# ESP32 Measurement Station

Jan Wołowiec, Szymon Głogowski

September 26, 2021

## Contents

# 1 Wprowadzenie

Projekt ten wykorzystuje układ ESP32-WROOM-32D wraz z podłączonym za pomocą magistrali I2C wielofunkcyjnym czujnikiem BME280. Czujnik pozwala realizować pomiary ciśnienia, temperatury i wilgotności.

Dodatkową funkcjonalnością systemu jest wykorzystanie aktualizacji OTA, sprzężonej z pipeline'em Azure DevOps, celem ograniczenia potrzeby ingerencji w sprzęt i ułatwienia procesu aktualizacji.

Dane zebrane przez czujnik są wysyłane za pomocą requestów HTTP na platformy typu ThingSpeak lub thinger.io pozwalające na wyświetlanie zebranych danych. Dodatkowo z pomocą darmowego hostingu oraz bazy danych MySQL zostało stworzone własne rozwiązanie.

Link do projektu: `https://dev.azure.com/jwolowiec/ESP32`
Link do strony: `https://esp32ms.000webhostapp.com`

# 2 Architektura systemu

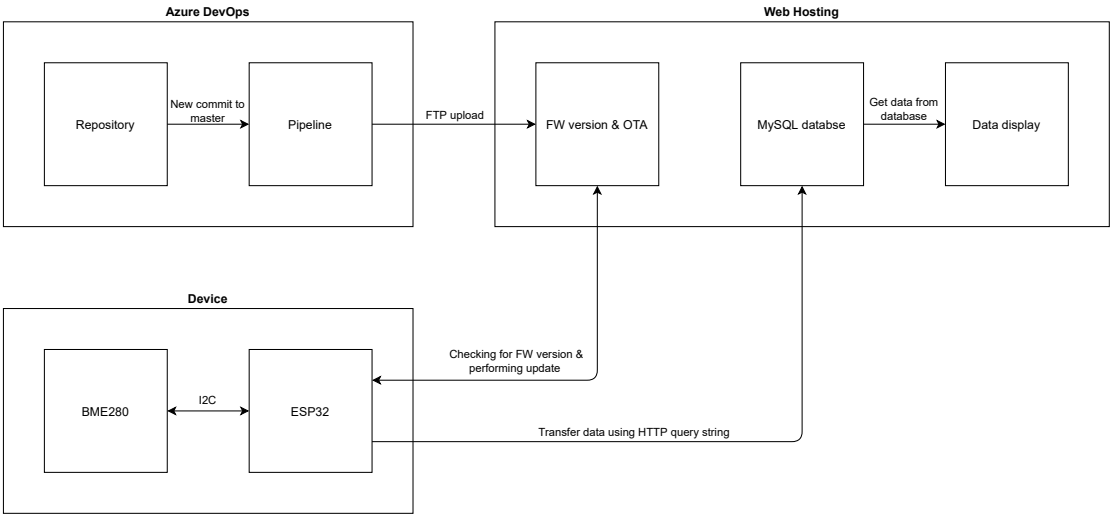


Figure 1: Schemat blokowy zastosowanej architektury

Rozdział ten ma na celu ogólne opisanie poszczególnych bloków składowych systemu. Szczegółowy opis implementacji znajdzie się w kolejnych rozdziałach.

## 2.1 Azure DevOps

Platforma Azure DevOps została wybrana do wykorzystania w projekcie ze względu na zaawansowane funkcjonalności zarządzania projektem oraz wdrażaniem systemów CI/CD.

Szczegółowy opis Platformy został umieszczony w rozdziale 3.

### 2.1.1 Repository

Repozytorium **GIT** znajdujące się na platformie **Azure DevOps**. Bezpośrednie wprowadzaniem zmian do brancha **master** zostało zablokowane na rzecz wykorzystania mechanizmu Pull Requestów.

### 2.1.2 Pipeline

Na platformie **Azure DevOps** został skonfigurowany system CI/CD wykonujący buildowanie projektu po każdym nowym commicie do brancha **master**. Artefakty otrzymane po wykonaniu pipeline'u zostają umieszczone na Web Hostingu za pomocą protokołu FTP.

## 2.2 Device

Urządzenie może być zasilane z dowolnego portu USB, jak np. ładowarki do telefonu. Dodatkowo wymagane jest połączenie z WIFI, celem wysyłania danych i pobierania aktualizacji.

Szczegółowy opis implementacji został umieszczony w rozdziale 4.

### 2.2.1 ESP32-WROOM-32D

Układ SoC oferujący możliwość komunikacji m.in. po WiFi oraz Bluetooth.

Figure 2: Wykorzystany układ SoC.

### 2.2.2 BME280

Sensor firmy Bosch realizujący pomiary ciśnienia, temperatury i wilgotności. Wykorzystuje on do komunikacji z układem ESP32 magistralę I2C.

