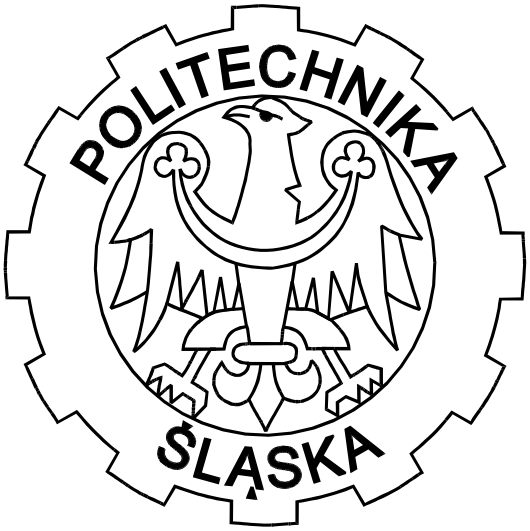


POLITECHNIKA ŚLĄSKA
WYDZIAŁ ELEKTRYCZNY
Katedra Mechatroniki – Wydział Elektryczny

PRACA MAGISTERSKA

Zastosowanie Raspberry Pi w systemach IoT

The use of Raspberry Pi in IoT systems

Student: **Inż. Zbigniew Adam SROCZYŃSKI**
Nr albumu: 282025

Studia: Stacjonarne II stopnia
Kierunek: Mechatronika
Specjalność: Mechatronika

Promotor: Dr. Inż. Krzysztof KONOPKA

Gliwice 2023

Tytuł pracy: Zastosowanie Raspberry Pi w systemach IoT

Streszczenie: Praca obejmuje utworzenie elementu automatyki budynkowej. Jest to system inteligentnego sterowania roletami okiennymi. Wykorzystano dwa ESP32 wraz z czujnikami temperatury i naświetlenia. Dane są wysyłane przez sieć WiFi protokołem MQTT do Raspberry Pi która gromadzi dane, wyświetla na stronie internetowej i steruje pracą okiennej rolety.

Słowa kluczowe: Raspberry Pi, IoT, inteligentny system

Thesis title: The use of Raspberry Pi in IoT systems

Abstract: The work includes creating an element of building automation. It is an intelligent control system for window blinds. Two ESP32s were used along with temperature and exposure sensors. The data is sent via the WiFi network using the MQTT protocol to the Raspberry Pi, which collects the data, displays it on the website and controls the operation of the window roller shutter.

Keywords: Raspberry Pi, IoT, intelligent system

Spis treści

1. Wstęp

Internet Rzeczy (IoT) to koncepcja, która opiera się na połączeniu urządzeń ze sobą i z Internetem w celu umożliwienia komunikacji, zbierania danych i sterowania nimi na odległość. Zastosowania IoT są nieograniczone i obejmują wiele różnych sektorów, takich jak przemysł, rolnictwo, opieka zdrowotna, energetyka czy inteligentne domy.

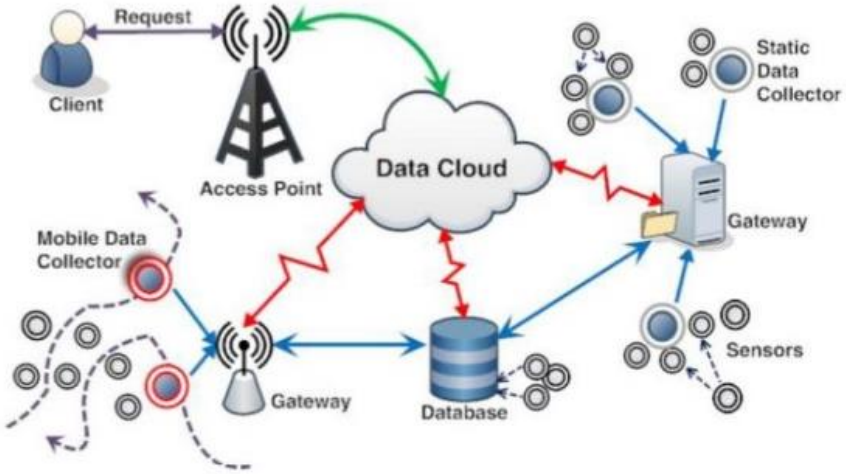
W przemyśle IoT jest wykorzystywane w celu automatyzacji procesów produkcyjnych i monitorowania stanu maszyn. Urządzenia IoT, takie jak czujniki, akcelerometry czy kamery, zbierają dane na temat temperatury, wilgotności, wibracji czy obciążenia maszyn, co umożliwia ich szybkie wykrycie i naprawę przed wystąpieniem awarii. W ten sposób można również zoptymalizować zużycie energii i surowców, co przekłada się na zmniejszenie kosztów produkcji.

W inteligentnych domach IoT umożliwia sterowanie urządzeniami domowymi, takimi jak oświetlenie, klimatyzacja czy zamki drzwi. Dzięki temu można zwiększyć bezpieczeństwo domu i zmniejszyć zużycie energii. W energetyce IoT umożliwia monitorowanie i zarządzanie siecią energetyczną na odległość. Dzięki temu można zoptymalizować zużycie energii, przewidywać awarie i szybko reagować na nie, co zwiększa niezawodność dostaw energii.

Systemy IoT to dziedzina technologii która prężnie się rozwija w różnych dziedzinach naszego życia. Takich systemów będzie coraz więcej, za sprawą wykorzystania tych systemów w rozwiązaniach inteligentnych domów, inteligentnych miast, pojazdów itd.

1.1. Systemy IoT

Systemy IoT to bardzo rozległe pojęcie które zawiera technologie z zakresu: informatyki, elektroniki, automatyki [1][2][3]. Jest to koncepcja, która opiera się na połączeniu urządzeń ze sobą i z Internetem w celu umożliwienia komunikacji, zbierania danych i sterowania nimi na odległość. Nowoczesne systemy zawierają technologie typu sztuczna inteligencja, bazy danych czy umiejętność tworzenia stron internetowych. Wiedza na temat hardware także okaże się bardzo przydatna. To rozległe pojęcie (systemy IoT) wymaga tak naprawdę wielu specjalistów w różnych dziedzinach.



Rysunek 1. Poglądowy rysunek sieci IoT [1].

Omawiane systemy działają w sieciach bezprzewodowych, takich jak WiFi, Bluetooth czy Zigbee, umożliwiają bezprzewodową komunikację między urządzeniami. Dzięki temu możliwe jest zbieranie danych z różnych czujników i urządzeń oraz sterowanie nimi na odległość. Czujniki są kluczowymi elementami systemów IoT. Dzięki nim możliwe jest zbieranie danych na temat temperatury, wilgotności, wibracji czy obciążenia maszyn, co umożliwia ich szybkie wykrycie i naprawę przed wystąpieniem awarii.

Sztuczna inteligencja (AI) jest ważnym elementem systemów IoT, ponieważ umożliwia analizę dużych ilości danych i automatyczne podejmowanie decyzji. AI jest wykorzystywana w systemach monitoringu, zarządzania energią czy w inteligentnych domach, gdzie na podstawie analizy zachowań mieszkańców można dostosować ustawienia urządzeń domowych.

Automatyzacja to kolejna składowa technologii w systemach IoT. Systemy IoT umożliwiają automatyzację różnych procesów, co pozwala na zmniejszenie kosztów i zwiększenie efektywności. Przykładowo, w przemyśle IoT umożliwia automatyzację procesów produkcyjnych, co pozwala na zmniejszenie kosztów produkcji i zwiększenie szybkości produkcji.

Bazy danych to kolejna składowa technologii w systemach IoT. Dzięki bazom danych możliwe jest gromadzenie i przetwarzanie dużych ilości danych pochodzących z różnych źródeł. Bazy danych umożliwiają również analizę danych i podejmowanie na ich podstawie decyzji, co jest kluczowe w systemach IoT.

Wszystkie te składowe technologii w systemach IoT działają razem, umożliwiając efektywne zbieranie i przetwarzanie danych oraz sterowanie urządzeniami na odległość. Systemy IoT mają duży potencjał w poprawie efektywności i jakości życia, a ich wykorzystanie będzie coraz bardziej popularne w przyszłości.

2. Cel projektu

Celem pracy magisterskiej było utworzenie rozwiązania dla inteligentnego domu. Docelowo ma to być system sterowania roletami okiennymi w domu. Dane z czujników (czujnik temperatury, czujnik natężenia światła) będą wysyłane przez dwa urządzenia ESP8266 do Raspberry Pi 4. Jedno ESP na zewnątrz okna, drugie w środku pokoju. Komunikacja między urządzeniami przez sieć WiFi, protokół MQTT, broker Mosquitto.

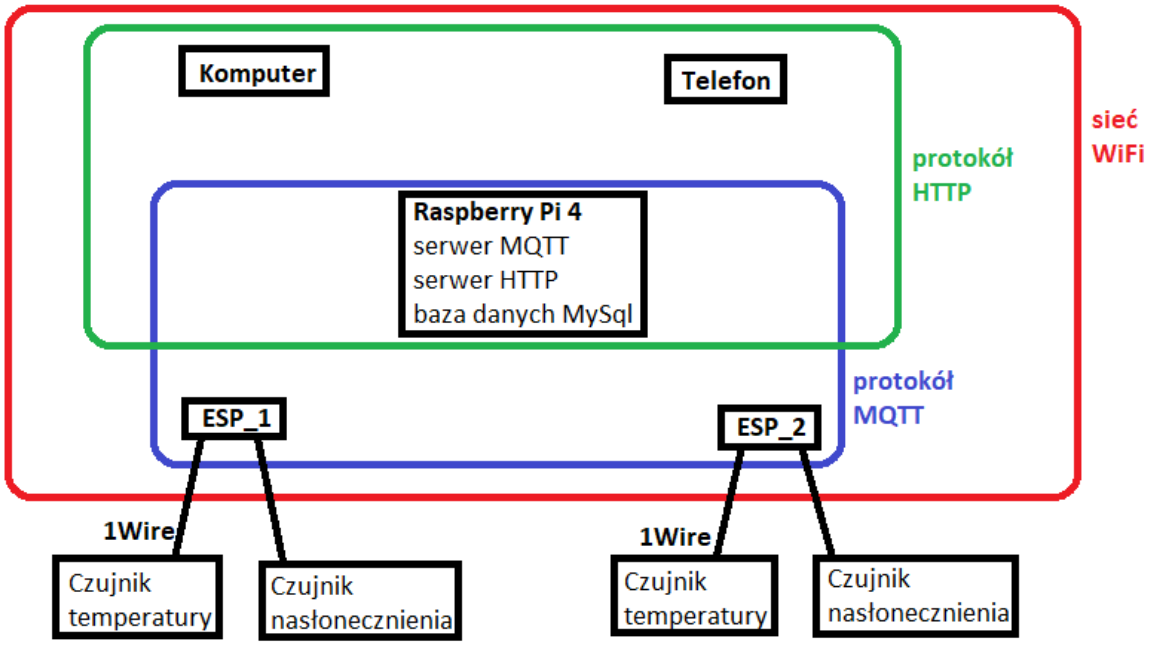
ESP to mały układ zasilany napięciem 5V, posiada wbudowane modułu do komunikacji z siecią WiFi i Bluetooth oraz wyprowadzenia GPIO. Układ programowalny m.in. w językach Arduino lub C.

Na Raspberry Pi znajduje się serwer WWW Apache2 i Django, protokół HTTP. Na serwerze Apache2 znajduje się baza danych MySql. Na serwerze Django znajduje się serwis internetowy wykonany we frameworku Django. Dane z czujników odbierane są przez Raspberry Pi wykorzystując protokół MQTT, następnie wysyłane są do bazy danych znajdującej się na serwerze HTTP.

