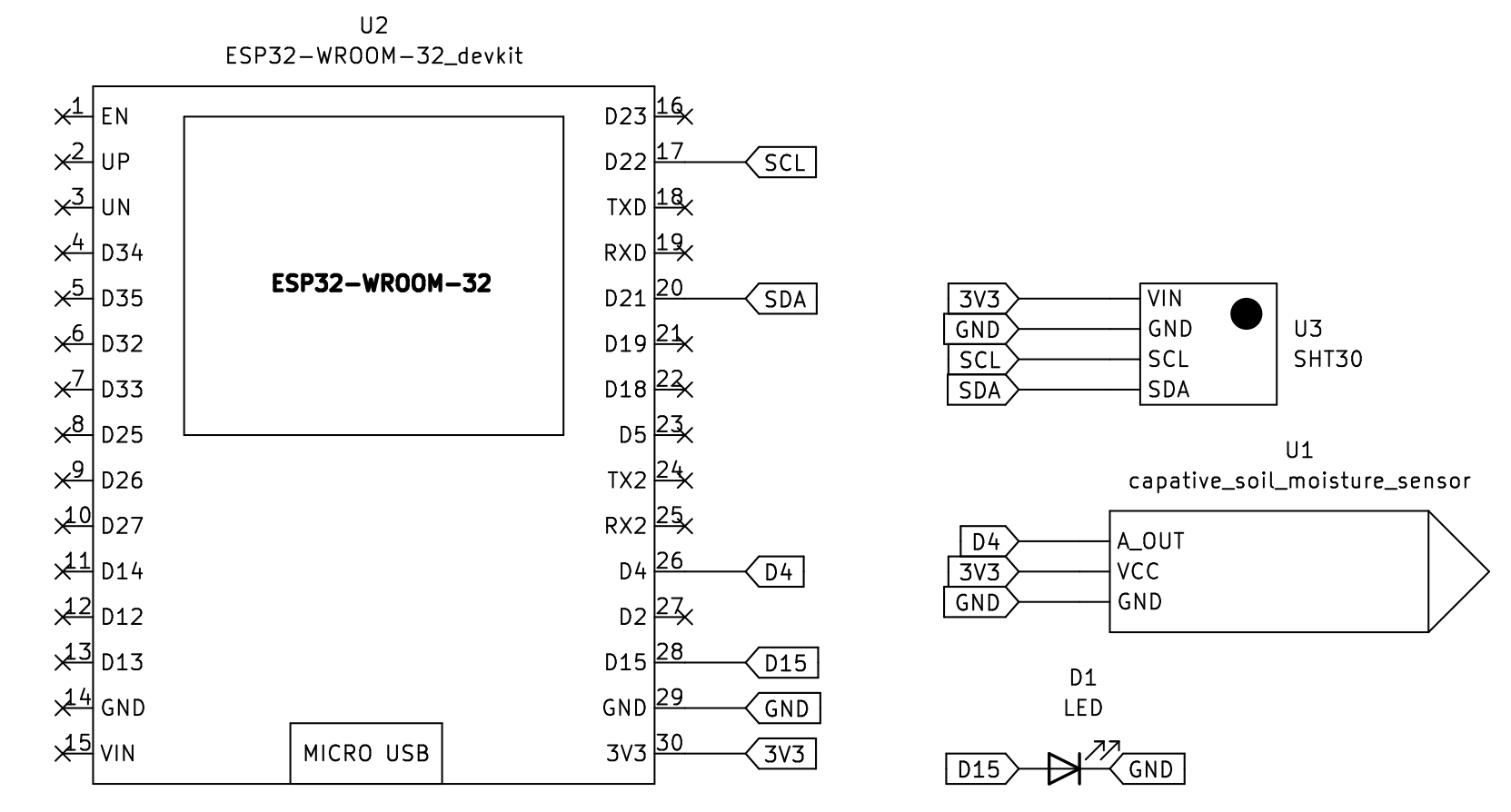
Im wyższa wilgotność badanego medium, tym niższa wartość sygnału na wyjściu czujnika.

Brak wiarygodnej dokumentacji i jakiegokolwiek wzorcowania czujnika powoduje, że uzyskane w ten sposób pomiary są w najlepszym wypadku poglądowe. Procentowe wartości wilgotności gleby obliczane przez program obsługujący urządzenie pomiarowe należy traktować jedynie jako sugestię i w przypadku faktycznego stosowania przeprowadzić testy, które pozwolą stwierdzić, jakie wartość wskazana przez czujnik będzie odpowiednia dla prowadzonych upraw.

Zakres wartości sygnału został oszacowany poprzez zrobienie testu, w którym suchy czujnik został umieszczony w wodzie. Uzyskany w ten sposób zakres został przeliczony na wartości od ~0% do 100%. Aby przeprowadzić sensowną kalibrację odczytów czujnika dobrze by było przeprowadzić serie pomiarów w komorze klimatycznej o zmiennej wilgotności powietrza.

**Schemat**

Poniżej schemat rzeczywistych połączeń w projekcie.



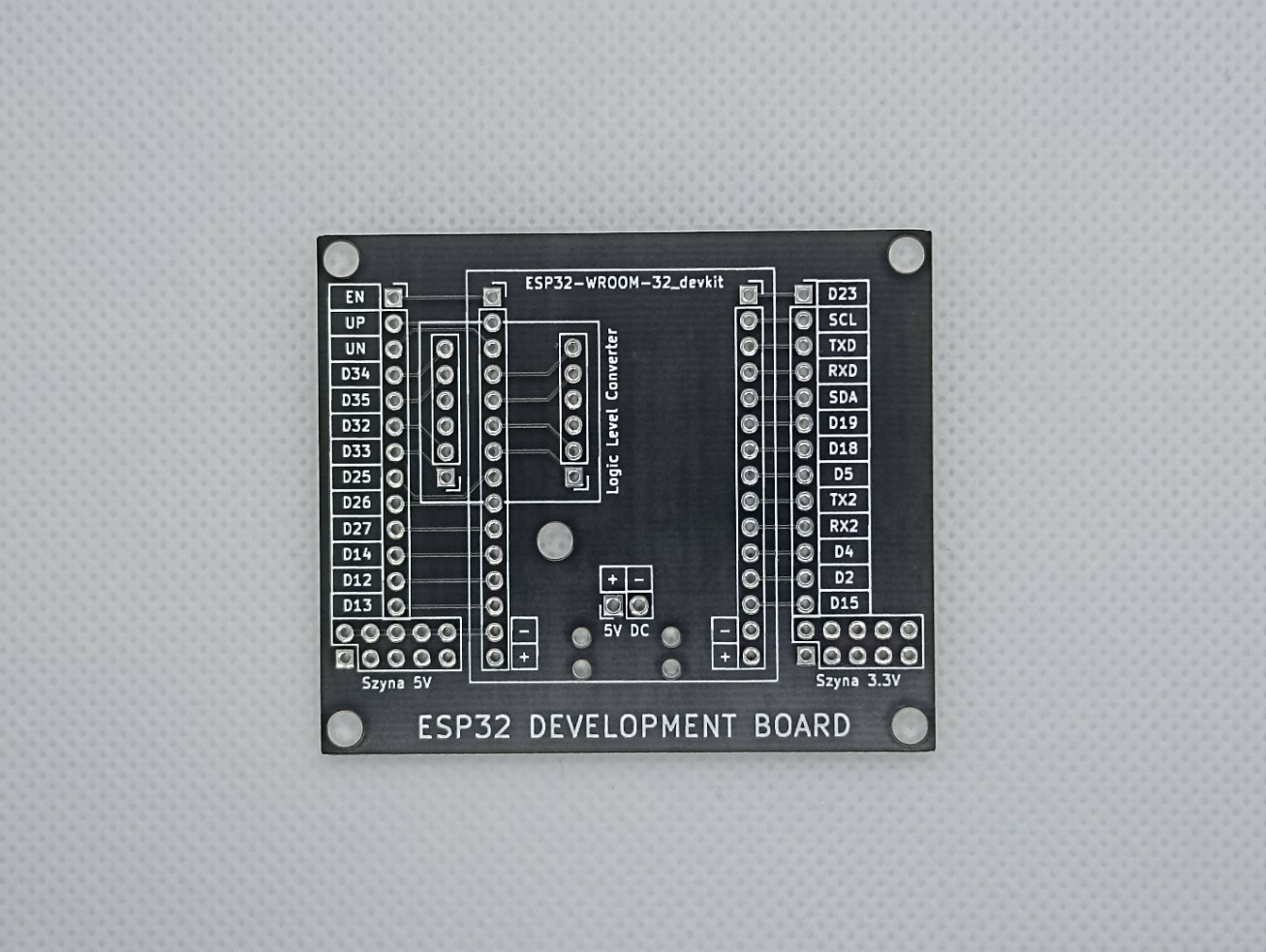
**Płytka**

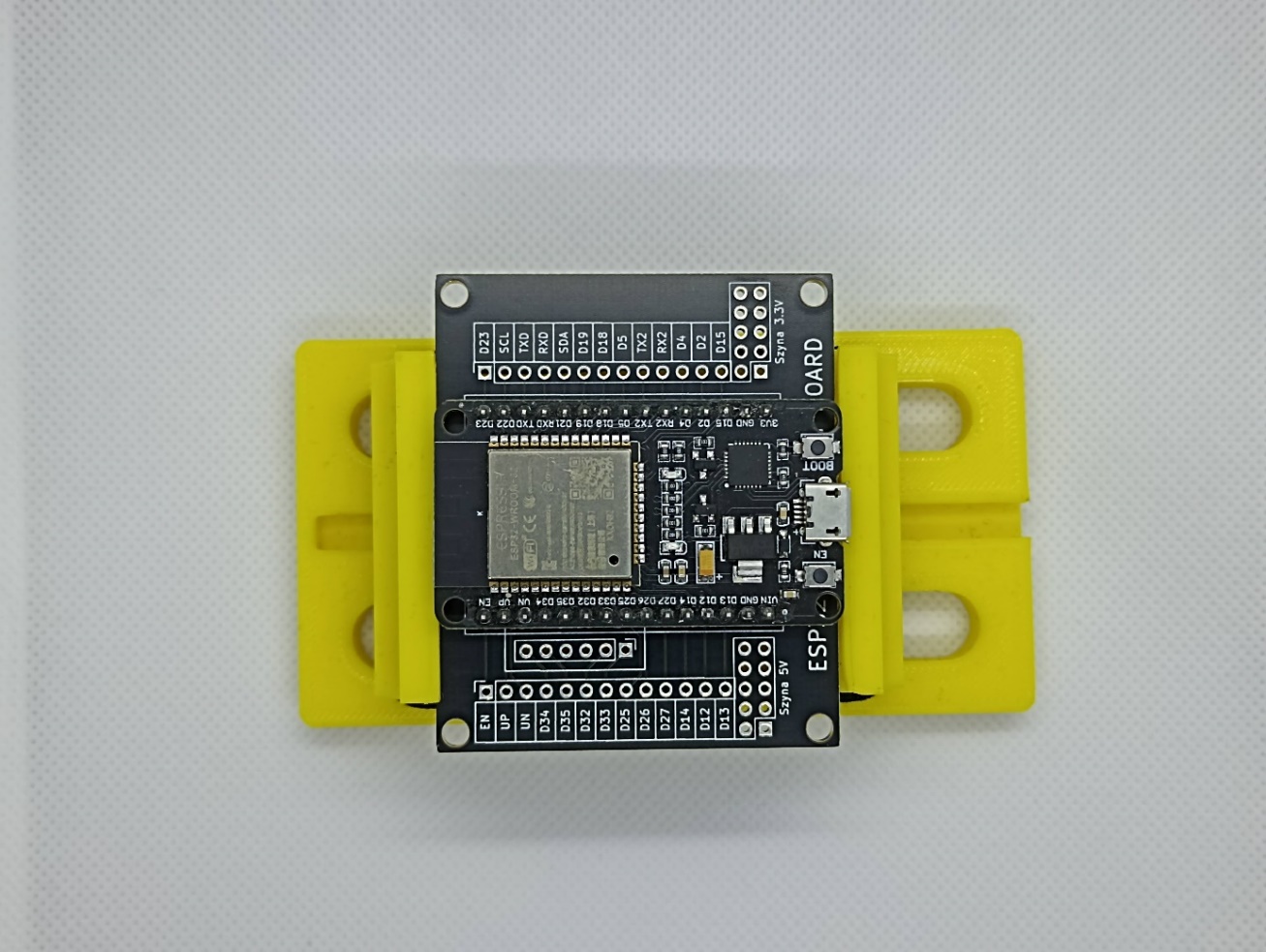
W projekcie zastosowana została płytka PCB – nie jest ona dedykowana do urządzenia pomiarowego. Ze względów ekonomicznych została zaprojektowana płytka uniwersalna z możliwością zastosowania konwertera stanów logicznych do zamiany stosowanej logiki 3V3 na 5V.