Link do dokumentacji: BME280

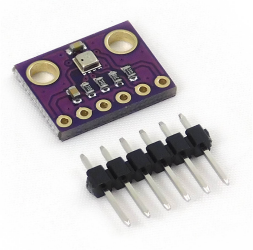

Figure 3: Wykorzystany czujnik.

## 2.3 Web Hosting

Do naszych celów został wybranych darmowy hosting `https://www.000webhost.com`, ze względu na rozbudowaną darmową wersję usługi. Zapewnia on m.in. certyfikat SSL, bazę danych MySQL, oraz co najważniejsze nie posiada wbudowanych zabezpieczeń przed botami. Pierwsze próby korzystania z hostingu odbyły się u dostawcy `https://infinityfree.net`, okazało się jednak że dostawca ten stosuje zebezpieczenie przed botami wymagające obsługi JavaScriptu i ciasteczek w kliencie, co skutecznie eliminowało możliwość dostępu do strony przez nasze urządzenie.

Szczegółowy opis stworzonej strony został umieszczony w rozdziale 5.

### 2.3.1 FW version & OTA

Web Hosting przechowuje wersję najnowszego build'u oraz binarny plik aktualizacji OTA, pozwala to na weryfikację przez układ ESP czy jego wewnętrzna wersja oprogramowania jest równa tej przechowywanej na serwerze. Jeśli zostanie wykryta nowsza wersja, nastąpi pobranie pliku binarnego OTA oraz zostanie przeprowadzona aktualizacja.

Omawiane wcześniej pliki przechowywane są pod adresem:
`https://esp32ms.000webhostapp.com/uploads`

### 2.3.2 MySQL database

Wymagane było zaimplementowanie mechanizmu przechowywania danych odczytywanych przez urządzenie. W naszym przypadku wzięliśmy przykład z istniejących platform do wizualizacji danych typu thinger.io lub thingspeak.com. Posiadają one Web API pozwalające na przesyłanie do nich danych za pomocą requestów HTTP typu POST lub w query dla requestów GET. W naszym rozwiązaniu zostały wykorzystane zapytania GET wraz z query stringami, do umieszczania danych w bazie MySQL.

Zapytania GET wysyłane są pod adres:
`http://esp32ms.000webhostapp.com/add_data.php`

Przykładowy query string zawierający dane zostało przedstawione poniżej:
`?temp=19.99&humi=23.45&pres=1001.01`

### 2.3.3 Data display

Omawiana strona służąca do wizualizacji zebranych danych znajduje się pod adresem:
`https://esp32ms.000webhostapp.com`

Strona ta pozwala na wyświetlanie danych zgromadzonych w bazie danych MySQL. Wykresy pozwalają na obserwowanie danych z ostatnich 48h godzin, dodatkowo zostały umieszczone tam wskaźniki z ostatnimi wartościami z bazy danych.

# 3 Azure DevOps

W rozdziale tym znajduje się opis najważniejszych funkcji platformy Azure DevOps wykorzystanych w projekcie.

Link do projektu: `https://dev.azure.com/jwolowiec/ESP32`

## 3.1 Boards

Zakładka Boards służy do zarządzania zadaniami. Pozwala ona tworzyć tzw. Work Itemy, które z kolei moga być zadaniami(Task) lub bugami(Issue). Stworzone tak taski(Figure 4) mogą być później organizowane w Sprinty(Figure 5) jak ma to miejsce w Scrumie.

| ID | Title | Assigned To | State | Area Path | Tags | Comments |
|---|---|---|---|---|---|---|
| 9 | Creating own platform for visualizing data | Głogowski | Done | ESP32 | | 1 |
| 22 | Fix bugs on the website and in the database | Głogowski | Done | ESP32 | | |
| 21 | Implement suggested changes | Głogowski | Done | ESP32 | | |
| 23 | Create database for sensor data stores purpose | Głogowski | Done | ESP32 | | |
| 20 | Place working website on 000webhost.com/ | Głogowski | Done | ESP32 | | |
| 19 | Prepare website prototype on virtual server | Głogowski | Done | ESP32 | | |
| 3 | Zapoznanie ze środowiskami Azure i ESP IDF | Głogowski | Done | ESP32 | | |
| 18 | Add new data sink | Jan Wołowiec | Done | ESP32 | | 2 |
| 17 | Send data to ThingSpeak | Jan Wołowiec | Done | ESP32 | | 1 |
| 15 | Send data to thinger.io platform | Jan Wołowiec | Done | ESP32 | | 4 |
| 14 | Verify implemented functionalities | Jan Wołowiec | Done | ESP32 | | 2 |
| 13 | Add BME280 support | Jan Wołowiec | Done | ESP32 | | 1 |
| 16 | Code refactoring and improving wifi functions | Jan Wołowiec | Done | ESP32 | | 1 |
| 12 | Add ota update functionality | Jan Wołowiec | Done | ESP32 | | 2 |
| 1 | Przygotowanie specyfikacji projektu | Jan Wołowiec | Done | ESP32 | | |
| 11 | Add https client functionality to project | Jan Wołowiec | Done | ESP32 | | 1 |
| 2 | Przygotowanie listy zadań | Jan Wołowiec | Done | ESP32 | | |
| 10 | Create repository with basic esp32 project | Jan Wołowiec | Done | ESP32 | | |
| 8 | Thinger.io platform investigation & configuration | Jan Wołowiec | Done | ESP32 | | |
| 7 | Integrate webhosting with pipelines | Jan Wołowiec | Done | ESP32 | | |
| 6 | Configure webhosting for ota purposes | Głogowski | Done | ESP32 | | |
| 4 | Azure project configuration | Jan Wołowiec | Done | ESP32 | | |
| 5 | Pipeline configuration | Jan Wołowiec | Done | ESP32 | | |