Utworzona jest strona internetowa na serwerze Django, która pobiera dane z bazy danych. Na stronie zamieszczone są wykresy temperatury i nasłonecznienia, przedstawia dane w przystępnej formie dla użytkownika. Z poziomu dowolnego urządzenia (komputer, telefon) połączony do lokalnej sieci WiFi w której znajduje się Raspberry Pi 4, mamy dostęp do strony internetowej.

Składowe elementy projektu:

- **Wykorzystane urządzenia:** Raspberry Pi 4, ESP8266
- **Wykorzystana sieć:** WiFi
- **Wykorzystane protokoły sieciowe:** MQTT, HTTP
- **Utworzone serwery:** Django, Apache2, Mosquitto
- **Wykorzystana baza danych:** MySql
- **Wykorzystane czujniki:** DS18B20, dzielnik napięcia z fotorezystorem
- **Wykorzystana magistrala szeregowa:** 1Wire
- **Wykorzystane języki programowania:** Python, Arduino, SQL, PHP, HTML, CSS, JS



Rysunek 2. Schemat poglądowy utworzonego projektu.

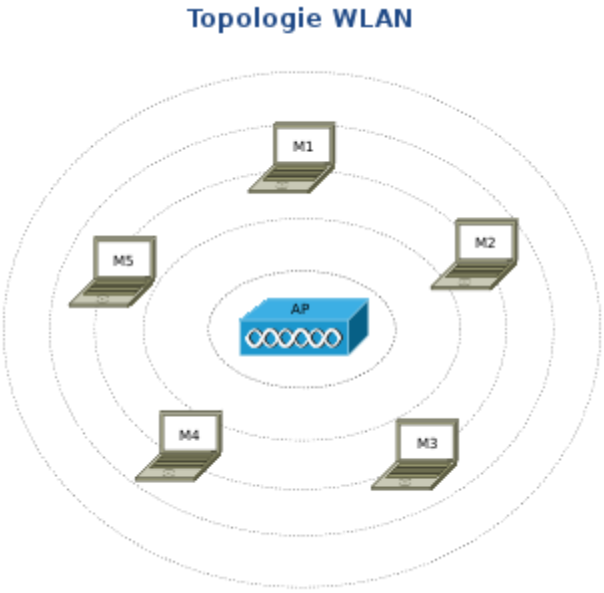
3. Rozważania projektowe

Podczas realizacji projektu należało wybrać urządzenia które będą odpowiednie. Należało dokonać przeglądu istniejących rozwiązań na rynku. Dokonano wyboru sieci, protokołów, serwerów. W dalszej części rozdziału zostaną przedstawione rozważania projektowe.

3.1. Przegląd sieci bezprzewodowych - Sieć WiFi

Początkowo dokonano przeglądu dostępnych rozwiązań bezprzewodowych [4][5][6]. Istnieje kilka rodzajów sieci bezprzewodowej, z których najpopularniejsze to:

1. **Wi-Fi** - sieć bezprzewodowa wykorzystująca standard IEEE 802.11, która pozwala na przesyłanie danych w zakresie od kilku do kilkudziesięciu metrów od punktu dostępu do sieci (AP).
2. **Bluetooth** - technologia bezprzewodowa wykorzystywana głównie do połączenia urządzeń mobilnych, takich jak smartfony, słuchawki, głośniki czy też klawiatury, na krótkich dystansach, zazwyczaj do 10 metrów.
3. **ZigBee** - niskonapięciowa sieć bezprzewodowa wykorzystywana do komunikacji w systemach automatyki budynkowej, przemysłowej i kontrolno-pomiarowej.
4. **NFC (Near Field Communication)** - technologia bezprzewodowa stosowana do przesyłania danych na krótkich dystansach, zazwyczaj do 10 centymetrów, np. do przeprowadzania płatności zbliżeniowych.
5. **LTE (Long-Term Evolution)** - technologia bezprzewodowa wykorzystywana w sieciach telefonii komórkowej, która pozwala na szybką transmisję danych na duże odległości [16].
6. **WiMAX** - bezprzewodowa technologia transmisji danych na długie dystanse, wykorzystywana głównie do dostępu do internetu na obszarach wiejskich lub w miejscach, gdzie brakuje infrastruktury kablowej [15].
7. **LoRa (Long Range)** - technologia bezprzewodowa, która umożliwia transmisję danych na bardzo duże odległości, przy jednoczesnym niskim poborze energii, co jest przydatne np. w systemach IoT (Internet of Things) [14].



Rysunek 3. Działanie sieci WiFi [18].

Sieć WiFi [7] jest jednym z najpopularniejszych rozwiązań sieci bezprzewodowych, które znajduje zastosowanie w różnych projektach, w tym także w projektach związanych z Internetem rzeczy (IoT). W tym wypracowaniu przedstawię argumenty, które skłoniły do wyboru sieci WiFi z dostępnych rozwiązań w kontekście projektu IoT.

Kolejnym ważnym argumentem przemawiającym na korzyść sieci WiFi w kontekście projektu IoT jest jej łatwość w instalacji i konfiguracji. Wiele urządzeń IoT posiada już wbudowany moduł WiFi, co sprawia, że połączenie z siecią jest proste i wymaga jedynie podania hasła do sieci. Dzięki temu możliwe jest szybkie wdrożenie systemu IoT bez konieczności stosowania skomplikowanych konfiguracji i dodatkowych urządzeń.

Kolejnym argumentem na korzyść sieci WiFi w kontekście projektu IoT jest jej uniwersalność. Sieć WiFi obsługuje wiele protokołów i standardów, co umożliwia korzystanie z różnych urządzeń IoT bez konieczności zmiany sposobu połączenia. Dzięki temu, system IoT oparty na sieci WiFi może korzystać z różnych sensorów, modułów bezprzewodowych czy innych urządzeń, bez konieczności stosowania różnych sposobów połączenia.

Podsumowując: Na podstawie dostępnych rozwiązań, zdecydowano wybrać się sieć WiFi do omawianego projektu. Sieć WiFi jest często i powszechnie wykorzystywana. Jest też uniwersalna, bo z powodzeniem będą działać na niej projekty IoT ale także serwisy internetowe lub bazodanowe serwery.

Podstawowe parametry sieci WiF [7]i:

1. **Przepustowość**: Przepustowość określa maksymalną ilość danych, które mogą być przesłane przez sieć w ciągu jednostki czasu. Współczesne standardy sieci WiFi, takie jak 802.11ac i 802.11ax, oferują przepustowość do kilku gigabitów na sekundę.

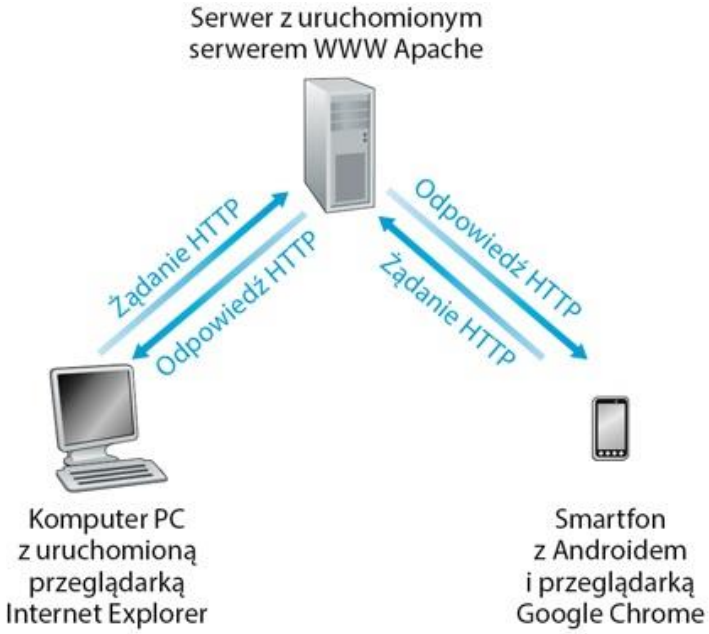
2. **Częstotliwość**: Sieć WiFi działa na różnych częstotliwościach, zależnie od zastosowania. Najpopularniejsze częstotliwości to 2,4 GHz i 5 GHz. Częstotliwość 2,4 GHz jest popularniejsza ze względu na większy zasięg, ale oferuje niższą przepustowość niż częstotliwość 5 GHz.
3. **Zasięg**: Zasięg sieci WiFi zależy od wielu czynników, takich jak siła sygnału, przeszkody w środowisku, rodzaj anteny itp. Współczesne standardy sieci WiFi oferują zasięg do kilkudziesięciu metrów.
4. **Bezpieczeństwo**: Bezpieczeństwo sieci WiFi jest kluczowe ze względu na to, że jest ona bezprzewodowa i potencjalnie podatna na ataki. Współczesne standardy sieci WiFi, takie jak WPA2, oferują wysoki poziom bezpieczeństwa dzięki szyfrowaniu danych i autentykacji.
5. **Przepływność**: Przepływność sieci WiFi zależy od ilości użytkowników korzystających z sieci jednocześnie. Im więcej użytkowników, tym mniejsza przepływność. Przepływność może być również ograniczona przez sieć i urządzenia w niej działające.
6. **Latencja**: Latencja to czas opóźnienia między wysłaniem danych z jednego urządzenia a odbiorem ich przez drugie urządzenie. Latencja w sieci WiFi zależy od wielu czynników, takich jak przepustowość, zasięg i obciążenie sieci.
7. **Liczba kanałów**: Sieć WiFi działa na różnych kanałach, które dzielą dostępną przepustowość. Im więcej kanałów, tym więcej urządzeń może korzystać z sieci jednocześnie, co zwiększa przepływność.

3.2. Protokół HTTP

Dokonano przeglądu dostępnych protokołów komunikacyjnych dostępnych w sieci WIFi

1. **MQTT (Message Queuing Telemetry Transport):** Protokół ten jest często stosowany w urządzeniach IoT w sieciach WiFi, ze względu na swoją lekkość i niezawodność. MQTT jest protokołem opartym na subskrypcji i publikacji, który pozwala na szybkie i niezawodne przesyłanie danych między urządzeniami w sieciach WiFi [9].
2. **CoAP (Constrained Application Protocol):** Protokół ten jest opracowany z myślą o urządzeniach z ograniczoną mocą obliczeniową i niskim zużyciem energii, ale jest też stosowany w sieciach WiFi. CoAP wykorzystuje protokół UDP (User Datagram Protocol) i zapewnia niskie opóźnienia i niskie zużycie energii [12].
3. **http (Hypertext Transfer Protocol):** Protokół ten jest dobrze znany i powszechnie stosowany w internecie, ale jest też wykorzystywany w sieciach WiFi. W sieciach IoT w sieciach WhttpHTTP może być wykorzystywany do komunikacji między urządzeniami a aplikacjami internetowymi[10][11].
4. **WebSockets:** Protokół ten umożliwia dwukierunkową komunikację między urządzeniami i aplikacjami w czasie rzeczywistym w sieciach WiFi. WebSockets wykorzystują protokół HTTP, ale umożliwiają trwałe połączenie między urządzeniem a serwerem[8].

5. **Wi-Fi Direct:** Protokół ten umożliwia bezpośrednie połączenie między dwoma urządzeniami w sieci WiFi bez potrzeby korzystania z punktu dostępowego. Wi-Fi Direct może być stosowany w urządzeniach IoT do bezpośredniej komunikacji między nimi [13].