Płytka została zaprojektowana od podstaw w programie KiCad a następnie jej wykonanie zostało zlecone

Cała część elektroniczna została umieszczona na płytce PCB. Nie jest ona dedykowana do urządzenia pomiarowego – jest konstrukcją uniwersalną zaprojektowaną od podstaw w programie KiCad. Zastosowanie takiego rozwiązania zostało podyktowane ograniczonym czasem wykonania pracy – stworzenie optymalnego obwodu drukowanego mogłoby wymagać kilku rewizji projektu, co przy okresie oczekiwania na zrealizowanie zlecenia oscylującym w granicy 6 tygodni byłoby bardzo czasochłonne. Dodatkowo przy minimalnej ilości zamówionych płytek wynoszącej 5 sztuk wygenerowałoby dużo zbędnych odpadów. Płytka uniwersalna pozwoli na ponowne wykorzystanie konstrukcji w projektach opartych na mikrokontrolerze ESP32.



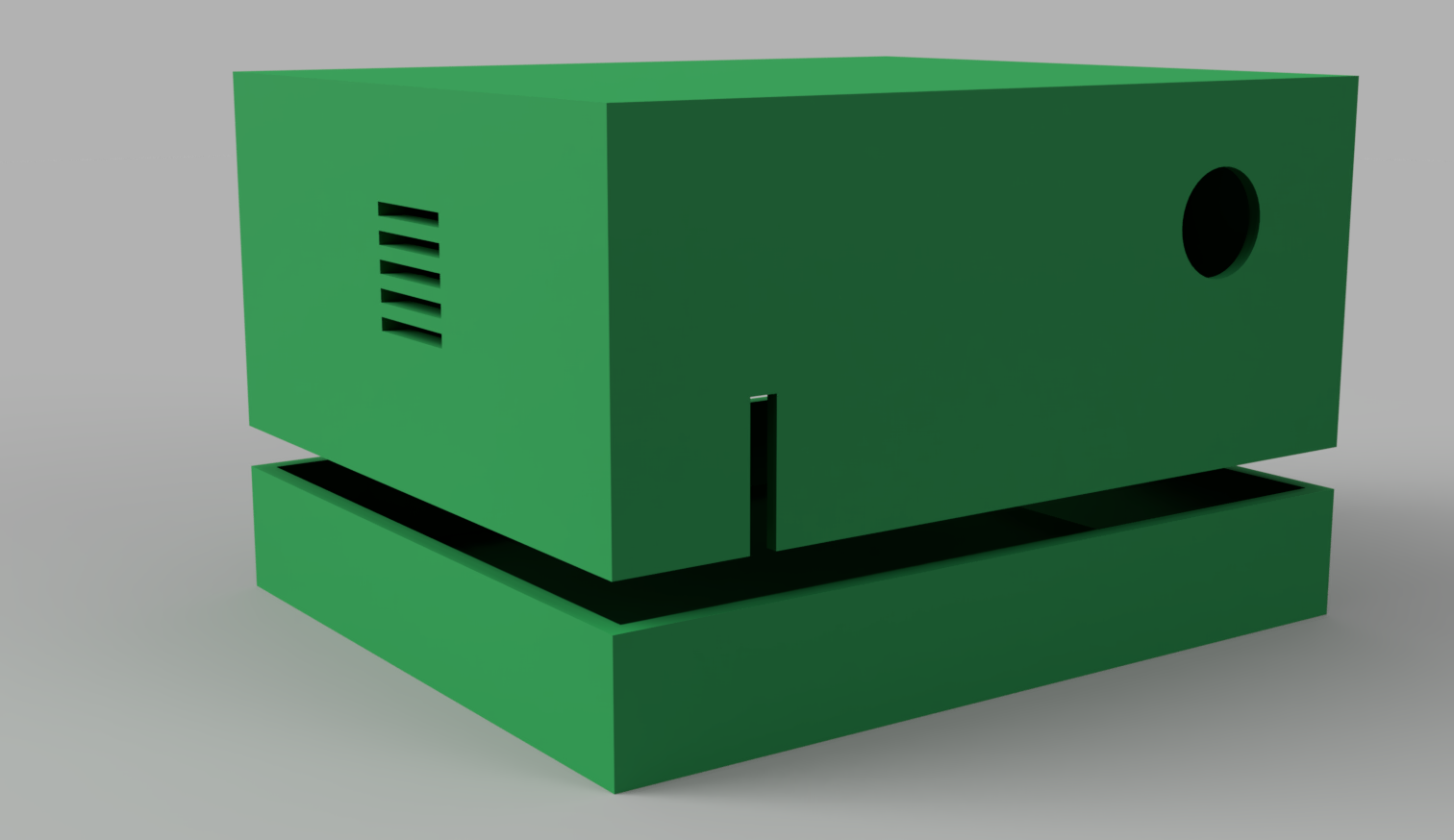


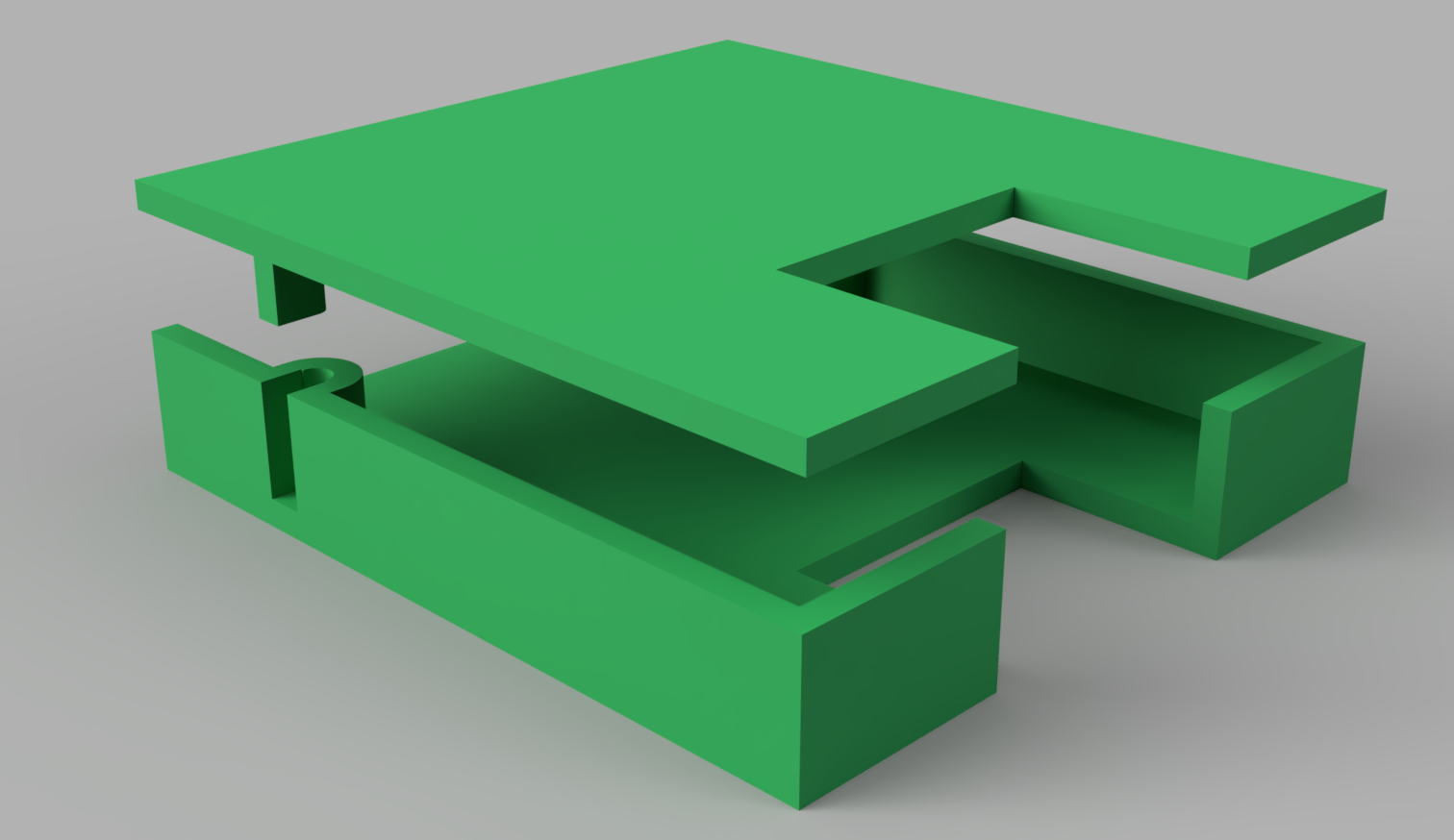
****

**Zabezpieczenie przed warunkami zewnętrznymi**

Aby zabezpieczyć podzespoły urządzenia pomiarowego przed działaniem warunków zewnętrznych i przypadkowymi zwarciami zostały wykonane obudowy (oddzielna dla zespołu mikrokontrolera z sensorem SHT30 oraz pojemnościowego czujnika wilgotności gleby.

Obudowy zostały zaprojektowane w programie Fusion360 – jest to oprogramowanie CAD, które stanowi hobbystyczną alternatywę dla skierowanego do zastosowań profesjonalnych Autodesk Inventor.





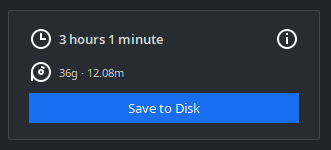
Wyeksportowane modele stworzone w Fusion360 musiały zostać przystosowane do druku w tym celu wykorzystane zostało oprogramowanie Cura – jest to tzw. „Slicer”, czyli program dzielący model na warstwy i generujący zrozumiały dla drukarki 3D gcode.