Figure 4: Lista stworzonych tasków.

Tworzenie zadań i początkowe planowanie pozwala w późniejszych etapach projektu łatwiej dzielić pracę oraz estymować ile jej pozostało. Dodatkowo jedną z ważniejszych zalet stosowania Work Itemów jest możliwość śledzenia wprowadzanych zmian. Po ustawieniu wymogu, aby każdy Pull Request posiadał przypisanego taska, można łatwo sprawdzić okoliczności danego zadania np. poprzez dodane komentarze lub historię edycji danego zadania.

Sprinty z kolei pozwalają łatwo rozplanować pracę w danym tygodniu ustalając priorytety w ich wykonywaniu. Przy dwuosobowym zespole nie jest to konieczne, lecz w przypadku większych zespołów przynosi to już znaczącą korzyść poprawiając efektywność.

## 3.2 Repos

Repozytorium **GIT** znajdujące się na platformie **Azure DevOps**. Wszelkie wprowadzanie zmian powinno być realizowane na prywantych branch'ach, po czym merge powinien być wykonany z wykorzystaniem mechanizmu **Pull Request**(dalej zwanym **PR**). Dodatkowo celem utrzymania porządku na branchu main, możliwe typy merge zostały ograniczone do typu squash.

Pod linkiem `https://dev.azure.com/jwolowiec/_git/ESP32` można obejrzeć pliki projektu, dodatkowo Azure umożliwia edytowanie plików za pomocą interfejsu online.