Rysunek 4. Model protokołu HTTP [19]

HTTP , czyli Hypertext Transfer Protocol [19], to jeden z najczęściej używanych protokołów do przesyłania danych w Internecie. Jest on oparty na modelu klient-serwer, co oznacza, że dane są przesyłane między klientem (np. przeglądarką internetową) a serwerem (np. serwerem Apache2). W projekcie dotyczącym IoT, gdzie dane są przesyłane między urządzeniami w sieci lokalnej, protokół HTTP może być użyty do przesyłania danych z urządzeń IoT do serwera Apache2 i bazy danych MySql.

Przede wszystkim, protokół HTTP jest bardzo popularny i powszechnie stosowany w Internecie. Wiele aplikacji internetowych, takich jak strony internetowe, serwisy społecznościowe, sklepy internetowe i aplikacje mobilne, korzysta z protokołu HTTP. Dlatego wykorzystanie HTTP w projekcie IoT może ułatwić integrację z innymi systemami i narzędziami.

Kolejnym powodem, dla którego protokół HTTP jest dobrym wyborem w projekcie IoT, jest to, że jest on bardzo prosty w użyciu i łatwy do zrozumienia. Dzięki temu programiści mogą szybko nauczyć się korzystać z tego protokołu i łatwo wdrożyć go w swoich projektach.

Protokół HTTP jest również wysoce konfigurowalny i elastyczny, co oznacza, że można go dostosować do różnych wymagań i potrzeb. Na przykład, w projekcie IoT można skonfigurować serwer Apache2 tak, aby działał jako serwer HTTP i obsługiwał żądania HTTP przychodzące z urządzeń IoT.

W końcu, protokół HTTP oferuje wiele narzędzi i bibliotek, które ułatwiają jego stosowanie w projektach. Istnieją różne biblioteki klientów HTTP, takie jak Requests w języku Python, które ułatwiają wysyłanie żądań HTTP i odbieranie odpowiedzi. Istnieją również biblioteki serwerów HTTP, takie jak Flask i Django w języku Python, które ułatwiają tworzenie serwerów HTTP i obsługę żądań HTTP.

Podsumowując, protokół HTTP jest dobrym wyborem w projekcie IoT, w którym dane są przesyłane między urządzeniami IoT a serwerem Apache2 i bazą danych MySql. Jest popularny, prosty w użyciu, elastyczny i oferuje wiele narzędzi i bibliotek, które ułatwiają jego stosowanie w projektach.

Podsumowując: Na podstawie ww. wymienionych rozwiązań, zdecydowano się na wykorzystanie protokołów HTTP i MQTT. Protokół HTTP będzie wykorzystywany do obsługi połączenia klient-serwer. Klientem będzie urządzenie dostępne w lokalnej sieci WiFi (telefon, komputer itp.). Serwerem będzie serwer Apache2 utworzony na Raspberry Pi. Na serwerze będą znajdować się strony internetowe oraz baza danych MySql.

Parametry protokołu http [10]:

1. Metoda (ang. method): określa rodzaj żądania, jakie zostanie przesłane do serwera. Najczęściej stosowanymi metodami są GET i POST.
2. Nagłówki (ang. headers): zawierają dodatkowe informacje na temat żądania, takie jak typ MIME, rozmiar pliku, dane uwierzytelniające itp.
3. Ciało (ang. body): opcjonalna część żądania, w której przesyłane są dane.
4. Kod odpowiedzi (ang. status code): informuje o tym, czy żądanie zostało pomyślnie wykonane czy też nie, i określa typ błędu, jeśli taki wystąpił.
5. Zawartość odpowiedzi (ang. response body): dane przesłane przez serwer w odpowiedzi na żądanie.
6. Wersja protokołu (ang. protocol version): wskazuje, jakiej wersji protokołu HTTP używa żądanie i odpowiedź.

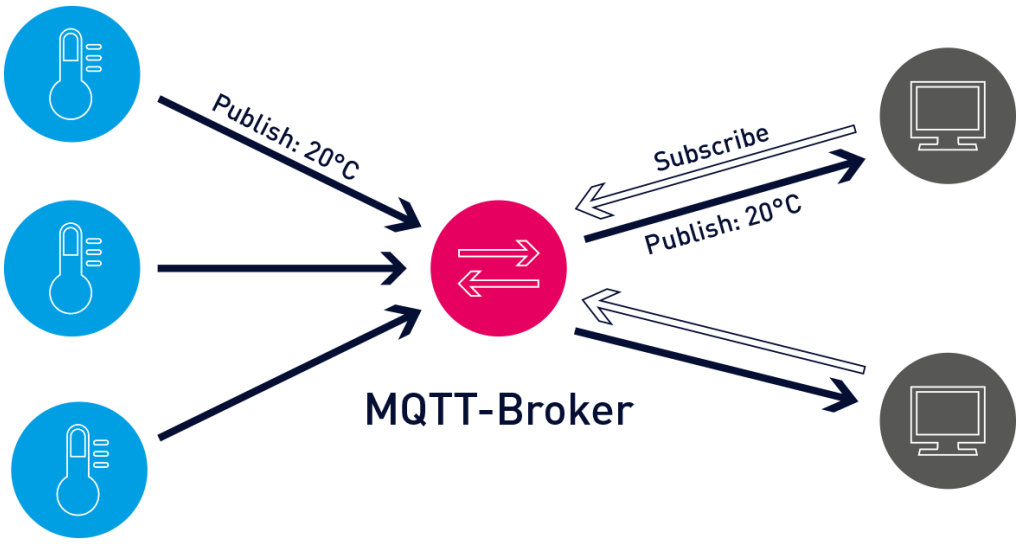
3.3. Protokół MQTT

Protokół MQTT [9][20] (Message Queue Telemetry Transport) to protokół komunikacyjny, który jest stosowany w sieciach IoT do przesyłania danych między urządzeniami. Główną zaletą protokołu MQTT jest jego lekkość i szybkość, co sprawia, że jest idealnym wyborem do przesyłania danych w czasie rzeczywistym.

W przypadku projektu IoT, w którym wykorzystywane są urządzenia ESP8266 i Raspberry Pi 4, protokół MQTT był dobrym wyborem ze względu na jego niskie wymagania dotyczące zasobów sprzętowych, które są szczególnie ważne w przypadku urządzeń mikroprocesorowych takich jak ESP8266. Protokół ten działa na niskim poziomie i nie wymaga dużych ilości pamięci i mocy obliczeniowej, co pozwala na szybsze i bardziej wydajne przesyłanie danych.

W projekcie IoT, w którym przesyłane są dane między urządzeniami, ważne jest również zapewnienie bezpieczeństwa przesyłanych informacji. Protokół MQTT oferuje kilka możliwości w zakresie zabezpieczeń, w tym szyfrowanie SSL/TLS, uwierzytelnienie użytkownika i hasła oraz kontrole dostępu na poziomie użytkownika. To sprawia, że protokół ten jest idealnym wyborem do zastosowań wymagających wysokiego poziomu bezpieczeństwa, takich jak w przypadku IoT.

Oprócz tego, protokół MQTT oferuje wiele funkcjonalności, które są przydatne w przypadku przesyłania danych w IoT, takich jak kontrola jakości usługi (QoS), co zapewnia dostarczenie danych w określonym czasie i z określoną jakością, a także możliwość publikowania i subskrybowania tematów. To wszystko umożliwia bardziej elastyczne zarządzanie przesyłanymi danymi, co jest niezbędne w projektach IoT.



Rysunek 5. Model protokołu MQTT [28].

Podsumowując: wybór protokołu MQTT jako rozwiązania do przesyłania danych między urządzeniami ESP8266 i Raspberry Pi 4 w projekcie IoT jest uzasadniony ze względu na jego lekkość, szybkość i niskie wymagania dotyczące zasobów sprzętowych. Protokół ten oferuje również wysoki poziom bezpieczeństwa oraz wiele funkcjonalności, które są przydatne w przypadku przesyłania danych w IoT.

3.4. Przegląd dostępnych serwerów w systemach IoT

Wśród najczęściej wykorzystywanych serwerach w systemach IoT, możemy wymienić:

1. **Apache2** – to popularny serwer http, który umożliwia udostępnianie treści internetowych, takich jak strony internetowe i pliki. W projektach IoT Apache2 może być wykorzystywany jako serwer internetowy do wyświetlania interfejsów użytkownika lub do udostępniania plików konfiguracyjnych dla urządzeń [21].
2. **Mosquitto** – to broker MQTT, który umożliwia komunikację między urządzeniami IoT. MQTT jest lekkim i efektywnym protokołem komunikacyjnym, który pozwala na przesyłanie danych w czasie rzeczywistym między urządzeniami IoT. Mosquitto może

być wykorzystywany jako broker MQTT do przesyłania danych między urządzeniami [22].

3. **Node-RED** – to oprogramowanie, które umożliwia łatwe tworzenie aplikacji IoT poprzez wizualny interfejs graficzny. Node-RED można wykorzystać do tworzenia przepływów danych między urządzeniami, integracji z różnymi serwerami i bazami danych, jak również do przetwarzania i analizy danych [23].

3.5. Serwer Apache2

Wykorzystanie serwera Apache2 [21] w projekcie IoT jest jednym z najlepszych rozwiązań ze względu na jego uniwersalność i wszechstronność. Apache2 to popularny serwer HTTP, który jest darmowy i open-source. Jest on dostępny na wielu platformach, takich jak Linux, Unix, Windows i inne. Jego popularność wynika z tego, że jest łatwy w instalacji i konfiguracji, a także ze względu na jego wydajność, skalowalność i stabilność.

Rysunek 6. Główne logo serwera Apache2 [29].

Serwer Apache2 jest często stosowany w projektach IoT do zbierania danych w bazie danych MySQL oraz wyświetlania informacji na stronach internetowych. Jest to możliwe dzięki modułom, które można zainstalować na serwerze Apache2, takim jak mod_php czy mod_perl. Te moduły pozwalają na uruchomienie skryptów PHP i Perl, które są często wykorzystywane w projektach IoT.

Ponadto, serwer Apache2 oferuje wiele funkcjonalności, takich jak wirtualne hosty, uwierzytelnianie, obsługę protokołu SSL i wiele innych. Pozwala to na dostosowanie serwera do specyficznych potrzeb projektu IoT.

Ważnym atutem Apache2 jest również bogata dokumentacja, która ułatwia korzystanie z serwera dla początkujących i zaawansowanych użytkowników. Ponadto, istnieje wiele społeczności i forów, gdzie użytkownicy mogą uzyskać pomoc i porady w razie problemów z serwerem.

Wydajność serwera Apache2 jest również znana ze względu na to, że może obsługiwać wiele jednoczesnych połączeń. Dzięki temu serwer jest w stanie obsłużyć wiele urządzeń IoT, które przesyłają dane jednocześnie.

Podsumowując: wykorzystanie serwera Apache2 w projekcie IoT jest jednym z najlepszych rozwiązań ze względu na jego uniwersalność, łatwość konfiguracji, wydajność i skalowalność. Ponadto, bogata dokumentacja i społeczność użytkowników Apache2 czynią go łatwym w użyciu nawet dla początkujących użytkowników.

3.6. Serwer Mosquitto

Wykorzystanie serwera Mosquitto [22] w projekcie IoT związane z zbieraniem danych z urządzeń ESP8266 jest jednym z najlepszych rozwiązań ze względu na jego specyficzne cechy i zalety. Mosquitto jest niewielkim i wydajnym serwerem, który obsługuje protokół MQTT, co czyni go idealnym rozwiązaniem dla projektów IoT. Poniżej przedstawiam argumenty za wykorzystaniem serwera Mosquitto.