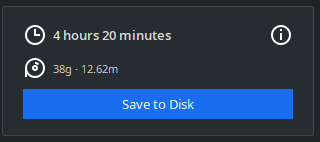
Obraz zawierający tekst, monitor, wewnątrz, sprzęt elektroniczny

Opis wygenerowany automatycznie

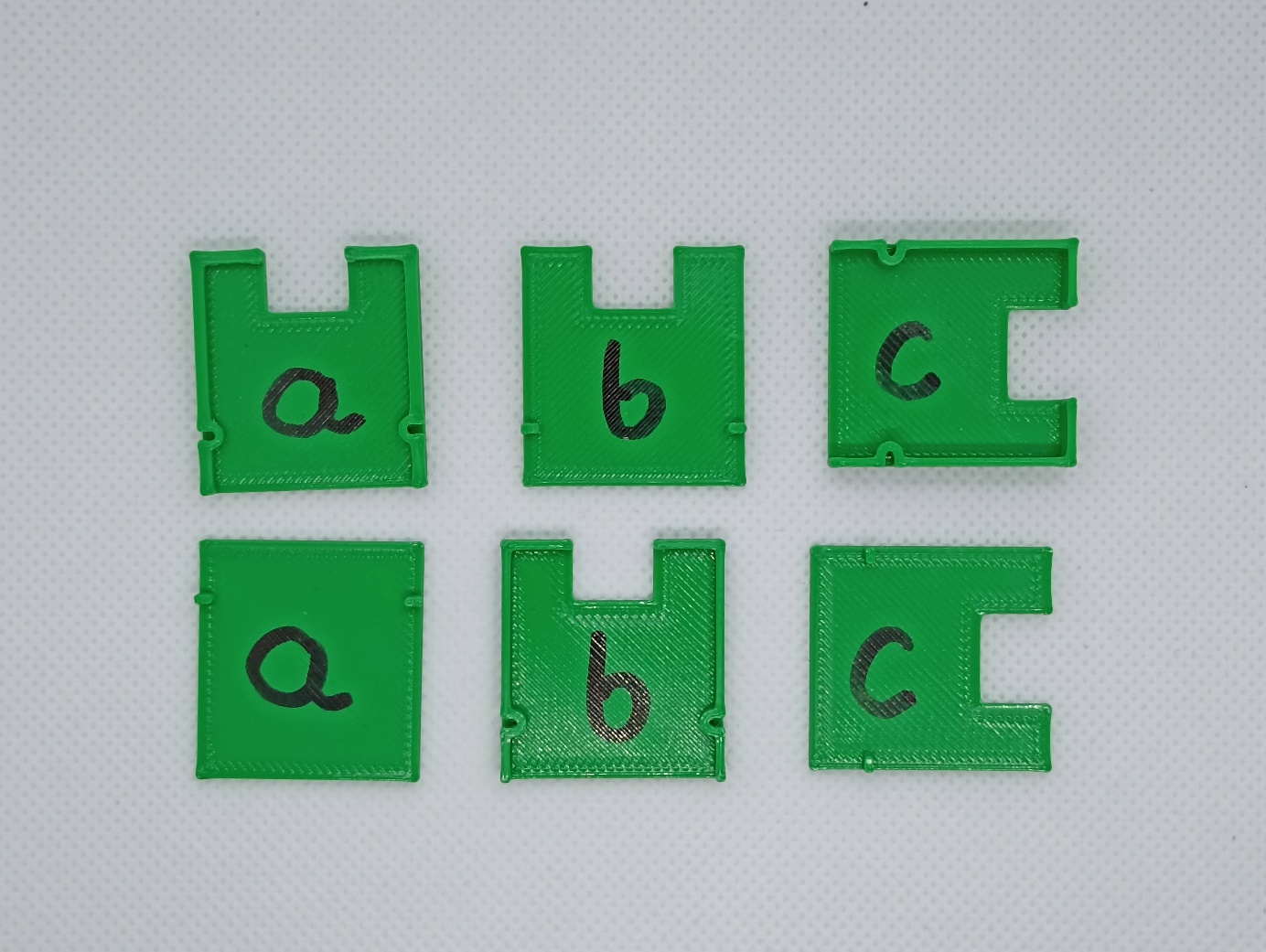
Wydruki zostały wykonane w technologii FDM na drukarce Ender 3 Pro. Zastosowany materiał to PLA. Skrót PLA oznacza poliaktyd, czyli kwas polimlekowy. Jest to polimer otrzymywany z surowców roślinnych, a precyzyjniej rzec ujmując z sfermentowanej skrobi. Zastosowane tworzywo jest biodegradowalne, zatem nie sprawdzi się w omawianym zastosowaniu, jednak ze względu na niską cenę i łatwość druku idealnie sprawdza się w prototypach.

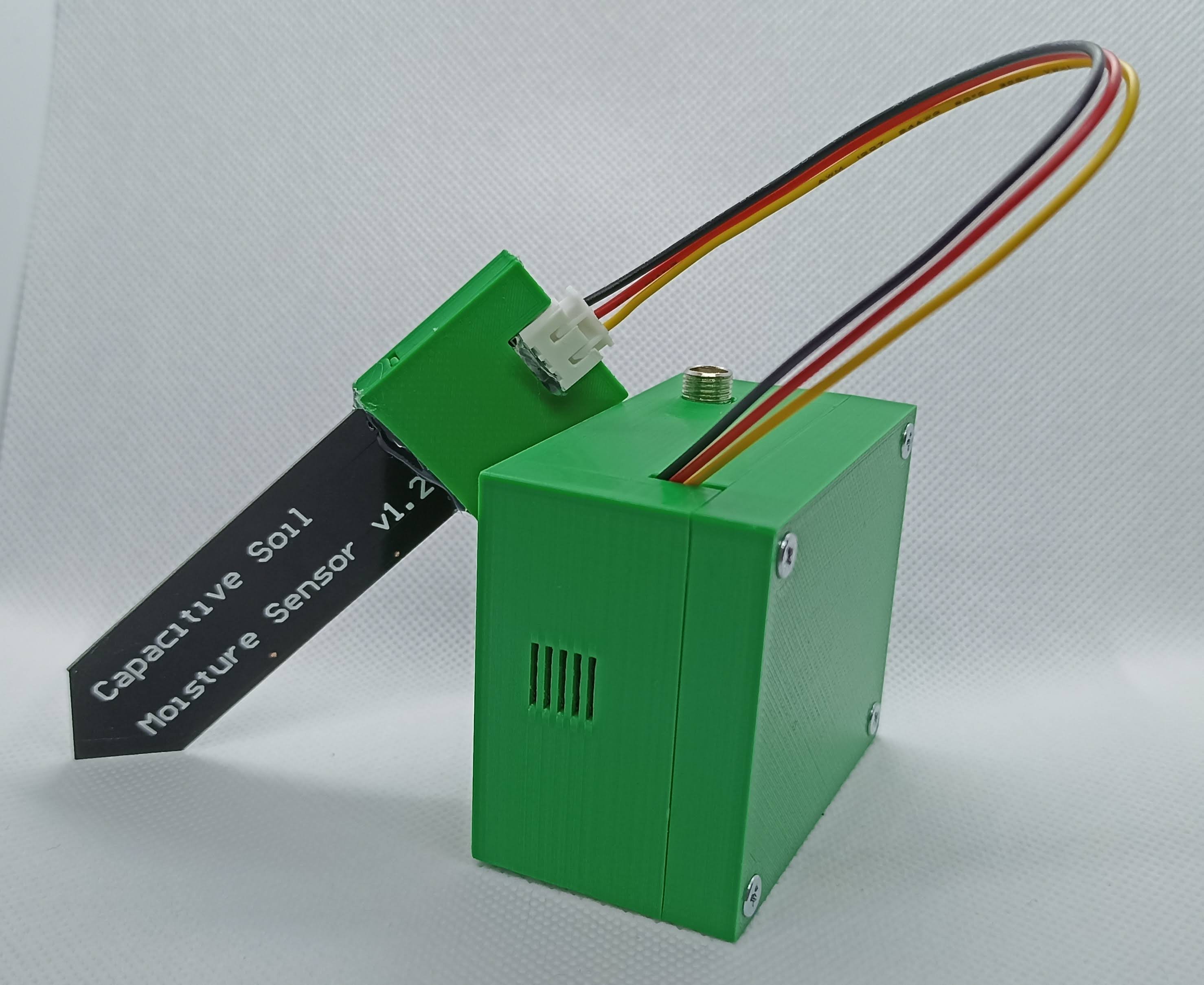
Dojście do finalnej wersji obudowy wymagało kilku iteracji procesu projektowania. Początkowe wydruki dla zaoszczędzenia czasu i materiału były drukowane przy niskim wypełnieniu modelu (10%), prędkości 100 mm/s i na stosunkowo dużej wysokości warstwy (0.28mm). Pozwoliło to na przyspieszenie druku o około 30% w stosunku do finalnej wersji, która została wykonana na 0.2 mm wysokości warstwy, 60% wypełnieniu i prędkości równej 70 mm/s.





Wersja *a* obudów powstała przed otrzymaniem obwodów drukowanych i czujników, stanowiła model poglądowy tego jak może wyglądać złożone urządzenie pomiarowe.  
W iteracji *b* obudowa czujnika wilgotności gleby została poprawiona z uwzględnieniem rzeczywistych wymiarów (parametry podane na stronie dystrybutora były niedokładne). Obudowa urządzenia pomiarowego została wyposażona we wszystkie niezbędne przepusty na kable i gniazdo zasilania. Dodatkowo został dołożony uchwyt czujnika SHT30.  
Finalna wersja *c* została pozbawiona nadmiarowego materiału na ściankach. Zostały wyprofilowane wpusty pod śruby oraz zostało dodane miejsce na lepsze ułożenie przewodów.