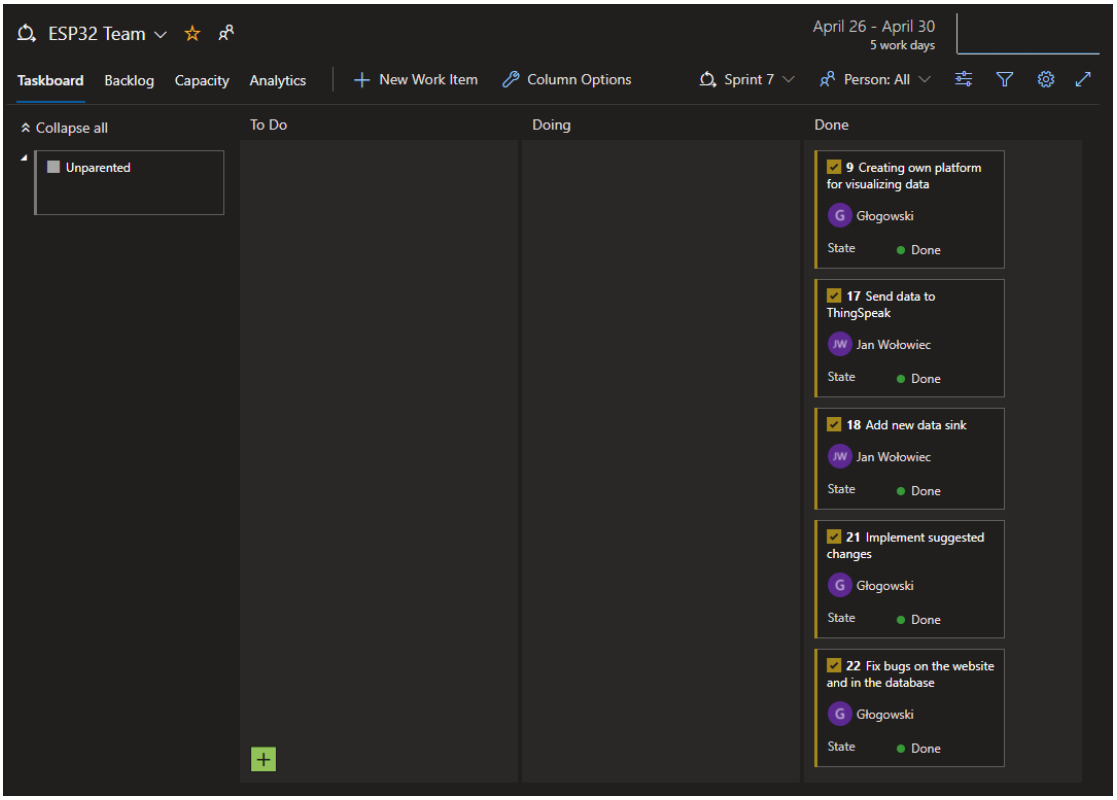


Figure 5: Widok pojedynczego sprintu.

Z kolei pod adresem `https://dev.azure.com/jwolowiec/ESP32/_git/ESP32/pullrequests` dostępne są PR.

Warunkami akceptacji PR są:

- Podlinkowanie taska związaniego z danym PR
- Rozwiązanie wszystkich komentarzy
- Sukcesywne wykonanie się pipeline walidacyjnego (PR_ESP32), celem wykrycia potencjanych błędów w kodzie

Na Figure 6 przedstawiono widok zmian wprowadzonych w danym PR, pozwala on na wykonanie Code Review poprzez wygodny widok zmian w mergowanym branchu oraz możliwość dodawania komentarzy do kodu. Możliwe jest ustawienie na przykład wymogu, aby 2 osoby zatwierdziły PR zanim możliwe będzie jego mergowanie lub dodanie wymogu pozytywnego ukończenia Pipeline'u dedykowanego do PR, co zostało opisane w podrozdziale 3.3. Pozwoli to podnieść jakość kodu poprzez ciągłą weryfikację wprowadzanych zmian.

## 3.3 Pipelines

Na platformie Azure DevOps zostały utworzone 2 pipeline'y:

- PR_ESP32
- CI_ESP32

Pipeline z przedrostkiem PR wykorzystywany jest do walidacji Pull Requestów przed mergowaniem ich do branch'a **main**. Z kolei pipeline z przedrostkiem CI aktywowany jest przy