Rysunek 7. Główne logo serweru Mosquitto [30].

Po pierwsze, Mosquitto jest łatwy w instalacji i konfiguracji, dzięki czemu jego uruchomienie nie zajmuje wiele czasu. Ponadto, jest to serwer open source, co oznacza, że kod jest dostępny publicznie, a jego modyfikacja i rozwój są możliwe dla każdego.

Po drugie, Mosquitto jest niskowydajnym serwerem, co oznacza, że zużywa niewiele zasobów systemowych, co jest szczególnie ważne dla projektów IoT, gdzie urządzenia nie pobierają dużo prądu.

Po trzecie, Mosquitto obsługuje protokół MQTT, który jest uznawany za jeden z najlepszych protokołów dla IoT. MQTT jest protokołem komunikacyjnym, który zapewnia efektywną wymianę informacji między urządzeniami IoT i serwerami. Dzięki temu Mosquitto może pośredniczyć w wymianie danych między urządzeniami ESP8266 a serwerem Apache2 i MySQL.

Wreszcie, Mosquitto oferuje zaawansowane funkcje, takie jak szyfrowanie SSL / TLS, uwierzytelnienie użytkowników i kontroli dostępu, które są ważne w przypadku projektów IoT, w których bezpieczeństwo i prywatność są kluczowe.

Podsumowując: Mosquitto jest najlepszym rozwiązaniem dla projektów IoT związanych z zbieraniem danych z urządzeń ESP8266 i wymianą ich z serwerem Apache2 i MySQL. Jego zalety, takie jak łatwość instalacji i konfiguracji, niskie wymagania dotyczące zasobów, obsługa protokołu MQTT oraz zaawansowane funkcje bezpieczeństwa, czynią go idealnym wyborem dla projektów IoT.

3.7. Framework Django

Django [24] to popularny open-sourceowy framework dla języka Python, służący do tworzenia aplikacji webowych. Django pozwala na szybkie i łatwe tworzenie skalowalnych aplikacji webowych dzięki wykorzystaniu wzorca projektowego Model-View-Controller (MVC).

Django dostarcza wiele wbudowanych funkcjonalności, takich jak autoryzacja użytkowników, zarządzanie sesjami, obsługa plików statycznych i wiele innych, dzięki czemu programiści mogą skupić się na implementacji logiki biznesowej aplikacji.

Wraz z frameworkiem Django dostarczany jest wbudowany serwer WWW, który umożliwia szybkie uruchomienie aplikacji na lokalnym komputerze w czasie rozwoju. Serwer ten może być wykorzystany jako środowisko deweloperskie. Jednak do hostowania aplikacji w środowisku produkcyjnym, zalecane jest wykorzystanie dedykowanych serwerów WWW, takich jak Apache, Nginx lub Gunicorn.

Django może być również wykorzystywany w sieciach IoT, na przykład do tworzenia aplikacji monitorujących lub sterujących urządzeniami. Dzięki integracji z popularnymi bibliotekami Pythona takimi jak NumPy, Pandas czy Scikit-learn, Django umożliwia analizę danych i wizualizację wyników, co może być bardzo przydatne w przypadku aplikacji IoT.

3.8. Magistrala 1-wire

1-Wire [25] to prosty, łączący protokół komunikacyjny oraz sprzętowy interfejs do komunikacji z urządzeniami elektronicznymi. Ten system został opracowany przez firmę Dallas Semiconductor (obecnie Maxim Integrated) i jest szeroko stosowany w różnych dziedzinach, w tym w automatyce domowej, monitoringu środowiska, przemysłowych systemach sterowania i wielu innych aplikacjach.

1-Wire wykorzystuje pojedynczy przewód do przesyłania danych oraz zasilania podłączonych urządzeń. Dzięki temu prostemu interfejsowi możliwe jest komunikowanie się z wieloma urządzeniami za pomocą tylko jednego przewodu, co zapewnia wygodę i oszczędność miejsca.

Podstawowe zasady działania 1-Wire są następujące:

- **Zasilanie i identyfikacja urządzeń**: Każde urządzenie 1-Wire ma unikalny 64-bitowy identyfikator, który pozwala na jego jednoznaczną identyfikację w sieci. Urządzenia są zasilane z linii danych za pomocą tzw. Zasilania parasitarnego lub zewnętrznego źródła zasilania.

- **Komunikacja**: Komunikacja między urządzeniami a kontrolerem odbywa się za pomocą sekwencji impulsów na linii danych. Komunikacja jest oparta na zasadzie czasowej, gdzie różne wartości impulsów oznaczają różne informacje. Przykładowo, wysoki impuls może oznaczać logikę „1”, a niski impuls logikę „0”.
- **Topologia sieci**: Sieć 1-Wire może być zbudowana w różnych topologiach, takich jak linia prostokątna, pierścień lub gwiazda. Każde urządzenie ma swoje unikalne miejsce na linii danych, co pozwala na komunikację z konkretnym urządzeniem.

Zastosowania 1-Wire są szerokie i obejmują wiele dziedzin. Może być używany do pomiaru temperatury, monitorowania wilgotności, zarządzania energią, sterowania oświetleniem, monitorowania przepływu wody, identyfikacji i autentykacji urządzeń, a także w innych aplikacjach, gdzie wymagana jest prosta, niezawodna i ekonomiczna komunikacja z urządzeniami.

Dzięki swojej prostocie, niezawodności i niskim kosztom implementacji, 1-Wire znalazł szerokie zastosowanie w różnych dziedzinach. Jest to popularny protokół komunikacyjny, który umożliwia tworzenie inteligentnych systemów opartych na zbieraniu danych i sterowaniu różnymi urządzeniami w sposób wydajny i elastyczny.

3.9. Raspberry Pi 4

Raspberry Pi [26][27] to niewielki, wszechstronny komputer jednopłytkowy, który cieszy się ogromną popularnością na całym świecie. Został zaprojektowany przez Raspberry Pi Foundation w celu dostarczenia taniego, ale potężnego narzędzia do nauki programowania, eksperymentowania z elektroniką i budowania różnorodnych projektów.

Raspberry Pi składa się z jednej płytki, która zawiera wszystkie podstawowe komponenty potrzebne do działania. W zależności od modelu, może on mieć różne specyfikacje techniczne, takie jak procesor, pamięć RAM, porty USB, porty HDMI, slot kart microSD i inne interfejsy komunikacyjne. Raspberry Pi jest oparty na architekturze ARM i działa na różnych systemach operacyjnych, takich jak Raspbian (wersja systemu Linux oparta na Debianie), Ubuntu czy Windows 10 IoT Core.

Dzięki swojej wszechstronności i niskiej cenie, Raspberry Pi znalazło zastosowanie w wielu dziedzinach. Oto kilka przykładów wykorzystania Raspberry Pi:

- **Projektowanie i programowanie**: Raspberry Pi jest doskonałym narzędziem do nauki programowania. Można na nim pisać w różnych językach, takich jak Python, C/C++, Java, Scratch i wiele innych. Można tworzyć aplikacje, gry, projekty związane z sztuczną inteligencją czy Internetem Rzeczy.
- **Domowa automatyzacja**: Raspberry Pi może być używane do sterowania domowymi urządzeniami, takimi jak oświetlenie, zamki, termostaty czy systemy alarmowe. Może działać jako centrum kontroli i monitoringu, umożliwiając zdalne sterowanie i monitorowanie różnych aspektów domu.
- **Media center**: Raspberry Pi można przekształcić w centrum multimedialne, które odtwarza filmy, muzykę, serwuje treści wideo na telewizorach czy obsługuje strumieniowanie multimediów.

- **Serwer domowy**: Raspberry Pi może działać jako serwer domowy, umożliwiając przechowywanie i udostępnianie plików, hostowanie stron internetowych, serwowanie mediów strumieniowych czy udostępnianie drukarki w sieci lokalnej.
- **Projektowanie elektroniki**: Raspberry Pi może być używane jako platforma do eksperymentów z elektroniką. Dzięki dostępnym interfejsom GPIO (General Purpose Input/Output) można podłączyć czujniki, przekaźniki, silniki czy inne komponenty elektroniczne i tworzyć interaktywne projekty.
- **Internet Rzeczy (IoT):** Raspberry Pi jest często wykorzystywane w projektach związanych z Internetem Rzeczy. Może służyć jako bramka IoT, zbierać i przetwarzać dane z różnych czujników, sterować urządzeniami oraz komunikować się z chmurą.

Działanie Raspberry Pi polega na dostarczeniu komputera w formie jednej płytki, która jest wyposażona w potrzebne komponenty, aby uruchamiać system operacyjny i programy. Po podłączeniu zasilania i odpowiednich urządzeń peryferyjnych, takich jak monitor, klawiatura i mysz, Raspberry Pi jest gotowe do pracy.

Dzięki swojej otwartej architekturze i bogatej społeczności, Raspberry Pi jest nieustannie rozwijane i zyskuje coraz więcej funkcji. To niedrogi, ale wszechstronny komputer, który daje możliwość twórczego eksperymentowania, nauki i budowania własnych projektów.

3.10. ESP8266

ESP8266 [31] to mikrokontroler z wbudowanym modułem Wi-Fi, który został opracowany przez firmę Espressif Systems. Jest to popularne rozwiązanie wykorzystywane w projektach związanych z Internetem Rzeczy (IoT) i komunikacją bezprzewodową.

ESP8266 pełni wiele funkcji i może być wykorzystywany w różnych zastosowaniach. Oto kilka sposobów, w jakich ESP8266 znajduje swoje zastosowanie:

Podłączanie urządzeń do sieci: ESP8266 umożliwia podłączenie różnych urządzeń do sieci Wi-Fi. Może to obejmować czujniki, urządzenia sterujące, moduły komunikacyjne, systemy monitoringu i wiele innych. Dzięki temu mikrokontrolerowi urządzenia mogą komunikować się bezprzewodowo i wymieniać dane z innymi urządzeniami i usługami w sieci.

Internet Rzeczy (IoT): ESP8266 jest często wykorzystywany w projektach związanych z Internetem Rzeczy. Może działać jako bramka IoT, umożliwiając podłączenie wielu urządzeń i zbieranie danych z różnych czujników. Może również komunikować się z chmurą, przesyłać dane, odbierać instrukcje i sterować innymi urządzeniami.

Automatyzacja domowa: ESP8266 może być wykorzystywany w inteligentnych domach do sterowania oświetleniem, urządzeniami grzewczymi, klimatyzacją, systemami alarmowymi i innymi urządzeniami elektrycznymi. Mikrokontroler umożliwia zdalne sterowanie tymi urządzeniami poprzez połączenie z siecią Wi-Fi.

Monitorowanie i kontrola: ESP8266 może być wykorzystywany do monitorowania różnych parametrów, takich jak temperatura, wilgotność, poziom światła, ruch i wiele innych. Może również służyć do kontroli urządzeń, np. za pomocą przekaźników lub silników.