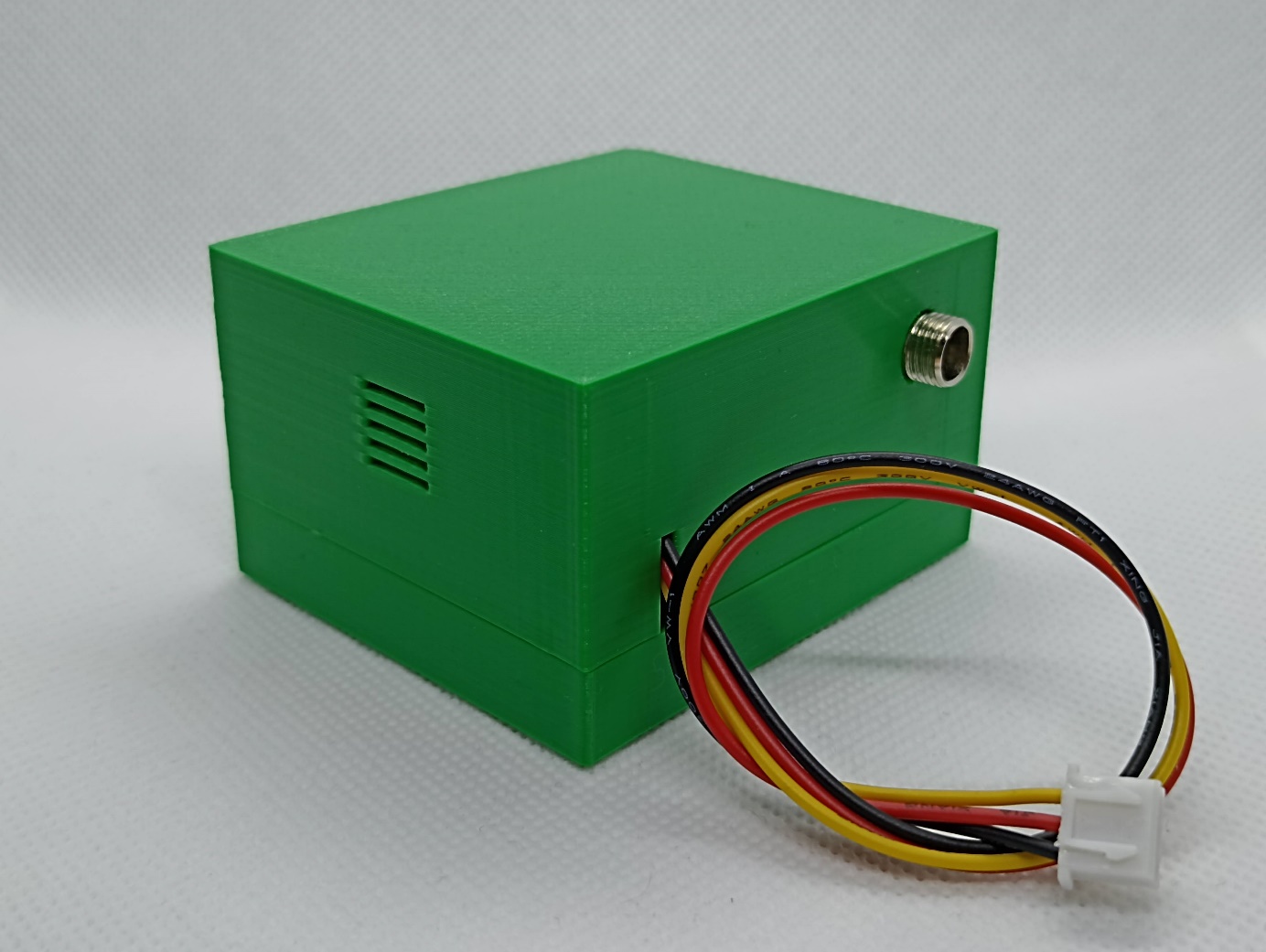




Wydrukowane obudowy dobrze zabezpieczają prototypy na etapie testowania koncepcji projektu, jednak wersja, która miałaby działać w realnych warunkach powinna być dodatkowo zabezpieczona przed działaniem wilgoci i kurzu oraz wykonana z materiału, który nie będzie ulegał biodegradacji w wyniku działania czynników atmosferycznych. Dobrym wyborem mógłby być ABS.

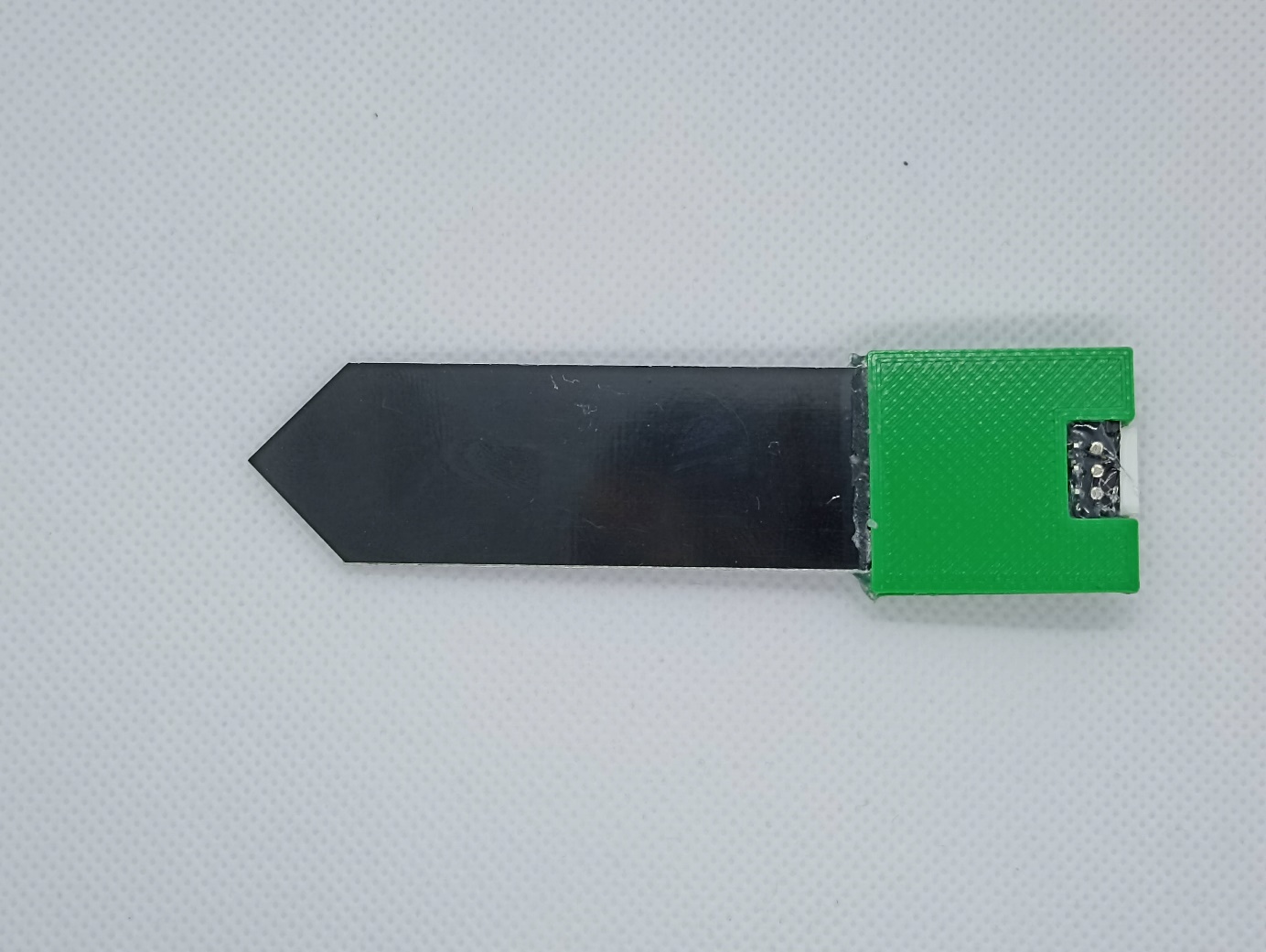
Urządzenie pomiarowe zostało zabezpieczone wykonanymi w technologii druku 3D obudowami. W prototypowej wersji druk został zrealizowany z wykorzystaniem materiału PLA, jednak jest on biodegradowalny, co znaczy, że na dłuższą metę nie sprawdzi się w finalnym zastosowaniu. Dodatkowo, gdyby urządzenie miało być używane w realnym zastosowaniu obudowy powinny lepiej zabezpieczać przed działaniem kurzu i wody.

Mikrokontroler wraz z sensorem SHT30





Obraz zawierający tekst

Opis wygenerowany automatycznie

**Soft**

**Komunikacja na linii mikrokontroler – serwer**

Komunikacja urządzenia pomiarowego z serwerem odbywa się przez Wi-Fi z wykorzystaniem protokołu HTTP (ang. Hypertext Transfer Protocol), który służy do przesyłania dokumentów hipertekstowych za pośrednictwem sieci WWW. Protokół ten pozwala na interakcje z serwerem z wykorzystaniem ustandaryzowanych metod.

Metody wykorzystywane w komunikacji na linii urządzenie pomiarowe – serwer to POST oraz GET.

Dane przesyłane są w formacie JSON ze względu na jego czytelność, łatwą obsługę w frameworku Arduino oraz prostą konwersję do słowników języka Python w części serwerowej.

Alternatywnym podejściem do komunikacji w projekcie mogłoby być zastosowanie MQRR (Message Queue Telemetry Protocol). Jest to protokół stworzony do transmisji danych między urządeniami MQTT i działa w oparciu o wzorzec publikacji-subskrypcji, co oznacza, że urządzenie może publikować wiadomości na określonych kanałach, a odbieranie wiadomości odbywa się poprzez subskrybowanie wspomnianych kanałów.

Decyzja o wybraniu protokołu HTTP zamiast MQTT, który wydaje się być lepszym wyborem w opisywanym zastosowaniu była podyktowana chęcią rozwinięcia nieszablonowego podejścia do stworzenia komunikacji na linii mikrokontroler – serwer.

**Oprogramowanie mikrokontrolera**

**Wprowadzenie**

Oprogramowanie mikrokontrolera ma spełniać trzy podstawowe funkcje:

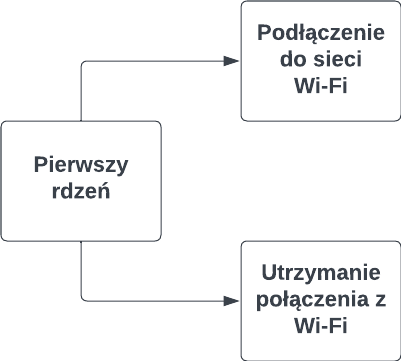
* Utrzymanie stabilnego połączenia z siecią Wi-Fi
* Odczyt i przetwarzanie informacji z podłączonych czujników
* Komunikacja z serwerem

Platformą na której zaimplementowane zostały powyższe funkcjonalności jest wspomniany wyżej mikrokontroler ESP32. Wbudowany moduł sieci bezprzewodowej pozwolił na zrezygnowanie z zewnętrznego modułu Wi-Fi oraz zegara czasu rzeczywistego. Ponadto mikroprocesor Tensilica Xtensa LX6 pozwolił na odseparowanie priorytetowej funkcjonalności utrzymania połączenia z siecią bezprzewodową od reszty.

Oprogramowanie urządzenia pomiarowego było tworzone z wykorzystaniem środowiska Arduino IDE 2.0 ze względu na wbudowane rozszerzenie do prostego pobierania potrzebnych bibliotek, wbudowany kompilator, monitor portu szeregowego oraz przede wszystkim narzędzie do wgrywania skompilowanego kodu bezpośrednio na zastosowaną płytkę.

https://docs.platformio.org/en/stable/frameworks/arduino.html

Język w jakim zostało napisane oprogramowanie to pochodna języka C wykorzystywana głównie w programowaniu płytek rozwojowych Arduino. Wykorzystanie tego frameworka było ponieważ poza mikroprocesorami firmy Atmel (wykorzystywanymi w Arduino) wspiera on również inne chipy między innymi te produkowane przez Espressif.