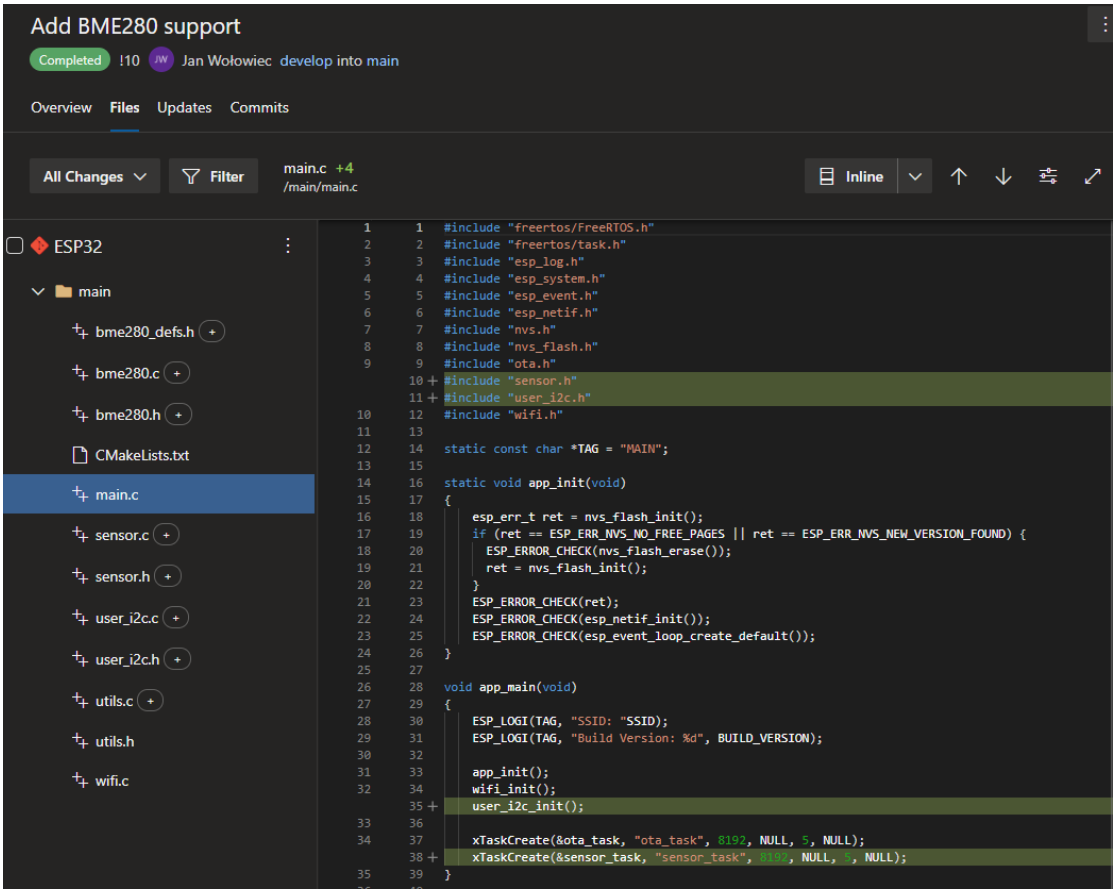


Figure 6: Widok zmian PR.

każdych zmianach wprowadzanych do brancha **main**, celem wygenerowania pliku binarnego służącego do zaprogramowania układu.

Konfiguracja pipeline'u przechowywana jest w pliku azure-pipelines.yml, pozwala ona skonfigurować jakie kolejno kroki będą wykonane. W naszym przypadku jest to:

1. Instalacja narzędzia Docker CLI
2. Uruchomienie obrazu Docker'a espressif/idf i wykonanie build'a ze specyfikowanymi parametrami takimi jak: `build_version`, `thinger_io_token`, `thingspeak_token`, `wifi_ssid`, `wifi_pass`
3. Przygotowanie wygenerowanych artefaktów, przeniesienie ich do odpowiedniego folderu oraz generacja pliku z wersją build'u
4. Jako dodatkowy krok pipeline'u CI_ESP32 wykonywane jest przesłanie artefaktów na hosting internetowy przechowujący dane do wykonania aktualizacji OTA za pomocą FTP
5. Opublikowanie artefaktów w systemie Azure Pipelines

Zmienne wykorzystowane w trakcie uruchamiania pipeline'ów:

- `wifi_ssid` - ssid sieci do której będzie się łączył układ ESP - **secret**
- `wifi_pass` - hasło sieci do której będzie się łączył układ ESP - **secret**
- `ftp_url` - link do serwera FTP

- `ftp_user` - nazwa użytkownika w serwerze FTP - **secret**
- `ftp_pass` - hasło użytkownika w serwerze FTP - **secret**
- `build_version` - wartość dostarczana przez sam pipeline pod zmienną `$(Build.BuildId)`
- `thinger_io_token` - token służący do autoryzacji na platformie Thinger.io - **secret**
- `thingspeak_token` - token służący do autoryzacji na platformie ThingSpeak - **secret**

Figure 7 pokazuje jak wygląda w praktyce działanie tak skonfigurowanego procesu. Po lewej stronie widać wykonane kroki oraz czas jaki zajęły, a po kliknięciu w nie logi z ich wykonania.