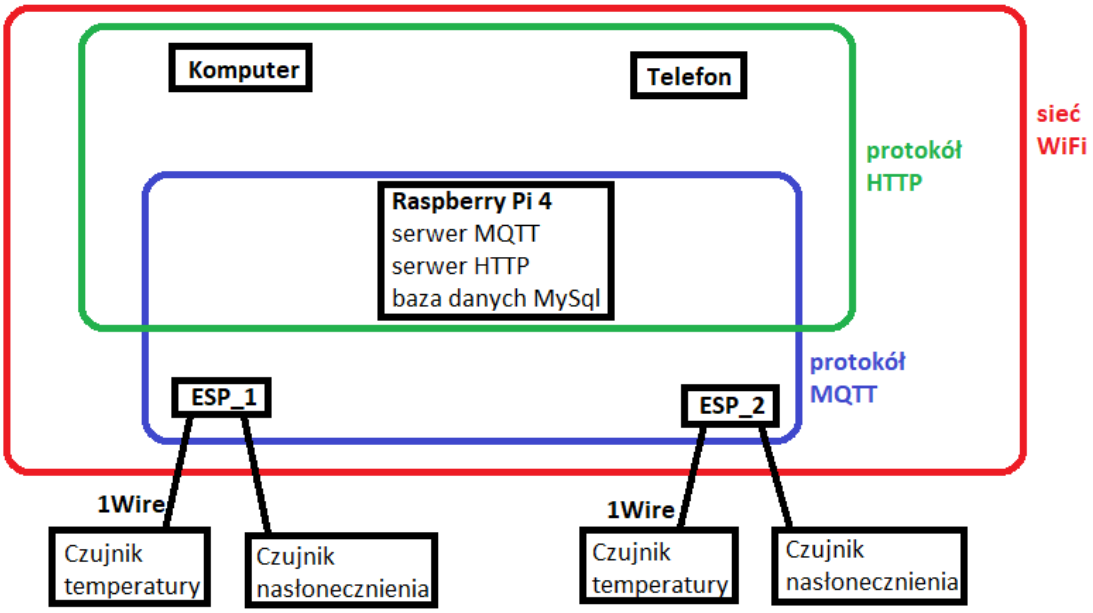
Tworzenie aplikacji mobilnych: ESP8266 może być wykorzystywany w połączeniu z aplikacjami mobilnymi, aby zapewnić interakcję i sterowanie urządzeniami za pomocą smartfonów lub tabletów. Może być wykorzystywany jako mostek między urządzeniami IoT a aplikacją mobilną.

Dzięki swojemu niewielkiemu rozmiarowi, niskiemu poborowi mocy i wbudowanemu modułowi Wi-Fi, ESP8266 jest wygodnym i niedrogim rozwiązaniem do podłączania urządzeń do sieci bezprzewodowej. Jest on często wykorzystywany przez hobbystów, twórców projektów DIY oraz profesjonalistów w branży IoT do budowania inteligentnych i zintegrowanych systemów.

4. Realizacja projektu

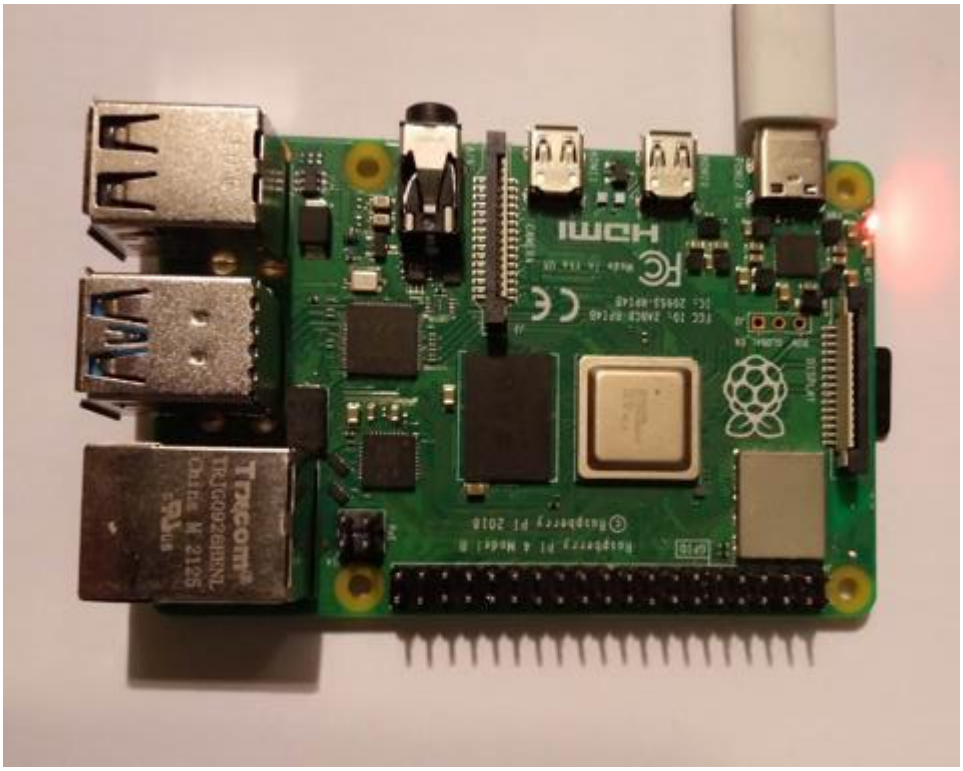
4.1. Rzeczywisty układ

Poniżej przedstawiono schemat działania utworzonej sieci i zakres działania protokołów. Jako urządzenie nadrzędne służyło Raspberry Pi. Tworzyło ono komunikację przez MQTT z urządzeniami ESP, oraz przez protokół HTTP z zewnętrznymi urządzeniami użytkownika, Typu telefon lub komputer które posiadały przeglądarkę internetową i były w tej samej sieci WiFi co Raspberry Pi.

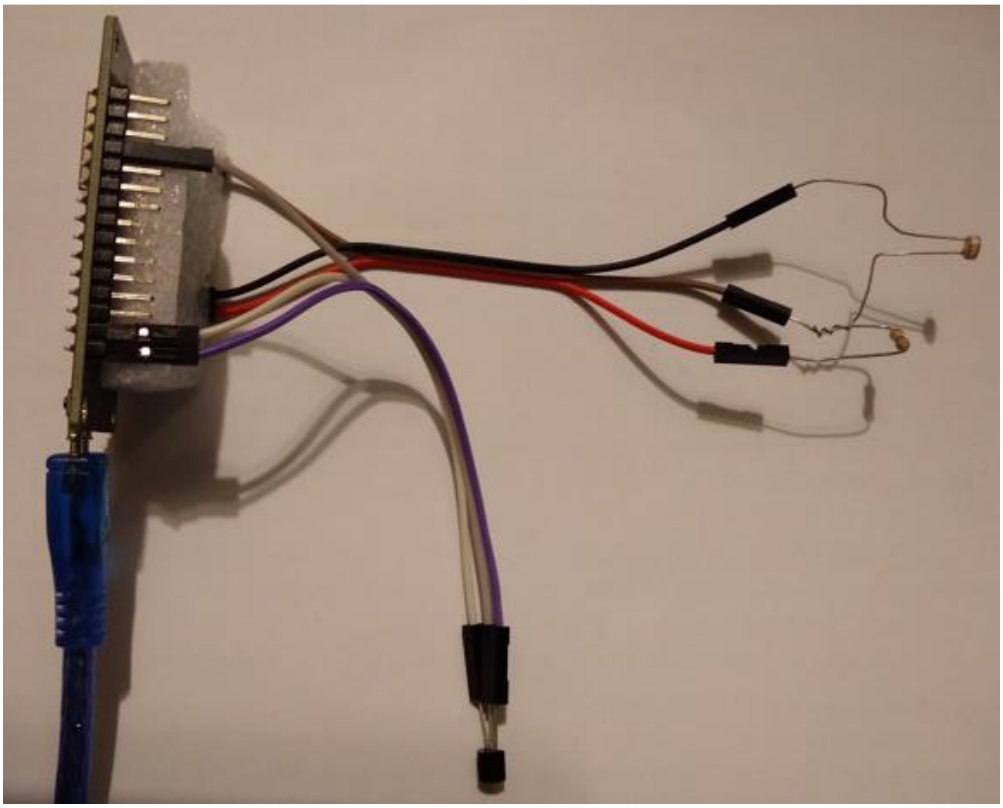


Rysunek 8. Schemat poglądowy utworzonego projektu.

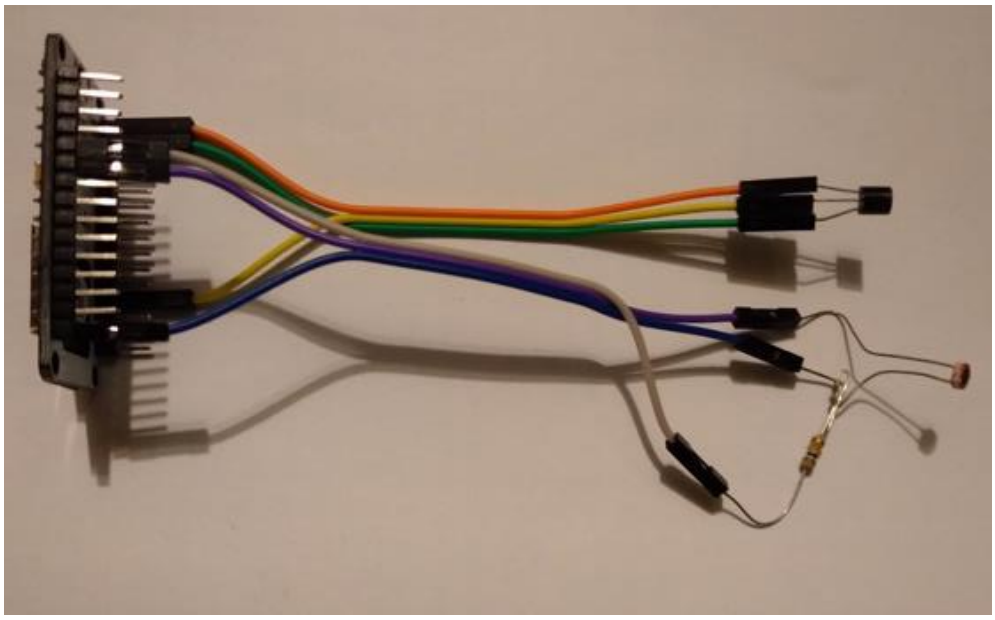
Poniżej zamieszczone zostały fotografie przedstawiające wykorzystane urządzenia w projekcie. Było to Raspberry Pi 4 znajdujące się w pokoju, dwa ESP8266 jedno w pokoju, drugie za oknem.



Rysunek 9. Raspberry Pi 4.



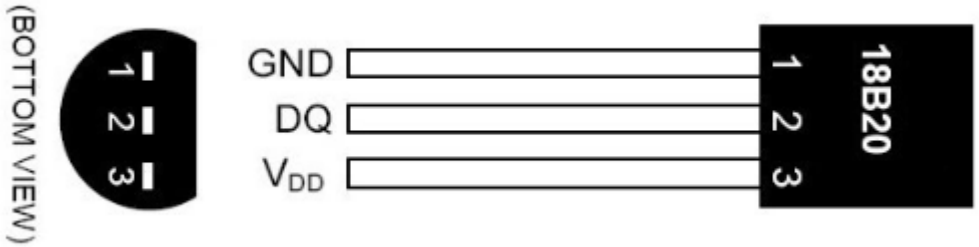
Rysunek 10. ESP8266 nr 1 oraz czujniki.



Rysunek 11. ESP8266 nr 2 oraz czujniki.

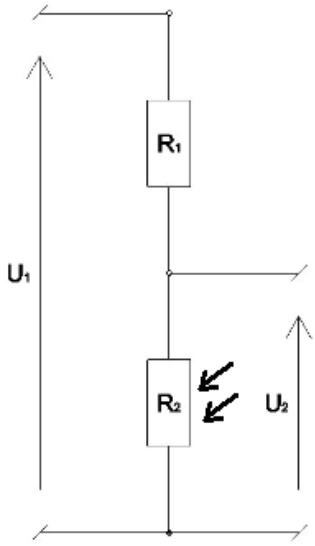
4.2. Hardware projektu

Wykorzystane czujniki to termometr cyfrowy DS18B20 oraz fotorezystor w układzie dzielnika napięcia.