**Schematy blokowe i omówienie działania programu**

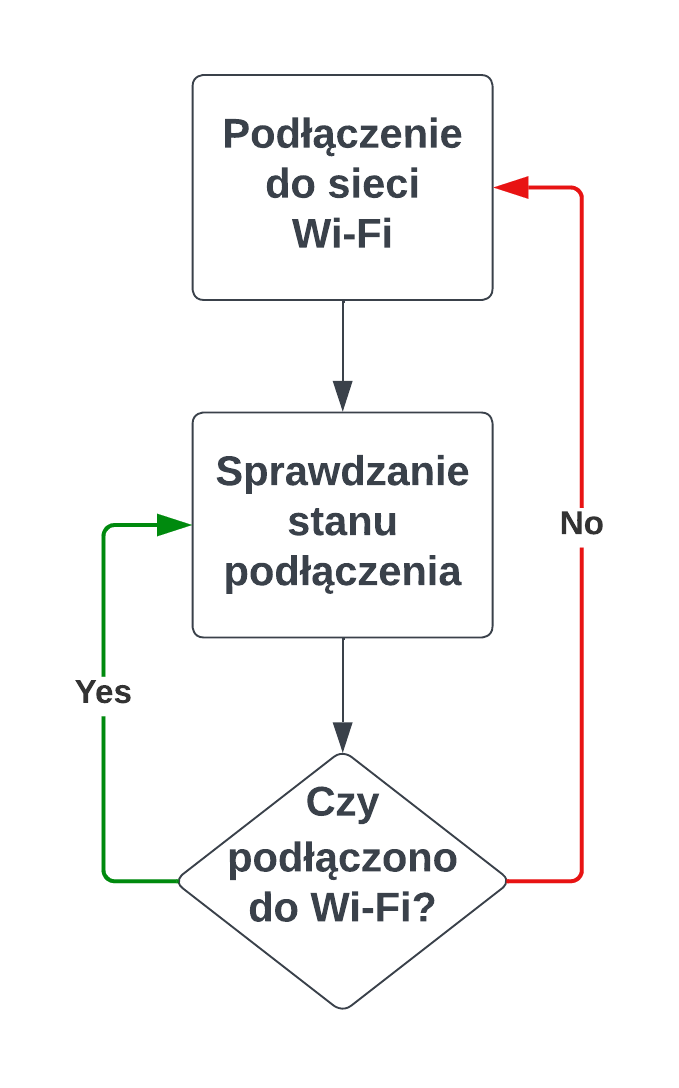
W oprogramowaniu urządzenia pomiarowego można wyszczególnić 3 główne sekcje:

* Obsługa połączenia z siecią Wi-Fi
* Zbieranie i przetwarzanie danych z czujników
* Komunikacja z serwerem

Aby zapewnić stabilną i pewną pracę programu wykorzystany został FreeRTOS – system operacyjny czasu rzeczywistego dla zastosowań embeded. Pozwala on na programowanie wielowątkowe oraz zarządzanie cechami wątków takimi jak m.in. priorytet wykonywania i przypisanie do wskazanego rdzenia procesora.

W omawianym przypadku dla każdej z wymienionych sekcji został utworzony dedykowany wątek (task) z czego obsługa połączenia z siecią bezprzewodową została odseparowana od reszty programu. Przypisana została do pierwszego rdzenia procesora. Powoduje to, że jej działanie nie zostanie w żaden sposób zakłócone przy wykonywaniu pozostałych funkcji. Drugi rdzeń odpowiada za obsługę czujników i komunikację z serwerem.

Stabilne działanie funkcji odpowiadającej za nawiązanie komunikacji bezprzewodowej jest priorytetowe, ponieważ stworzona procedura odpowiada również za ponowne połączenie w przypadku zerwania połączenia z siecią Wi-Fi



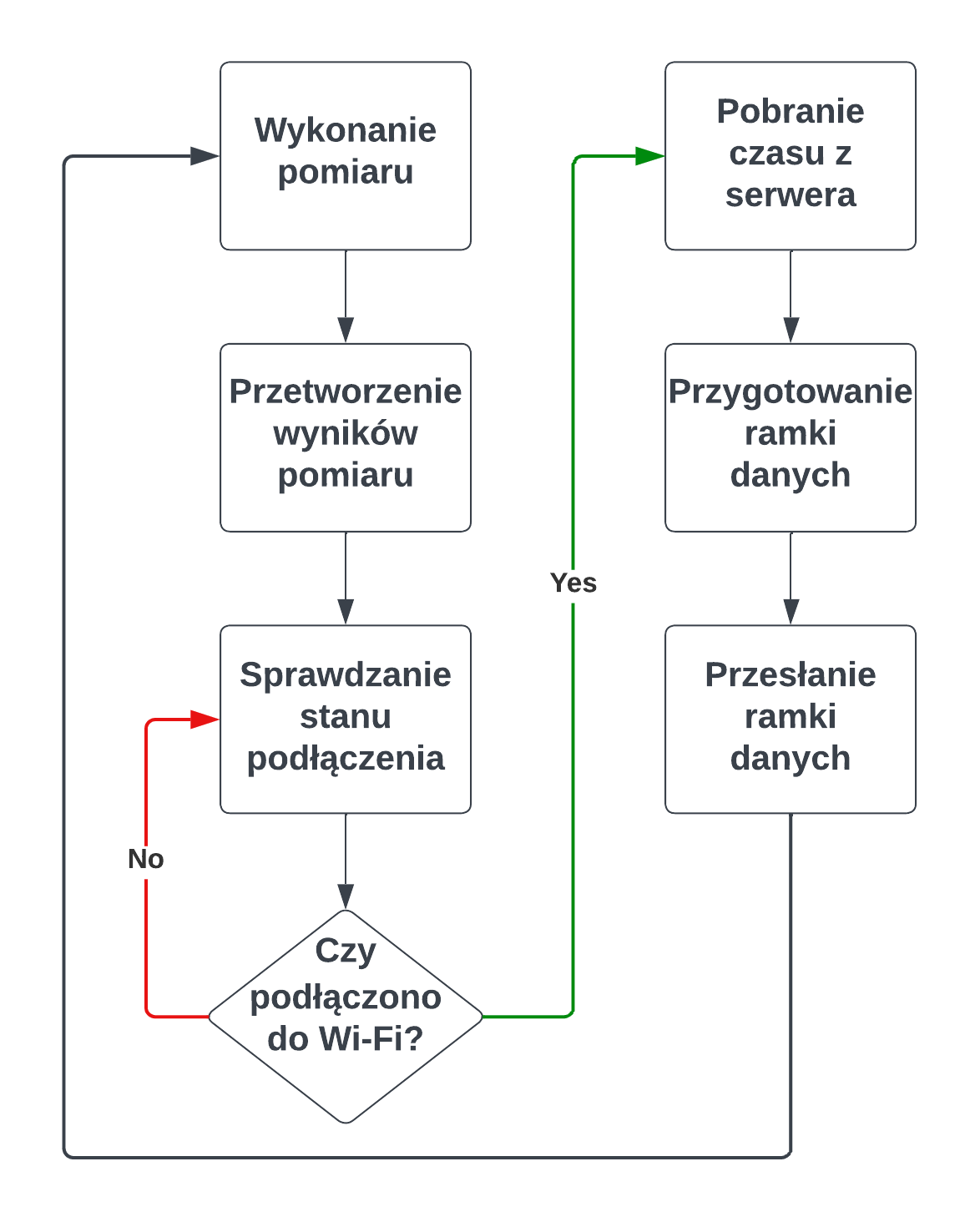
Schemat procedury wifi



Kod procedury wifi

Przy starcie urządzenia podejmowana jest próba podłączenia do Wi-Fi, jeśli się powiedzie procedura wchodzi w stan monitorowania stanu połączenia w stałych interwałach. W przypadku zerwania komunikacji z siecią bezprzewodową następuje próba ponownego połączenia. Urządzenie pozostaje w tym stanie, dopóki nie zostanie przywrócone połączenie, następnie przechodzi w tryb monitorowania.

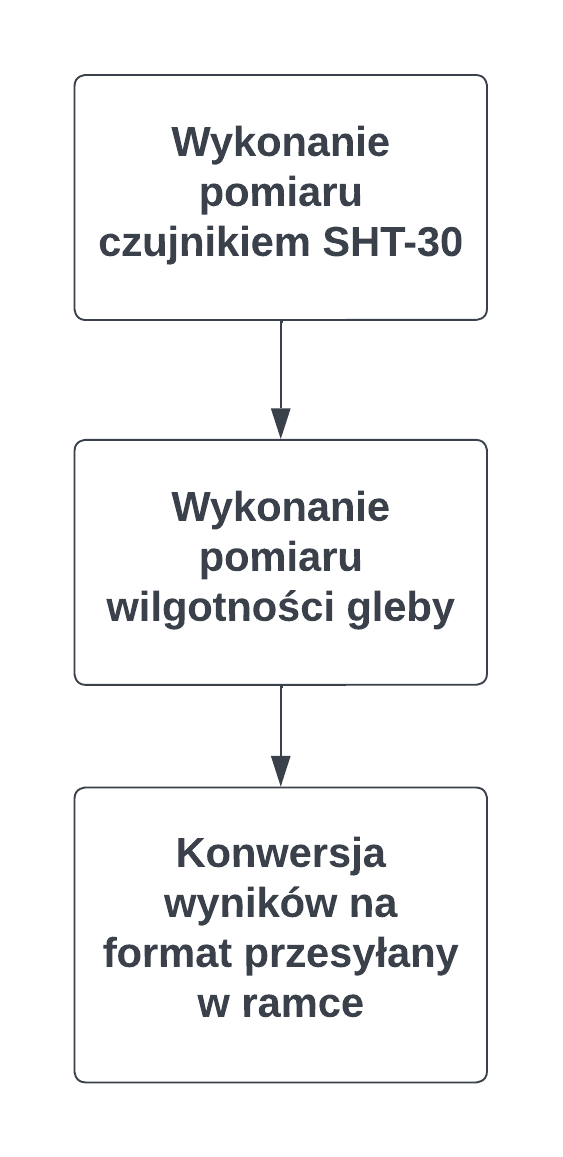
Drugi rdzen



Drugi rdzeń obsługuje przedstawione na schemacie blokowym procedury. Wykonują się one cyklicznie i ich działanie jest zależne od funkcjonalności zaimplementowanej w pierwszym rdzeniu. W przypadku braku połączenia z siecią bezprzewodową nie zostaje podjęta próba przesłania ramki danych do serwera.