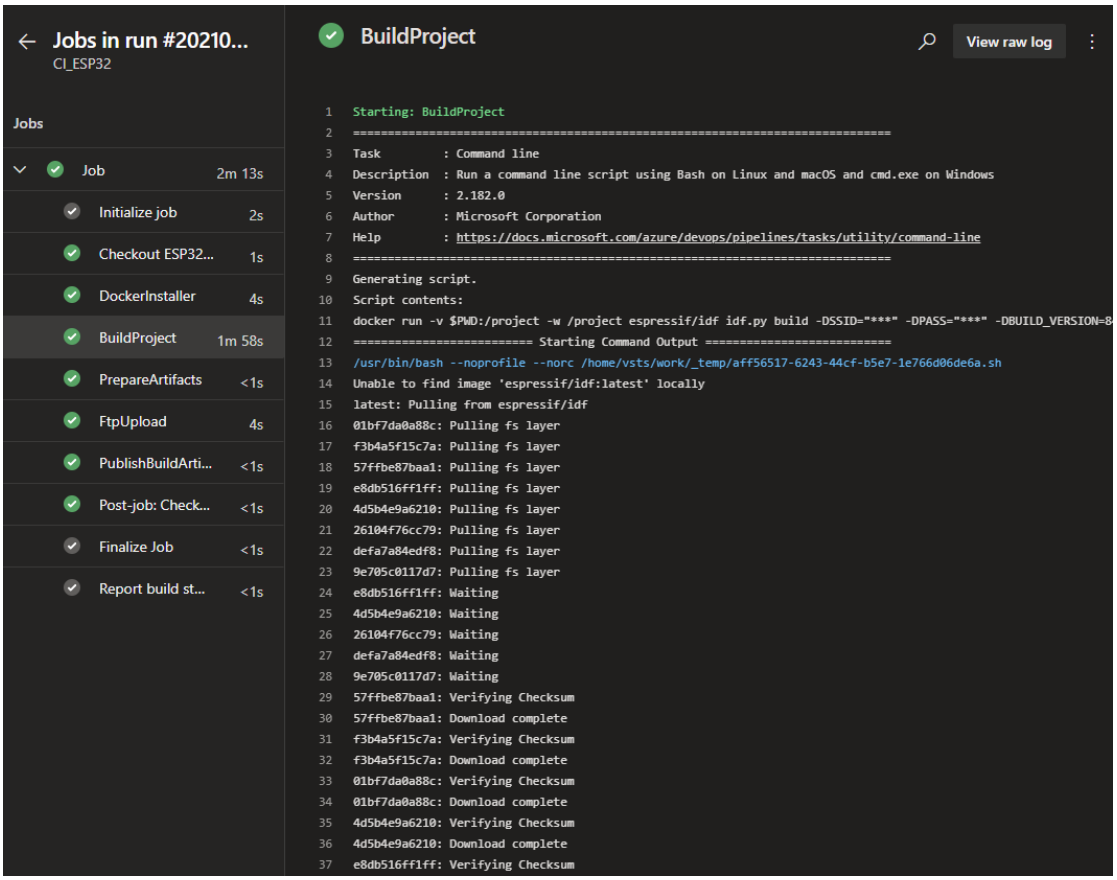


Figure 7: Szczegóły przebiegu danego pipeline'u.

# 4 Implementacja aplikacji na ESP

## 4.1 Taski

Napisana aplikacja została stworzona z użyciem dostarczanego przez producenta frameworka ESP-IDF, aplikacja wykorzystuje system FreeRTOS implementując 2 taski:

**sensor_task** - odpowiada za inicjalizację czujnika oraz okresowe odczyty i przekazywanie danych do odesłania.

**ota_task** - odpowiada za okresowe sprawdzanie wersji znajdujące się na hostingu i jeśli zostanie wykryta nowsza wersja przeprowadza aktualizację.

## 4.2 Parametryzacja

Aby uniknąć przechowywania tajnych danych takich jak hasła i tokeny do autentykacji są one przekazywane jako parametry w trakcie budowania, natomiast Azure przechowuje je w bezpiecznej formie, jako ukryte zmienne.

Aby umożliwić przekazywanie parametrów z użyciem pipeline oraz lokalnie, sprawdzana jest z pomocą CMake'a obecność danych zmiennych. Obsługa parametrów została umieszczona w pliku `main/CMakeLists.txt`.

```
1 if (DEFINED THINGER_IO_TOKEN)
2     add_definitions(-DTHINGER_IO_TOKEN="${THINGER_IO_TOKEN}")
3 else()
4     add_definitions(-DTHINGER_IO_TOKEN="")
5 endif()
```

Listing 1: Przykładowe sprawdzanie przekazywanych zmiennych

## 4.3 Pliki źródłowe

### 4.3.1 bme280.c

Sterownik do czujnika BME280, dostarczany przez BoschSensortec.
Źródło: `https://github.com/BoschSensortec/BME280_driver`

### 4.3.2 http.c

Wykonuję obsługę wychodzących i przychodzących zapytań HTTP. Zawiera funkcję `http_get` służącą do wykonywania zapytań GET.

Dodatkowo znajduje się tam funkcja `http_post_thinger_io`, wykonująca zapytanie POST specyficznie do platformy thinger.io ze względu na wymóg załączenia dodatkowych informacji w headerze.

Pozostałe platformy wykorzystują zapytania GET to transmisji danych.

### 4.3.3 main.c

Wykonuje inicjalizację wszystkich modułów oraz tworzy taski.

### 4.3.4 ota.c

Zawiera task `ota_task`, okresowo sprawdzający czy jest dostępna nowa wersja oprogramowania funkcją `ota_is_new_fw_available` i jeśli zwróci ona `true`, wywołana zostaje przeprowadzająca aktualizację funkcja `ota_perform_ota_update`.

#### 4.3.5 sensor.c

Zawiera task `sensor_task`, wykonujący okresowy odczyt danych i przeprowadzający ich weryfikację, po czym wysyłający dane za pomocą funkcji z modułu http.

Najważniejsze funkcje znajdujące się w tym module to:

`sensor_init` – Wykonuje inicjalizację czujnika wykorzystując funkcję udostępniane w sterowniku BME280 dostarczanym przez BoschSensortec.

`sensor_perform_measurement` – Ustawia sensor w tryb forced mode, który rozpoczyna pomiar, następnie po odczekaniu wymaganego czasu następuje odczyt zebranych danych.

`sensor_verify_measurement` – Z racji zdarzających się błędnych odczytów, zbyt niskiego ciśnienia lub jego zbyt szybkich zmian sprawdzana jest wartość ciśnienia oraz jak duża nastąpiła zmiana od ostatniego pomiaru. W razie zbyt dużej różnicy względem poprzedniego pomiaru następuje 5 prób ponownego pomiaru, lecz jeśli również znajdują się one poza założoną histerezą $\pm 0.2hPa$, uznawane są za prawidłowy pomiar.