Rysunek 12. Termometr cyfrowy.

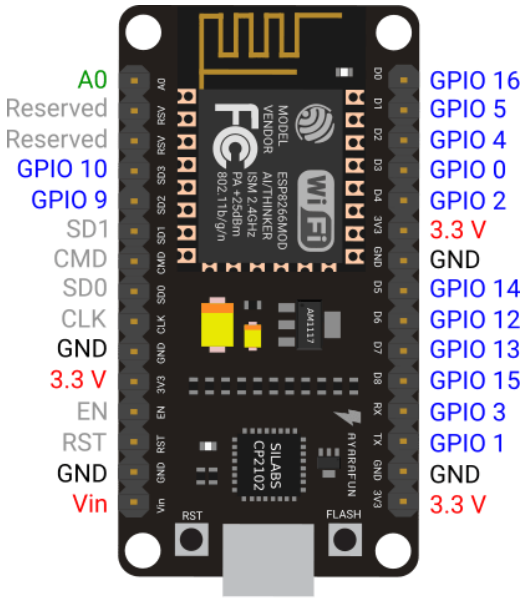
Powyższy czujnik temperatury 18B20, połączono z układem ESP8266. GND podłączono do masy, VDD podłączono do zasilania 3,3V. Wyprowadzenie DQ połączono z pinem GPIO0 układu ESP. Do komunikacji wykorzystano jeden przewód.



Rysunek 13. Fotorezystor w układzie dzielnika napięcia.

W powyższym układzie fotorezystora zastosowano rezystor R1 o wartości 560kΩ. R2 to fotorezystor o zmniejszającej się rezystancji. Z powyższego układu wynika że wraz ze wzrostem natężenia światła maleje wartość napięcia U2. Napięcie z dzielnika podawane jest na pin A0 układu ESP. Jest to przetwornik analogowo-cyfrowy.

Wykorzystane platformy sprzętowe to ESP8266 i Raspberry Pi 4.



Rysunek 14. ESP8266 pinout.

W przypadku ESP wykorzystano dwa piny do czujnika temperatury i światła. Wykorzystany został wbudowany moduł WiFi który jest wbudowany w układ SoC.

Dla Raspberry nie wykorzystano żadnych pinów. Został użyty moduł WiFi wbudowany w układ SoC znajdujący się na płytce.

4.3. Software – Raspberry Pi

Aby móc wykorzystać Raspberry Pi 4 w projekcie należało zainstalować system operacyjny na karcie microSD. Wykorzystany system operacyjny to:

Raspberry Pi OS Lite (32-bit) – port of Debian Bullseye with no desktop enviroment.

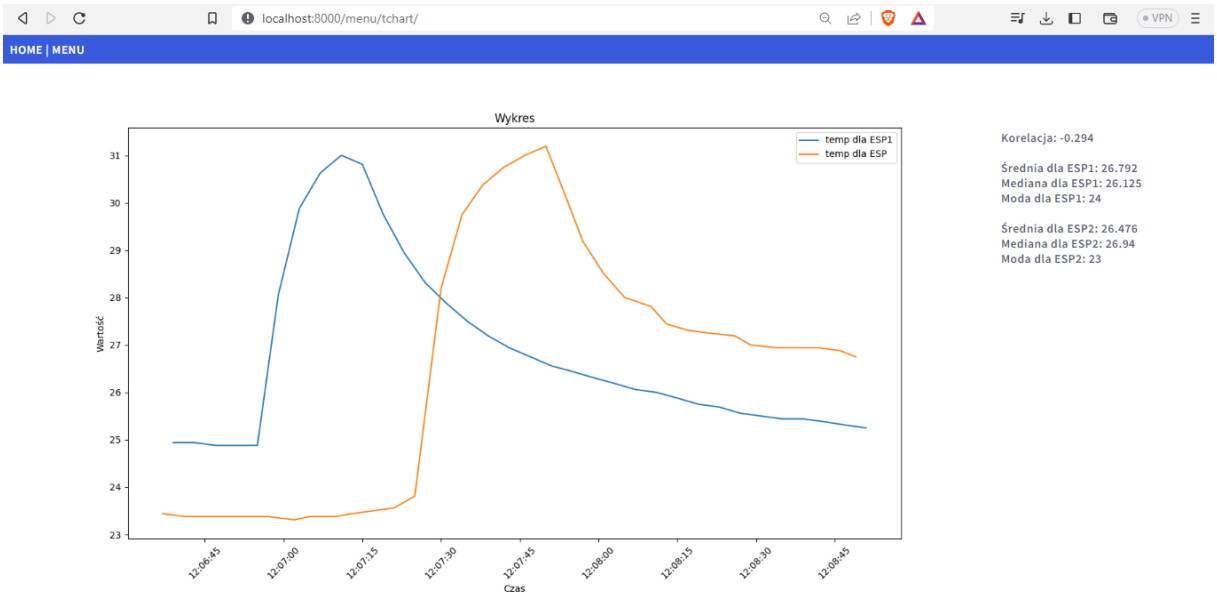
Kolejnym krokiem było ustawienie statycznego adresu IP urządzenia, tak aby można było połączyć się zdalnie przez protokół SSH. Kolejnych konfiguracji systemu dokonano z poziomu konsoli. Dostęp do konsoli był zdalny, wykorzystano program Putty. Do kopiowania plików między komputerem a Raspberry wykorzystano program WinSCP. Do pisania programów wykorzystano VisualStudio Code wraz z dodatkiem SSH-remote.

4.3.1. Serwer HTTP – Django

Dla stron internetowych skorzystano z serwera Django. Cała aplikacja została napisana w frameworku o takiej samej nazwie. Framework umożliwia pisanie aplikacji internetowych o architekturze plików MVC (Model-View-Controller). Utworzone strony internetowe:

Rysunek 15. Strona startowa aplikacji.

Rysunek 16. Menu aplikacji.



Rysunek 17. Wykresy temperatury + statystyki.

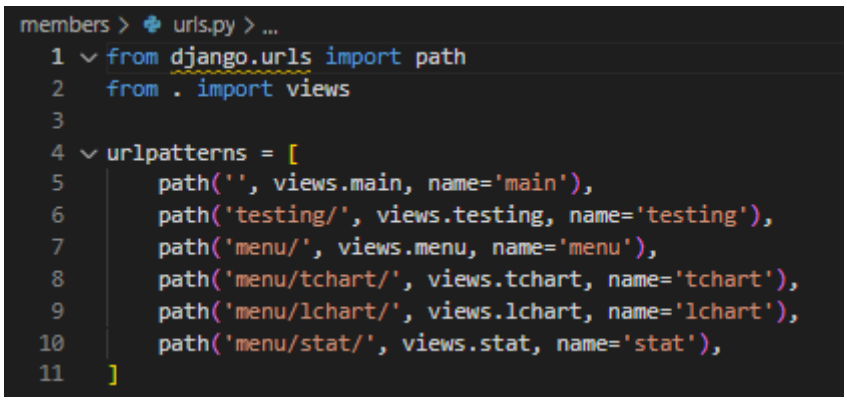
Rysunek 18. Wykresy naświetlenia + statystyki.

Zasada działania programu

Dane są pobierane z bazy danych MySQL i wyświetlane na wykresach. Do wykresów i analityki danych wykorzystano matplotlib i pandas.

Z punktu widzenia struktury MVC, najważniejsze pliki projektu to:

- urls.py - odpowiada za kontrolę adresów URL i przekierowania między widokami, których zadaniem jest generowanie odpowiedzi dla danego żądania HTTP.
- views.py - to plik, w którym znajdują się widoki aplikacji, czyli funkcje, które wykonują logikę biznesową aplikacji i generują odpowiedzi na żądania HTTP.
- Katalog templates - zawiera pliki HTML, które odpowiadają za prezentację danych generowanych przez widoki.



Rysunek 19. Kontroler aplikacji.

```
def main(request):
  template = loader.get_template('main.html')
  return HttpResponse(template.render())

def testing(request):
  myesp = esp1.objects.values()
  template = loader.get_template('template.html')
  context = {
    'myesp' : myesp,
  }
  return HttpResponse(template.render(context, request))
```

Rysunek 20. Fragment pliku odpowiedzialnego za widoki aplikacji.

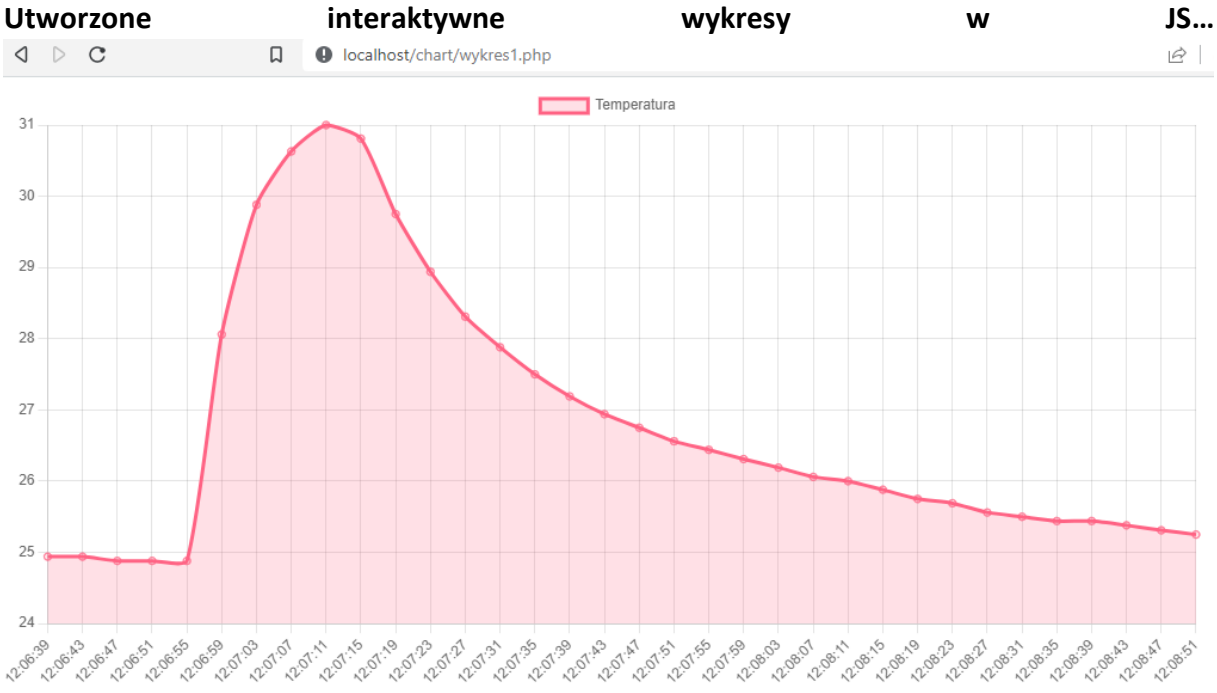
members > templates > mchart.html > ...

```
{% extends "main.html" %}

{% block content %}
    {% if chart %}
        <div style="display: flex; justify-content: left;">
            <img src=" {{chart|safe}}"/>
            <div style="white-space: nowrap; margin-right: 500px;margin-top: 100px; align-items: center;">
                Korelacja: {{ correlat }}<br><br>

                Średnia dla ESP1: {{ mn1 }}<br>
                Mediana dla ESP1: {{ med1 }}<br>
                Moda dla ESP1: {{ mod1 }}<br><br>

                Średnia dla ESP2: {{ mn2 }}<br>
                Mediana dla ESP2: {{ med2 }}<br>
                Moda dla ESP2: {{ mod2 }}<br>
            </div>

            </div>
        </div>
    {% endif %}
{% endblock content %}
```