Pomiary prowadzone są w przedstawionej funkcji. Zbiera ona wyniki z czujnika SHT30 oraz odczytuje i przelicza na wartości procentowe odczyty z pojemnościowego czujnika wilgotności gleby. Dane te są wykorzystywane w wątku odpowiadającym za przesyłanie danych do serwera

Schemat procedury pomiarowej

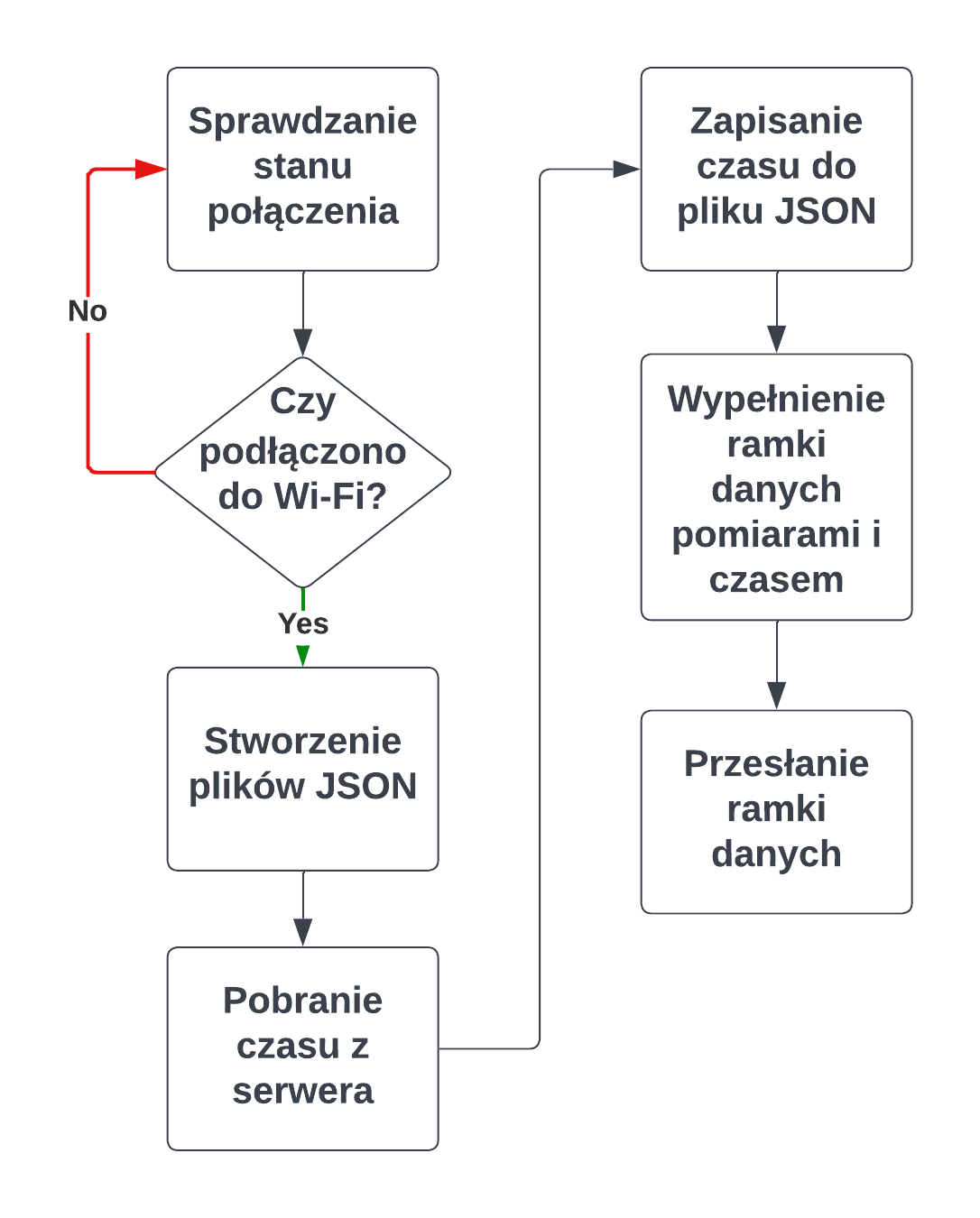


Kod procedury pomiarowej



Ostatnia z funkcjonalności to komunikacja z serwerem.

Schemat blokowy komunikacji



Kod komunikacji



Komunikacja z serwerem opiera się na dwóch zapytaniach protokołu http. Metoda GET pobiera aktualny czas. POST przesyła dane w formacie JSON. Wysyłana ramka danych zawiera datę i godzinę wykonanego pomiaru (timestamp) oraz wyniki pomiarów z sensorów (temperatura i wilgotność powietrza, wilgotność gleby). W pamięci urządzenia ramka danych przechowywana jest jako dokument o stałym rozmiarze. Waga dokumentu jest sztywno określona i bierze pod uwagę rozmiar przesyłanych danych oraz margines bezpieczeństwa. Możliwe było zaimplementowanie dynamicznego przypisywania wielkości pliku JSON, jednak predefiniowany rozmiar eliminuje błędy, które mogłyby wystąpić przy błędach w działaniu algorytmu definiującego wielkość pliku.

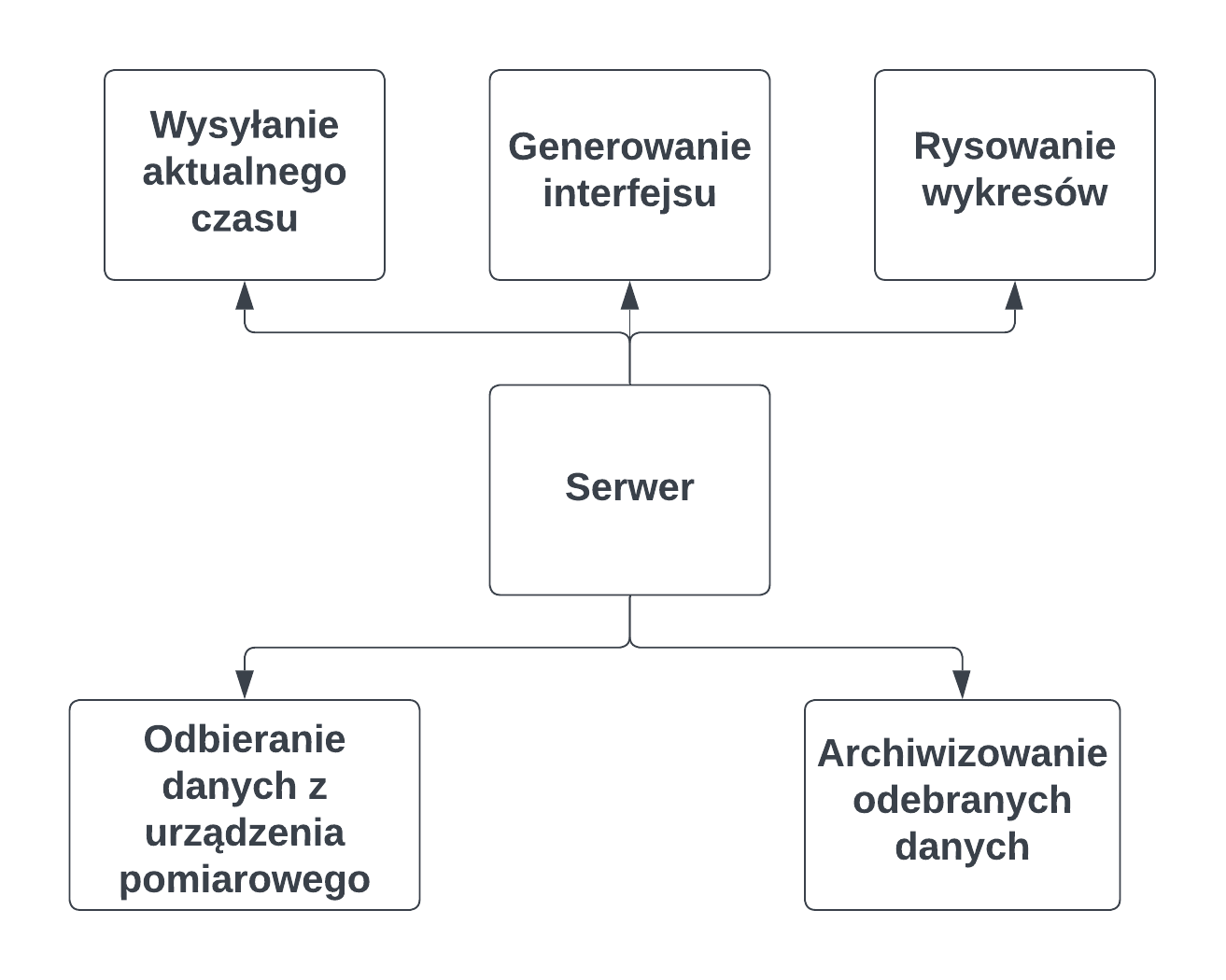
**Oprogramowanie serwerowe**

**Wprowadzenie**

Oprogramowanie serwerowe zgodnie z założeniami projektu zawiera poniższe funkcjonalności:

* Komunikacja bezprzewodowa z dowolną ilością urządzeń pomiarowymi
* Gromadzenie odczytów z *n* urządzeń
* Graficzne przedstawianie najnowszych pomiarów
* Graficzne przedstawienie wszystkich pomiarów z dowolnej ilości urządzeńurządzeń.

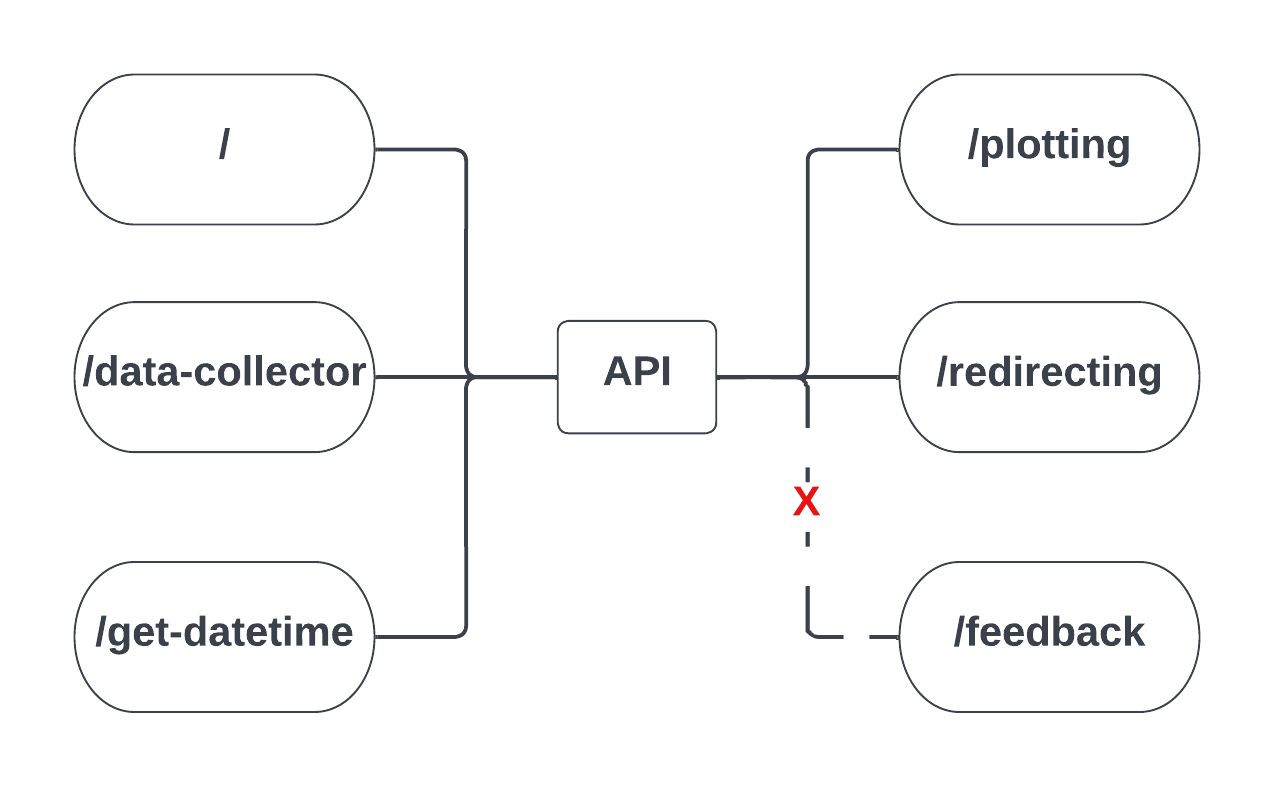
**Schemat ideowy strony serwerowej**



Oprogramowanie serwerowe pełni kluczową rolę w systemie, umożliwia przetwarzanie i analizowanie danych z dowolnej ilości urządzeń pomiarowych podłączonych do sieci bezprzewodowej. Zostało zaimplementowane w postaci API, czyli interfejsu umożliwiającego komunikację pomiędzy aplikacjami i serwisami internetowymi