`sensor_task` – Wywołuje inicjalizację sensora `sensor_init`, a następnie wykonuje okresowy pomiar z użyciem funkcji `sensor_handle_measurement`, która łączy w sobie pomiar i jego weryfikację. Jeśli pomiar jest prawidłowy następuje generowanie odpowiednich zapytań, które będą przesłane do użytych platform.

W przypadku naszej platformy adres jest generowany tak jak zostało to pokazane w listingu 2.

```
1  static const char *CUSTOM_URL = "http://esp32ms.000webhostapp.com/add_data.php?temp=%.2f&humi=%.2f&pres=%.2f";
2
3  static void sensor_get_custom_url_from_data(struct bme280_data *comp_data, char *buffer)
4  {
5      sprintf(buffer, CUSTOM_URL, comp_data->temperature, comp_data->humidity, comp_data->pressure);
6      ESP_LOGI(TAG, "URL: %s", buffer);
7  }
```

Listing 2: Generowanie adresu z query wykorzystywanego w zapytaniu GET

#### 4.3.6 user_i2c.c

Zawiera funkcje do inicjalizacji, zapisu oraz odczytu przez I2C. Funkcja `user_i2c_read` i funkcja `user_i2c_write` są przekazywane jako callback do sterownika BME280.

#### 4.3.7 utils.c

Zawiera funkcję `utils_delay_us` przekazywaną jako callback do sterownika BME280.

#### 4.3.8 wifi.c

Zawiera funkcję `wifi_init` odpowiadającą za inicjalizację WIFI oraz nawiązanie połączenia.

# 5 Strona internetowa

## 5.1 Platforma do wizualizacji danych

Rdzeń strony przeznaczonej do obrazowania danych wynikowych pomiaru został wykonany przy pomocy prostej struktury blokowej w HTML. Przy projektowaniu witryny zastosowano styl minimalistyczny z przewagą mniej męczącego oczy ciemnego motywu. Kolorystykę i pozycjonowanie poszczególnych sekcji ustawiono przy użyciu podstawowych dyrektyw kaskadowego arkusza stylów (CSS). Wszelkie dynamiczne elementy na stronie wykonano poprzez funkcje napisane w języku JavaScript.

Wraz z uruchomieniem witryny wywołane zostają funkcje tworzące wykresy przedstawiające ostatnie wartości zarejestrowane przez czujnik. Wczytane i wyświetlone zostają również najbardziej aktualne dane oraz uruchomiona zostaje metoda *setInterval* odświeżająca te wartości cyklicznie co 5 sekund. Wizualizacja zebranych danych odbywa się przy pomocy wykresów *Highcharts* aktualizowanych również z interwałem 5 sekundowym. Wykresy te utworzono w poszczególnych wywołaniach funkcji *create_charts*. Do każdego z wykresów przypisana została odpowiednia sekcja. Przekazany do funkcji tworzących argument odpowiada selektorowi ID poszczególnych bloków.

```
1  $(function(){
2    // create dynamic charts
3    create_charts('temp');
4    create_charts('humi');
5    create_charts('pres');
6
7    // set newest data and their refresh time
8    setInterval( reload_current_values, 5000 );
9    reload_current_values();
10 });
```

Listing 1: Wprowadzenie dynamicznych zmian po wczytaniu strony

Aby umożliwić dynamiczną komunikację z bazą danych wykorzystano asynchroniczne techniki jakimi są zapytania AJAX. Dzięki temu w celu doczytania najnowszych danych nie jest wymagane odświeżenie strony. Zapytania AJAX zostały umieszczone w oddzielnych funkcjach, których celem jest wyłącznie wykorzystanie pliku służącego do połączenia z bazą danych i przekazanie dalej wyniku działania. Połączenie z bazą danych napisane w języku PHP przebiega w standardowy sposób, a wynikiem zapytań (*query*) do bazy danych MySQL są najnowsze próbki danych lub tablica próbek zbieranych przez ostanie 48 godzin. Baza danych została utworzona na hostingu używającym strefy czasowej UTC, a domyślny użytkownik znajduje się w strefie UTC+2:00. Tą różnicę uwzględniono w query. Wyniki zwracane przez serwer są konwertowane do formatu JSON, tak aby funkcje używające AJAX mogły swobodnie korzystać z pobranych wartości.

```
1  function all_newest_data_call(){
2    return $.ajax({
3    url: "newest_data.php",
4    dataType: "json"
5    });
6  }
7
8  function previous_data_call(){
9    return $.ajax({
10   url: "previous_data.php",
11   dataType: "json"
12   });
13 }
```