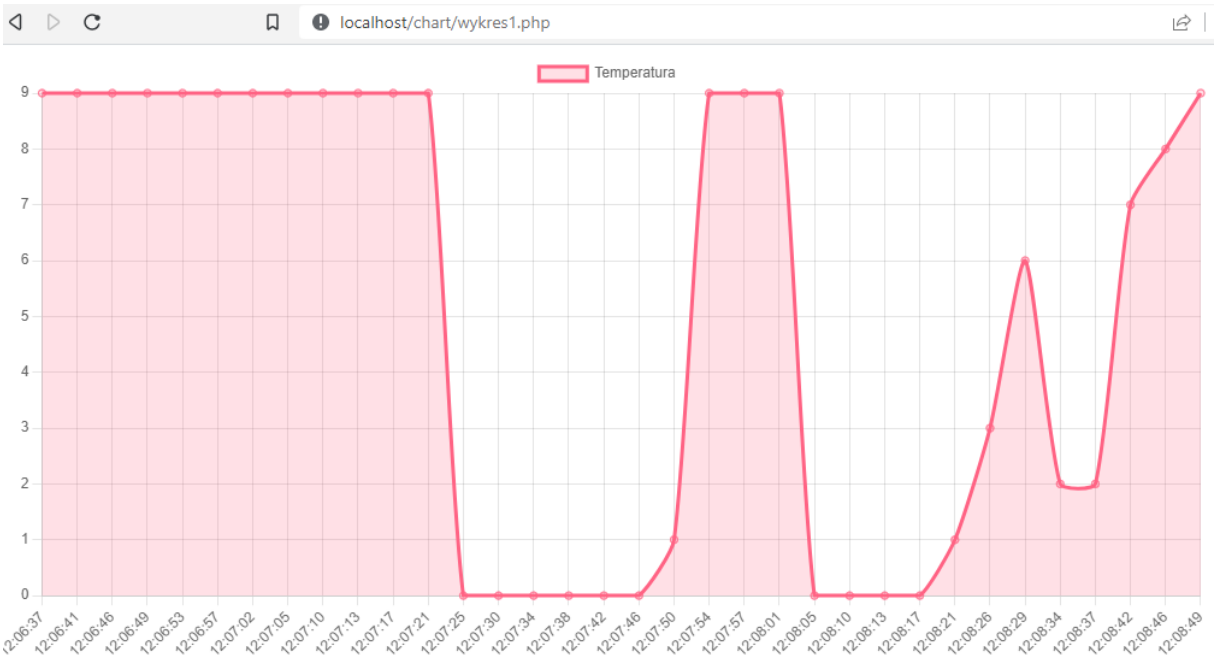
Rysunek 21. Model - przykładowa strona odpowiedzialna za wyświetlenie wykresu i statystyk.

W powyższym przypadku wykresy są generowane przez funkcje matplotlib i printowane na obraz png który jest umieszczany na stronie.

Poniżej przedstawiono koncepcje tworzenia interaktywnych wykresów w JavaScript. Ta część projektu jest jeszcze rozwijana.



Rysunek 22. Wykres temperaturowy.



Rysunek 23. Wykres naświetlenia.

4.3.2. Baza danych MySql – dostęp phpMyAdmin

Należało utworzyć serwer www na Raspberry Pi. Tworzenie i konfiguracja serwera sprowadzała się do podania następujących po sobie komend. Komendy były wprowadzane w powłoce Bash systemu Raspbian.

Instalacja LAMP Apache2:

```
sudo apt install apache2 -y
```

Instalacja interpretera PHP:

```
sudo apt install php –y
```

Instalacja bazy danych MySql:

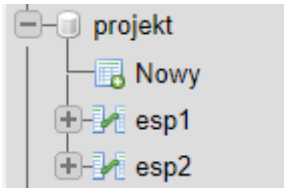
```
sudo apt install mysql-server php-mysql -y
```

Instalacja phpMyAdmin:

```
sudo apt install phpmyadmin
```

Powyższe kroki powodują instalacje podstawowych elementów związanych z serwerem Apache2 i bazą danych MySQL.

Dostęp do bazy danych odbywał się z poziomu przeglądarki internetowej. Należało przejść do strony phpMyAdmin znajdującej się na serwerze localhost Apache2 na Raspberry Pi. Dokonano utworzenie nowej bazy danych oraz tabeli.



Rysunek 24. Utworzona baza danych oraz tabele.

W tych tabelach były gromadzone dane z czujników. W każdej tabeli zostały utworzone następujące kolumny:

- date – aktualna data dokonania pomiaru
- time, - aktualna godzina, minuta i sekunda dokonania pomiaru
- temp – odczytana wartość temperatury w st. C
- light – odczytana wartość naświetlenia, wartości 0-10 (gdzie: 0 – bardzo jasno, 10 – bardzo ciemno)

Dokonano odczytu ww wartości co 4 sekundy, wynik wypełnionej tabeli esp1:

date	time	temp	light
2023-03-18	12:06:39	24.94	7
2023-03-18	12:06:43	24.94	7
2023-03-18	12:06:47	24.88	7
2023-03-18	12:06:51	24.88	7
2023-03-18	12:06:55	24.88	1
2023-03-18	12:06:59	28.06	0
2023-03-18	12:07:03	29.88	0
2023-03-18	12:07:07	30.63	0
2023-03-18	12:07:11	31.00	0
2023-03-18	12:07:15	30.81	7
2023-03-18	12:07:19	29.75	7
2023-03-18	12:07:23	28.94	7
2023-03-18	12:07:27	28.31	7
2023-03-18	12:07:31	27.88	7
2023-03-18	12:07:35	27.50	7
2023-03-18	12:07:39	27.19	7
2023-03-18	12:07:43	26.94	7
2023-03-18	12:07:47	26.75	7

Rysunek 25. Wypełniona tabela esp1.

4.3.3. Serwer MQTT – język Python

Poniżej zostaną omówione najważniejsze części programu wykonanego w języku Python. Wykonano dwa programy, odpowiednio dla subskrybcji i publikowania tematów. Poszczególne programy i ich funkcje:

- **rasp_sub.py**
  - nawiązanie połączenia z serwerem MQTT
  - ustawienie subskrybcji tematów od ESP8266
  - wysyłanie odepranych danych do bazy danych
- **rasp_pub.py**
  - nawiązanie połączenia z serwerem MQTT
  - ustawienie publikowanie tematów od Raspberry Pi
  - przykład użycia programu – wysyłanie inkrementowanej zmiennej

Najważniejsze fragmenty programu **rasp_sub.py**:

```
# funkcja - nawiązanie połączenia z serwerem MQTT
def on_connect(client, userdata, flags, rc):
    global flag_connected
    flag_connected = 1                  # flaga połączenia = 1
    client_subscriptions(client)        # ustawienie subskrybcji
    print("Connected to MQTT server")

# funkcja - przerwanie połączenia z serwerem MQTT
def on_disconnect(client, userdata, rc):
    global flag_connected
    flag_connected = 0                  # flaga połączenia = 0
    print("Disconnected from MQTT server")
```

Fragment kodu 1. Funkcje wykonane przy nawiązaniu i zerwaniu połączenia z serwerem.

```
# funkcja skalująca odczytaną wartość z przetwornika ADC 10-bitowego
# na wartości 0-10 bardziej intuicyjne
def light_scale(light):      # określa natężenie światła
    scale = int((light-100)/30)
    if(scale < 0): scale = 0
    return scale
```

Fragment kodu 2. Funkcja skalująca wartość natężenia światła.

```
# callback - wykonany gry wystąpi odbiór danych z ESP nr 1
def callback_esp32_sensor1(client, userdata, msg):
    print('ESP sensor1 data: ', msg.payload.decode('utf-8'))    # informacja wyświetlona w konsoli

    # utworzenie zmiennych daty, czasu oraz temperatury, czasu odebranego z ESP
    sql = "INSERT INTO esp1 VALUES (%s, %s, %s, %s)"
    now = datetime.now()
    actual_time = now.strftime('%H:%M:%S')
    actual_date = now.strftime('%Y-%m-%d')
    split_words = msg.payload.decode('utf-8').split("#")
    val = (actual_date, actual_time, split_words[0], light_scale(int(split_words[1])))

    mycursor.execute(sql, val)  # wykonanie zapytania do bazy danych MySql
    mydb.commit()
```

Fragment kodu 3. Callback ESP nr 1 - odbiór danych, wysłanie do bazy danych.

```python
# callback - wykonany gry wystąpi odbiór danych z ESP nr 2
def callback_esp32_sensor2(client, userdata, msg):
    print('ESP sensor2 data: ', msg.payload.decode('utf-8'))    # informacja wyświetlona w konsoli

    # utworzenie zmiennych daty, czasu oraz temperatury, czasu odebranego z ESP
    sql = "INSERT INTO esp2 VALUES (%s, %s, %s, %s)"
    now = datetime.now()
    actual_time = now.strftime('%H:%M:%S')
    actual_date = now.strftime('%Y-%m-%d')
    split_words = msg.payload.decode('utf-8').split("#")
    val = (actual_date, actual_time, split_words[0], light_scale(int(split_words[1])))

    mycursor.execute(sql, val)   # wykonanie zapytania do bazy danych MySql
    mydb.commit()
```

Fragment kodu 4. Callback ESP nr 2 - odbiór danych, wysłanie do bazy danych.

```python
# callback - wykonany gry wystąpi odbiór danych z Raspberry Pi
def callback_rpi_broadcast(client, userdata, msg):
    print('RPi Broadcast message:  ', str(msg.payload.decode('utf-8')))
```

Fragment kodu 5. Callback Raspberry Pi - odbiór danych, wyświetlenie w konsoli

```python
# definiowanie subskrypcji klientów MQTT do określonych tematów
def client_subscriptions(client):
    client.subscribe("esp32/#")         # odbiór wiadomości z każdego tematu "esp32/..."
    client.subscribe("rpi/broadcast")   # odbiór wiadomości z każdego tematu "rpi/broadcast"

client = mqtt.Client("rpi_client1")     # utwórz nowego klienta MQTT o nazwie "rpi_client1"
flag_connected = 0

# przypisanie funkcji do zmiennych w bibliotece MQTT
client.on_connect = on_connect
client.on_disconnect = on_disconnect
```

Fragment kodu 6. Zdefiniowanie subskrybcji tematów, tworzenie klienta sieci.

```python
# przypisanie tematów do callbacków
client.message_callback_add('esp32/sensor1', callback_esp32_sensor1)
client.message_callback_add('esp32/sensor2', callback_esp32_sensor2)
client.message_callback_add('rpi/broadcast', callback_rpi_broadcast)
client.connect('192.168.160.125',1883)  # adres IP serwera / localhost
```

Fragment kodu 7. Przypisanie callbacków.

```python
# utwórz nowy wątek
client.loop_start()             # uruchomienie w tle pętli sieciowej klienta MQTT
client_subscriptions(client)    # start subskrybcji
print("......client setup complete.............")
```

Fragment kodu 8. Subskrybcja odbywa się w nowym wątku.

```
while True:
    time.sleep(4)       # informowanie gdy uracono połączenie z serwerem
    if (flag_connected != 1):
        print("trying to connect MQTT server..")
```

Fragment kodu 9. Pętla główna programu.