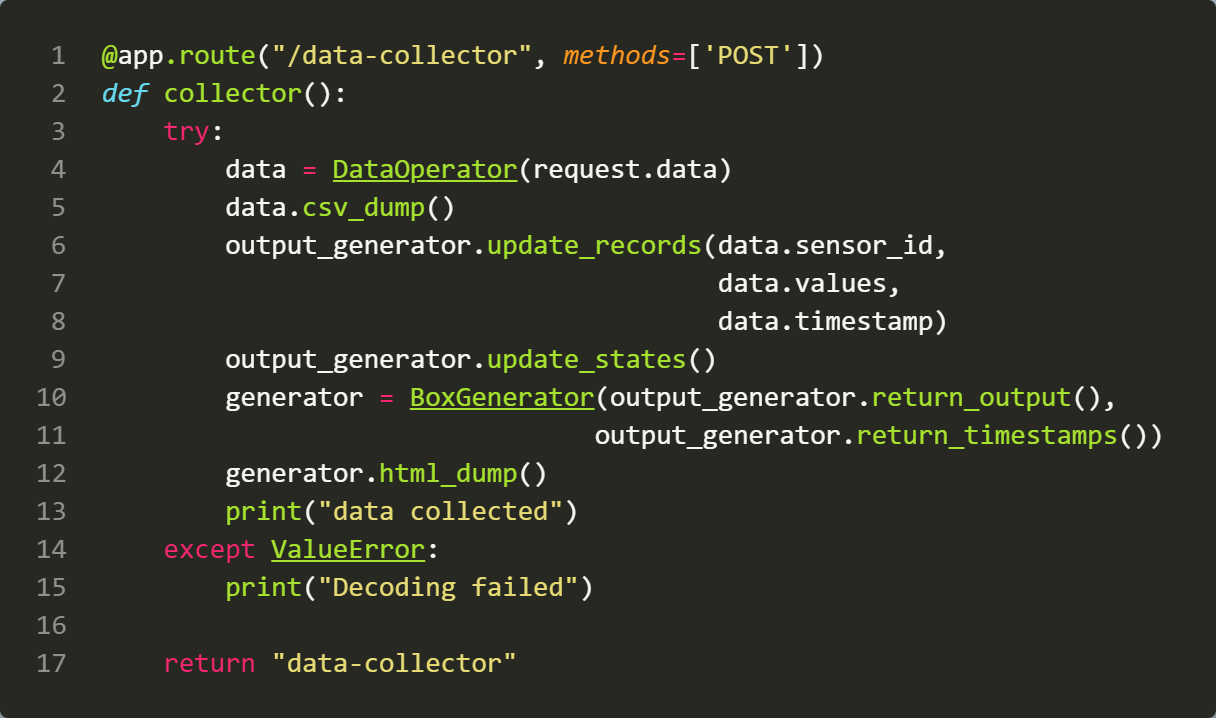
Program został napisany w języku Python z wykorzystaniem frameworka Flask, który jest lekką i prostą w obsłudze biblioteką pozwalającą tworzyć API do obsługi zapytań HTTP. Główną zaletą zastosowanej technologii jest możliwość szybkiego prototypowania kolejnych funkcjonalności oraz obszerna dokumentacja, która bardzo ułatwia rozwiązywanie skomplikowanych problemów. Ze względu na charakterystykę zastosowanego frameworka, napisany program jest otwarty na możliwość rozbudowy w przypadku chęci rozszerzenia projektu o np. urządzenia wykonawcze, lub bardziej skomplikowane urządzenia pomiarowe.

API pozwala na komunikację przy pomocy endpointów, czyli miejsc, przez które można nawiązać połączenie z aplikacją. Ich struktura została zaprojektowana na podstawie schematu ideowego części serwerowej projektu.



**Endpoint /** odpowiedzialny jest za wyświetlanie interfejsu użytkownika (dashboard). Jego jedyną rolą jest renderowanie wygenerowanego przez aplikację serwerową pliku HTML przy wejściu na stronę główną projektu.

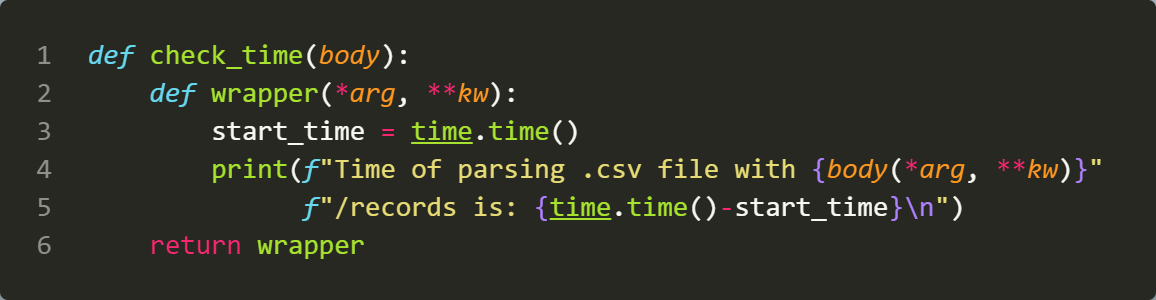
**Endpoint /data-collector** odpowiada za odbieranie ramki danych z urządzenia pomiarowego – z tego powodu obsługuje jedynie metodę POST. Obsługuje go funkcja **collector()**. W funkcji została wykorzystana konstrukcja *Try Except*, której zadaniem jest wyłapanie błędów związanych z dekodowaniem danych wejściowych i ewentualne przerwanie procedury.



Za przetworzenie odebranej ramki danych oraz jej zapisanie w pliku CSV odpowiada klasa DataOperator. Podczas tworzenia jej instancji ramka danych jest automatycznie konwertowana na słownik języka Python. Następnie wywoływana jest metoda *csv\_dump()*, która zapisuje odebrane i przetworzone dane do pliku CSV.

Zaletą stosowania formatu CSV jest możliwość otworzenia historii odczytów w arkuszu kalkulacyjnym, bez konieczności dodatkowej konwersji. Wadą może być rosnący czas dostępu do próbek wraz z wzrostem ich ilości.

Aby sprawdzić realny wpływ wspomnianego problemu na działanie projektu został przeprowadzony prosty test mający na celu sprawdzenie jak logarytmiczny przyrost ilości próbek wpływa na czas odczytu. Procedura testowa polegała na sprawdzeniu różnicy pomiędzy próbkami czasu przed i po rozpoczęciu działania funkcji przetwarzającej plik CSV.



Obraz zawierający tekst

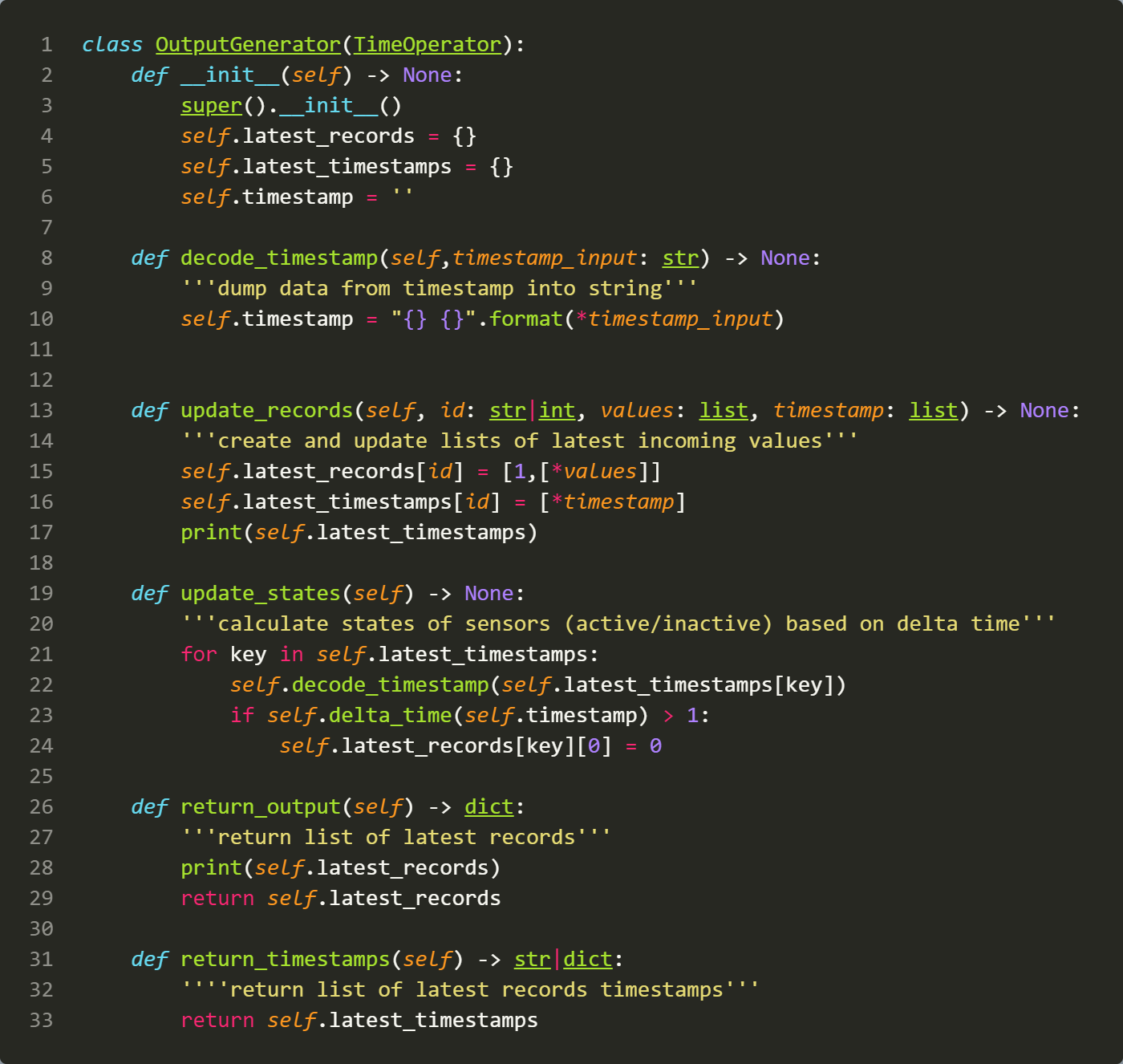
Opis wygenerowany automatycznie

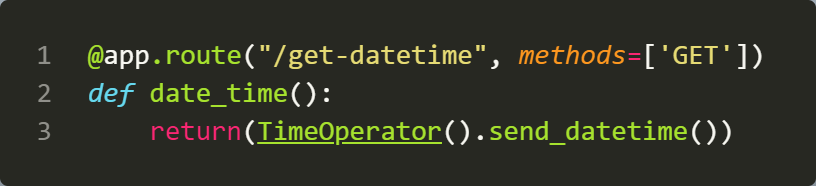
Na podstawie raportu działania funkcji można zauważyć, że do 100\_000 próbek, czas odczytu utrzymywany jest poniżej sekundy, co skutkuje stosunkowo szybkim dostępem do interfejsu odpowiadającego za rysowanie wykresów. Problem pojawia się przy ilości próbek wynoszącej 1\_000\_000. Czas przetwarzania pliku jest bliski 9 sekund, jest to wartość poważnie wpływająca na responsywność funkcjonalności.