Listing 2: Wczytywanie danych przy pomocy technik AJAX

```php
<?php
$servername = "localhost";
$username = "id16483562_esp32ms";
$password = "**************";
$dbname = "id16483562_test";

$conn = new mysqli( $servername, $username, $password, $dbname );

$RESULT_NOW = $conn -> query("SELECT `date`, `temp`, `humi`, `pres` FROM `000
    _test_czujnik` WHERE `date` BETWEEN CURRENT_TIMESTAMP - 0000100 + 0020000 AND
     CURRENT_TIMESTAMP - 0000000 + 0020000") -> fetch_assoc();

echo json_encode($RESULT_NOW);

$conn -> close();
?>
```

Listing 3: newest_data.php

```php
<?php
$servername = "localhost";
$username = "id16483562_esp32ms";
$password = "**************";
$dbname = "id16483562_test";

$conn = new mysqli( $servername, $username, $password, $dbname );

$RESULT_SQL = $conn -> query("SELECT `date`, `temp`, `humi`, `pres` FROM `000
    _test_czujnik` WHERE `date` BETWEEN CURRENT_TIMESTAMP - 2000000 + 0020000 AND
     CURRENT_TIMESTAMP - 0000000 + 0020000");

$RESULT = array();

while( $ROW = $RESULT_SQL -> fetch_assoc() ){
  array_push( $RESULT, $ROW );
}

echo json_encode($RESULT_NOW);

$conn -> close();
?>
```

Listing 4: previous_data.php

Przykładem korzystania z AJAX jest funkcja *reload_current_values*, która jest używana przy starcie strony i jako funkcja uruchamiana przez *setInterval*, a służy ona do odświeżania treści strony o najnowsze wartości odczytane przez czujnik. Funkcja ta używa zapytania AJAX, które po zakończeniu działania uruchamia metodę *done*. Wartości odczytane z bazy danych przekazywane są dalej do funkcji, gdzie wykorzystywane są do edycji zawartości strony.

```javascript
function reload_current_values(){
  all_newest_data_call().done( function(data){

  var temp = 'TEMPERATURE [C]<div class="number">' + data.temp + '</div>',
    humi = 'HUMIDITY [%]<div class="number">' + data.humi + '</div>',
    pres = 'PRESSURE [hPa]<div class="number">' + data.pres + '</div>';

  document.getElementById("temp").innerHTML = temp;
  document.getElementById("humi").innerHTML = humi;
  document.getElementById("pres").innerHTML = pres;
  } );
}
```

Listing 5: Wypisywanie najnowszych wartości na stronie

## 5.2 Dodawanie wartości do bazy danych

W celu aktualizacji informacji dotyczących danych zbieranych przez czujnik wyświetlanych na utworzonej stronie postanowiono wysyłać gromadzone wartości do bazy danych utworzonej na wykorzystywanym hostingu. Kolejne próbki zebranych wartości są przekazywane poprzez **query** do pliku *add_ data.php*. Plik ten nawiązuje połączenie z serwerem MySQL, a następnie przygotowuje instrukcję SQL mającą za zadanie dodanie rekordu z pobranymi danymi. Po powiązaniu wartości uzyskanych z zapytania przy pomocy metody **GET** do instrukcji SQL następuje jej wykonanie.

```php
<?php
$servername = "localhost";
$username = "id16483562_esp32ms";
$password = "**************";
$dbname = "id16483562_test";

$conn = new mysqli( $servername, $username, $password, $dbname );

if( $conn -> connect_error ){
  die( "Connection failed: " . $conn -> connect_error );
}

$sql = "INSERT INTO `000_test_czujnik`(`date`, `temp`, `humi`, `pres`) VALUES (
    CURRENT_TIMESTAMP + 20000, ?, ?, ? )";
$stmt = $conn -> prepare($sql);

$stmt -> bind_param( "sss",  $_GET['temp'], $_GET['humi'], $_GET['pres'] );
$stmt -> execute();

$conn -> close();
?>
```

Listing 6: add_data.php

# 6 Podsumowanie

## 6.1 Wykorzystane narzędzia

- Git
- ESP-IDF
- Visual Studio Code
- Docker
- WinScp
- Draw.io
- Overleaf

## 6.2 Zakres wykonanych prac

Zgodnie z założeniem udało się wykonać wszystkie zaplanowane zadania. Główne bloki składowe systemu wykonują poprawnie założone przez nas zadania:

**Azure** – obsługuje taski, pull requesty i budowę pipeline'ów.

**Device** – wykonuje okresowe pomiary oraz okresowo sprawdza możliwość aktualizacji.

**Web Hosting** – przechowuje pliki wymagane do aktualizacji OTA oraz wyświetla zebrane dane.