Najważniejsze fragmenty programu **rasp_pub.py**:

```
client = mqtt.Client("rpi_client2")     # utwórz nowego klienta MQTT o nazwie "rpi_client2"
client.on_publish = on_publish          # przypisanie funkcji do zmiennych w bibliotece MQTT
client.connect('192.168.160.125',1883)  # adres IP serwera / localhost

client.loop_start()                     # uruchomienie w tle pętli sieciowej klienta MQTT
```

Fragment kodu 10. Utworzenie nowego klienta sieci, nawiązanie połączenia z serwerem, przeniesienie procesów do nowego wątku.

```
k=0
# publikowanie wiadomości na temat "rpi/broadcast"
# w wiadomości wysyłana wartość zmiennej k inkrementowana od 1 do 20 co 2 sekundy
while True:
    k=k+1
    if(k>20):
        k=1

    try:
        msg =str(k)
        pubMsg = client.publish(
            topic='rpi/broadcast',
            payload=msg.encode('utf-8'),
            qos=0,
        )
        pubMsg.wait_for_publish()
        print(pubMsg.is_published())

    except Exception as e:
        print(e)

    time.sleep(2)
```

Fragment kodu 11. Pętla główna programu.

4.4. Software ESP8266 – język Arduino

Poniżej zostaną omówione najważniejsze części programu wykonanego w języku Arduino. Wykonany program odpowiedzialny jest m.in:

- Nawiązanie połączenie się z siecią WiFi,
- Nawiązanie połączenie z serwerem MQTT,
- odbiór danych z czujnika temperatury i światła,
- subskrybowanie i publikowanie tematów przez protokół MQTT.

```
// funkcja służąca do łączenia z siecią WiFi
void setup_wifi() {
  delay(50);      // stabilizacja układu przed rozpoczęciem połączenia z siecią
  Serial.println();     // pusta linia w monitorze szeregowym
  Serial.print("Connecting to ");     // napis "Connecting to " w monitorze szeregowym
  Serial.println(ssid);     // wypisanie nazwy sieci WiFi w monitorze szeregowym

  WiFi.begin(ssid, password);   // rozpoczęcie połączenia z siecią WiFi

  int c=0;
  // oczekiwanie na połączenie z siecią WiFi
  while (WiFi.status() != WL_CONNECTED) {
    blink_led(2,200); // mruganie diodą kiedy nawiązywane jest połączenie z siecią
    delay(1000); // oczekiwaniy czas na kolejną próbę nawiązania połączenia
    Serial.print(".");    // wizualne śledzenie procesu łączenia z siecią
    c=c+1;      // zliczanie liczby prób połączenia
    if(c>10){   // po 10 sekundach nie nawiązania połączenia, restart ESP
      ESP.restart(); //restart ESP after 10 seconds
    }
  }
```

Fragment kodu 12. Nawiązanie połączenie się z siecią WiFi.

```
// funkcja służąca do łączenia z serwerem/brokerem MQTT i subskrybcję tematów
void connect_mqttServer() {
  // sprawdzenie czy połączenie z serwerem MQTT zostało nawiązane
  while (!client.connected()) {

      // sprawdzenie połączenie z siecią WiFi zostało nawiązane
      if(WiFi.status() != WL_CONNECTED){
        setup_wifi();   //nawiąż połączenie z sieczią WiFi
      }

      // wypisanie informacji w monitorze szeregowym
      Serial.print("Attempting MQTT connection...");
      // dokonanie próby połączenia z serwerem MQTT, nadanie nazwy klientowi ESP
      if (client.connect("ESP32_client1")) {
        // połączenie nawiązane, monitor szeregowy
        Serial.println("connected");
        // subskrybowany temat - broadcast
        client.subscribe("rpi/broadcast");
        // ... możliwość subskrybowania większej ilości tematów

      }
      else {
        // nie udało się połączyć z serwerem MQTT
        Serial.print("failed, rc=");
        Serial.print(client.state());
        Serial.println(" trying again in 2 seconds");

        blink_led(3,200); // mruganie diodą
        // czekanie 2 sekundy na ponowną próbę połączenia z serwerem
        delay(2000);
      }
  }

}
```

Fragment kodu 13. Nawiązanie połączenie z serwerem MQTT.

```
void setup() {
  pinMode(ledPin, OUTPUT);    // dioda LED
  Serial.begin(115200);     // port szeregowy

  setup_wifi();     // nawiązanie połączenia z siecią WiFi
  client.setServer(mqtt_server,1883); // nawiązanie połączenia z serwerem MQTT
  client.setCallback(callback); // subskrybcja tematów
}
```

Fragment kodu 14. Konfiguracja portu szeregowego, łączenie z siecią WiFi - serwerem MQTT, subskrybcja tematu.

```
void loop() {

  if (!client.connected()) {
    connect_mqttServer();   // nawiązanie połączenia z serwerem MQTT
  }

  client.loop();    // obsługi komunikacji z serwerem MQTT

  // co 4 sekundy dokonywany jest pomiar temperatury i wartości z przetwornika ADC
  long now = millis();
  if (now - lastMsg > 4000) {
    lastMsg = now;

    sensors.requestTemperatures();  // Wysłanie komendy do czujnika DS18B20 w celu odczytu temperatury
    float temperature = sensors.getTempCByIndex(0);  // Odczytanie wartości temperatury w stopniach Celsiusza
    String strTemp = String(temperature);

    int value = analogRead(A0); // odczyt wartości z przetwornika ADC
    String str = String(value);

    String megaStr = strTemp + "#" + str;   // tworzenie wiadomości w formacie temperatura#światło

    client.publish("esp32/sensor1", megaStr.c_str());   // publikowanie wiadomości w temacie sensor1
  }
}
```

Fragment kodu 15. Odbiór danych z czujników, publikowanie tematu.

Program dla drugiego ESP3266 został napisany w analogiczny sposób.

5. Podsumowanie

W utworzonym projekcie z powodzeniem udało się zrealizować elektroniczną i informatyczną część projektu.

Elementy projektu które udało się zrealizować:

- utworzono serwer http i mqtt na Raspberry Pi.
- utworzono bazę danych na serwerze http.
- utworzono stronę internetową w Django, na której zamieszczono wykresy i dane statystyczne dotyczące pomiarów.
- dołączono czujniki temperatury i naświetlenia do ESP, oraz utworzono komunikację z Raspberry Pi przez sieć WiFi.

Elementy których nie udało się wykonać:

- mechaniczna część projektu, czyli montaż osprzętu m.in. silnik, przekładnia, roleta okienna.

6. Wnioski

Do realizacji projektu inżynierskiego wykorzystano wiedzę z następujących dziedzin:

- **Elektronika** – należało zapoznać się z dokumentacją układów Raspberry Pi oraz ESP. Do układu ESP należało dołączyć czujnik temperatury 18B20, gdzie wymiana informacji odbywa się magistralą 1-wire. Kolejnym czujnikiem był czujnik naświetlenia oparty na dzielniku napięcia z fotorezystorem. Należało znać zasadę działania dzielnika napięcia, oraz odpowiednio obsłużyć sygnał analogowy.
- **Informatyka** – dokonano instalacji systemu operacyjnego „No desktop enviroment Raspberry OS", należało znać podstawowe komendy Linuxa oraz pracę w terminalu. Na Raspberry Pi, utworzono serwer mqtt Apache2 i http Mosquitto. Utworzono bazę danych MySql na serwerze http. Wykonano strony internetowe na serwerze we frameworku Django, wykorzystano język Python i biblioteki m.in. pandas, paho-mqtt, django. Wykorzystano języki takie jak HTML, CSS do tworzenia stron internetowych. Wykorzystano język SQL do komunikacji z bazą danych. Skonfigurowano komunikację z pozostałymi układami w sieci WiFi. Utworzono programy na dwa urządzenia ESP, które miały odbierać dane z czujników i wysyłać przez sieć WiFi, protokół mqtt na Raspberry Pi. Programy napisano w języku Arduino.

Na podstawie ww działań można śmiało powiedzieć że projekt spełnia zagadnienia mechatroniki.

7. Bibliografia

[1] - Informacje ogólne o IoT

https://mikrokontroler.pl/2021/06/07/inteligentne-systemy-parkingowe-strumien-przestrzenny-czujnikow-iot/

[2] – Informacje ogólne o IoT

https://www.toptal.com/designers/interactive/smart-home-domestic-internet-of-things

[3] – Informacje ogólne o IoT

https://www.investopedia.com/terms/s/smart-home.asp

[4] - Porównanie sieci i protokołów bezprzewodowych

https://typeofweb.com/iot-smart-home-zigbee-z-wave-wifi

[5] - sieci i protokoły bezprzewodowe

https://en.wikipedia.org/wiki/Wireless_network

[6] - sieci i protokoły bezprzewodowe

https://ctrfantennasinc.com/characteristics-of-the-6-wireless-protocols/

[7] - parametry sieci WiFi

https://en.wikipedia.org/wiki/Wi-Fi

[8] - przegląd protokołu WebSockets

https://pl.wikipedia.org/wiki/WebSocket

[9] - przegląd protokołu MQTT

https://en.wikipedia.org/wiki/MQTT

[10] - przegląd protokołu HTTP

https://en.wikipedia.org/wiki/HTTP

[11] - przegląd protokołu HTTP

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

[12] - przegląd protokołu CoAP

https://en.wikipedia.org/wiki/Constrained_Application_Protocol

[13] - przegląd protokołu Wi-Fi Direct

https://en.wikipedia.org/wiki/Wi-Fi_Direct

[14] - przegląd sieci LoRa

https://pl.wikipedia.org/wiki/LoRaWAN

[15] - przegląd sieci WiMAX

https://pl.wikipedia.org/wiki/WiMAX

[16] - przegląd sieci LTE

https://pl.wikipedia.org/wiki/Long_Term_Evolution

[17] - przegląd sieci NFC

https://pl.wikipedia.org/wiki/Komunikacja_bliskiego_zasi%C4%99gu

[18] - obraz topologii sieci WiFi

https://sieci.infopl.info/index.php/rodzaje/lan/wlan

[19] - model protokołu HTTP

https://strefainzyniera.pl/artykul/1121/technologia-www-i-protokol-http

[20] - opis protokołu MQTT

https://iautomatyka.pl/czym-jest-protokol-mqtt/

[21] - opis Apache2

https://httpd.apache.org/ABOUT_APACHE.html

[22] opis Mosquitto

https://mosquitto.org/

[23] - opis Node-Red

https://en.wikipedia.org/wiki/Node-RED

[24] - opis Django

https://en.wikipedia.org/wiki/Django_(web_framework)

[25] - opis 1-wire

https://en.wikipedia.org/wiki/1-Wire

[26] - opis Raspberry Pi

https://pl.wikipedia.org/wiki/Raspberry_Pi

[27] - opis Raspberry Pi

https://www.spiceworks.com/tech/networking/articles/what-is-raspberry-pi/

[28] - obraz Mosquitto

https://medium.com/@saul.a.lopez.h/mqtt-vs-http-c5d97b3a8ed0

[29] - obraz Apache2

https://www.linuxadictos.com/en/install-apache-server-fedora.html

[30] - obraz Mosquitto

https://pomoc.unicloud.pl/unicloud/srodowiska-serwery/eclipse-mosquitto-serwer-osobistej-sieci-iot/

[31] - opis ESP

https://forbot.pl/blog/leksykon/esp8266