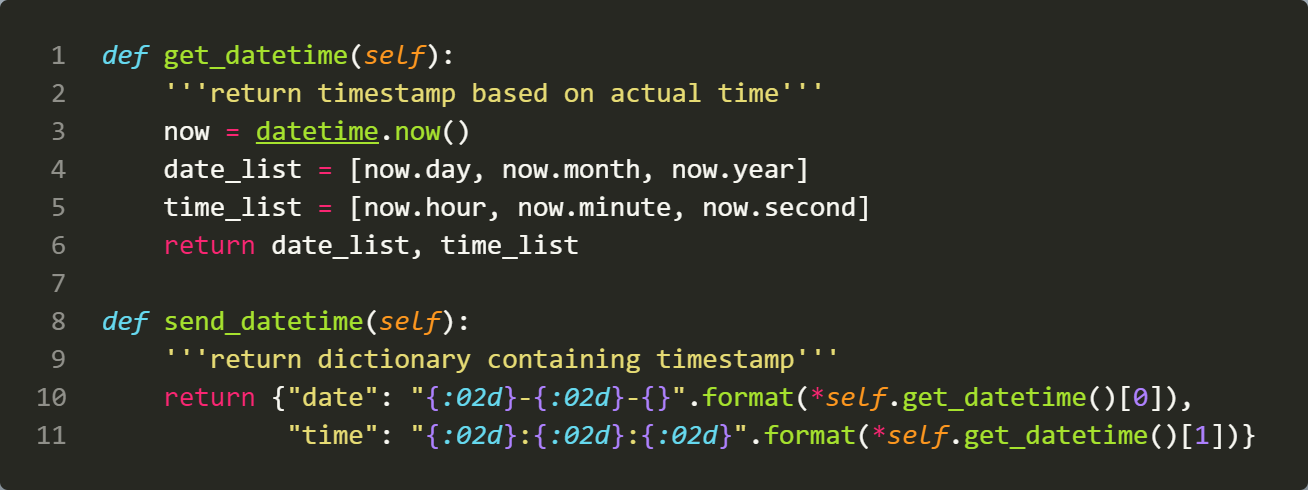
Przyjmując założenie, że system ma działać na tyle długo, żeby miał możliwość zebrania ilości próbek w granicy miliona, potrzebny byłby inny, szybszy w działaniu sposób przechowywania danych, jednak w aspekcie prototypowym osiągnięte czasy przetwarzania są wystarczające.

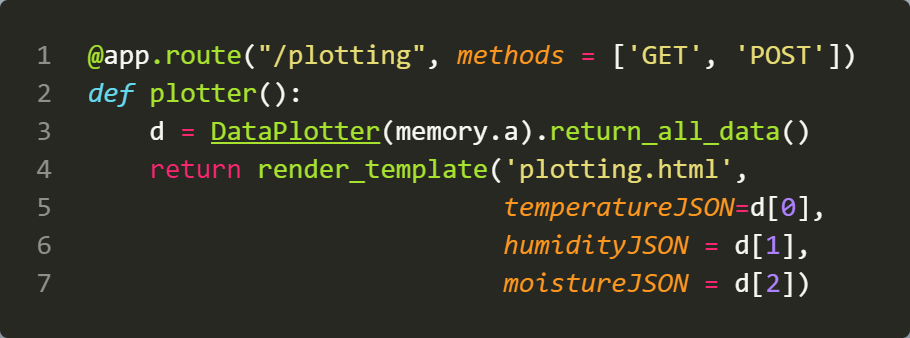
Funkcja *collector* określająca działanie endpointa wywołuje następnie funkcje *update\_records*, która jest metodą klasy *OutputGenerator*. Odpowiada ona za wygenerowanie słownika zawierającego najnowsze dane odebrane z urządzeń pomiarowych i stanem urządzenia (domyślnie stan określający urządzenie jako aktywne) oraz słownika z znacznikami czasowymi ostatnich pomiarów.

Metoda *update\_states* aktualizuje stany urządzeń pomiarowych. Stan definiowany jest na podstawie różnicy pomiędzy aktualnym czasem, a czasem ze znacznika w ramce danych.

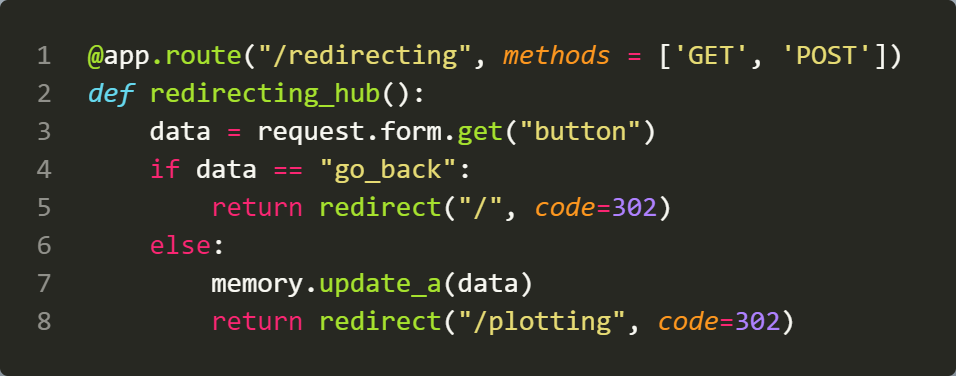


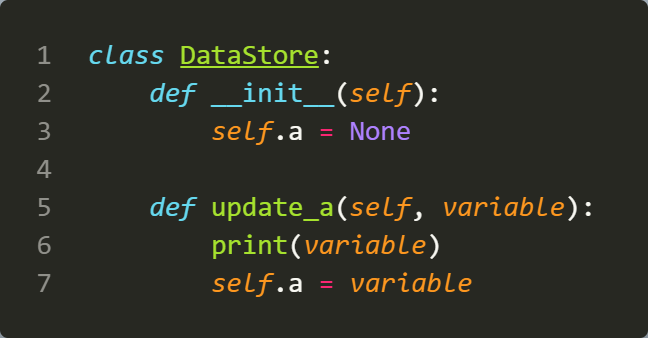
**Endpoint /get-datetime** publikuje aktualny czas. Celem jego wprowadzenia jest wyeliminowanie konieczności stosowania fizycznego zegara czasu rzeczywistego w urządzeniu pomiarowym, zamiast tego, czas potrzebny do uzupełnienia znacznika czasu pobierany jest z serwera.

Zastosowana metoda *send\_datetime* będąca elementem klasy *TimeOperator* generuje ramkę danych ograniczoną jedynie do czasu. Przesyłane dane są konwertowane do formatu JSON podczas pobierania przez urządzenie pomiarowe.

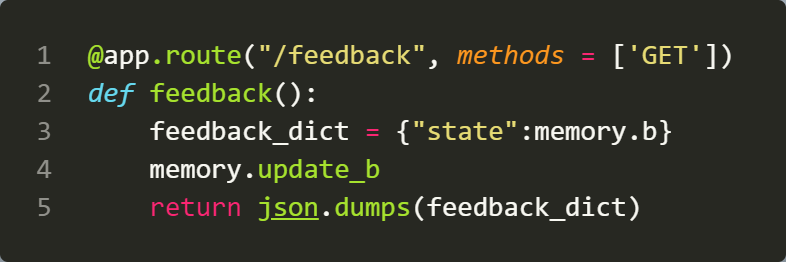
**Endpoint /plotting** odpowiada za wygenerowanie wykresów dla wskazanego czujnika. Wskazanie czujnika odbywa się poprzez kliknięcie odpowiedniego przycisku w głównym interfejsie programu. Danymi wejściowymi dla wygenerowanego wykresu jest pełna historia pomiarów przechowywana w pliku CSV.

**Endpoint /redirecting** obsługuje przełączanie się pomiędzy elementami interfejsu. W przypadku przejścia z głównego interfejsu do rysowania wykresów aktualizuje zmienną odpowiadającą za wybór czujnika. Jej wartość przesyłana jest do serwera poprzez plik HTML.



Do utrzymywania stanów pamięci i prostej wymiany danych pomiędzy funkcjami wprowadzona została klasa *DataStore*, która aktualnie posiada jedną zmienną – *a*, oraz jedną metodę *update\_a*, jednak w przypadku rozbudowania projektu może zostać łatwo rozszerzona, tak, aby pozwalała na dowolne manipulowanie elementami pamięci.

**Endpoint /feedback** został zaprojektowany oraz zaimplementowany w pierwszej wersji projektu, jednak został usunięty ze względu na problemy, które generował podczas testów. Jego zadaniem było zwracanie odpowiedzi dla czujnika o powodzeniu/niepowodzeniu przesłania ramki danych. Dane wyjściowe przekazywane były w formacie JSON. Implementacja takiego rozwiązania okazała się problematyczna przy dużej ilości urządzeń pomiarowych. Proponowane rozwiązanie działało w przypadku jednego, kilka urządzeń pomiarowych wymagałoby dynamicznego tworzenia endpointów przypisanych do konkretnych ID.



**Symulowanie działania z kilkoma czujnikami.**

Aby zasymulować działanie z kilkoma czujnikami, napisany został skrypt wysyłający ramki danych dla dowolnej ilości czujników. Pozwoliło to na zweryfikowanie, czy system jest w stanie zebrać dane, wygenerować interfejs oraz wykresy dla większej ilości urządzeń pomiarowych.



Skrypt symulujący działanie czujników działa na adresie lokalnym komputera (127.0.0.1), aby umożliwić testowanie systemu bez konieczności połączenia z siecią bezprzewodową. Ramka danych wypełniana jest losowymi wartościami dla ID sensora, temperatury i wilgotności powietrza oraz wilgotności gleby. Znaczniki czasu zawierają rzeczywisty czas przesłania ramki danych.

**FRONTEND**

Interfejs systemu dostępny jest z poziomu przeglądarki. Jest podzielony na dwie sekcje:

* listowanie czujników wraz z ich najnowszymi pomiarami oraz stanem (generowanym na podstawie znacznika czasu ostatnio odebranej ramki danych)
* rysowanie wykresu temperatury, wilgotności powietrza oraz wilgotności gleby

Interfejs listujący czujniki (dashboard) jest generowany z poziomu serwera. Ilość boxów zależy od ilości czujników, z których system odebrał dane. Plik HTML składa się z nagłówka, który pozostaje niezmienny



Szczegóły wyglądu interfejsu zapisane są w formacie CSS. W pliku ze stylem znajdują się klasy determinujące wygląd boxów dla aktywnego i nieaktywnego czujnika. Przypisanie klasy do odpowiedniego boxa czujnika odbywa się na etapie generowania pliku HTML przez serwer. Stan jest determinowany przez interwał czasowy między ostatnim pomiarem a aktualnym czasem.

HTML dla jednego czujnika



CSS

Interfejs wygenerowany dla jednego czujnika



GIT i Github

W projekcie do przechowywania i kontroli kodu źródłowego został wykorzystany system kontroli wersji GIT. Pozwolił on na porównywanie zmian, przywracanie konkretnych wersji pisanego oprogramowania oraz na zachowanie przejrzystości pracy. Dodatkowo możliwość pracy na gałęziach (branch) umożliwiła tworzenie testowych wersji funkcjonalności bez wpływania na główny tok rozwoju programu.

Przechowywanie kodu odbywa się przy pomocy serwisu GitHub.

Trello

Trello jest narzędziem do zarządzania projektami.