

	Instytut Informatyki Politechniki Śląskiej Zespół Mikroinformatyki i Teorii Automatów Cyfrowych			
Rok akademicki:	Rodzaj studiów*: SSI/NSI/NSM	Przedmiot (Języki Asemblerowe/ SMiW):	Grupa	Sekcja
2018/2019	SSI	SMiW	5	10
Imię:	Bartłomiej	Prowadzący:	GB	
Nazwisko:	Gładys	OA/JP/KT/GD/BSz/GB		

Raport końcowy

Tytuł: Alarm wilgotności i temperatury

Urządzenie sprawdza wilgotność i temperaturę w pomieszczeniu. Informuje użytkownika gdy jest sucho poprzez wysłanie powiadomienia na aplikacje mobilną. Użytkownik sam wcześniej deklaruje zakresy temperatur i wilgotności, w których zostanie powiadomiony. Dodatkowo urządzenie odczytuje poprzez zewnętrzny serwis informację pogodową na zewnątrz pomieszczenia.

Data oddania: dd/mm/rrrr	04.12.2018
-----------------------------	------------

1. Analiza zadania

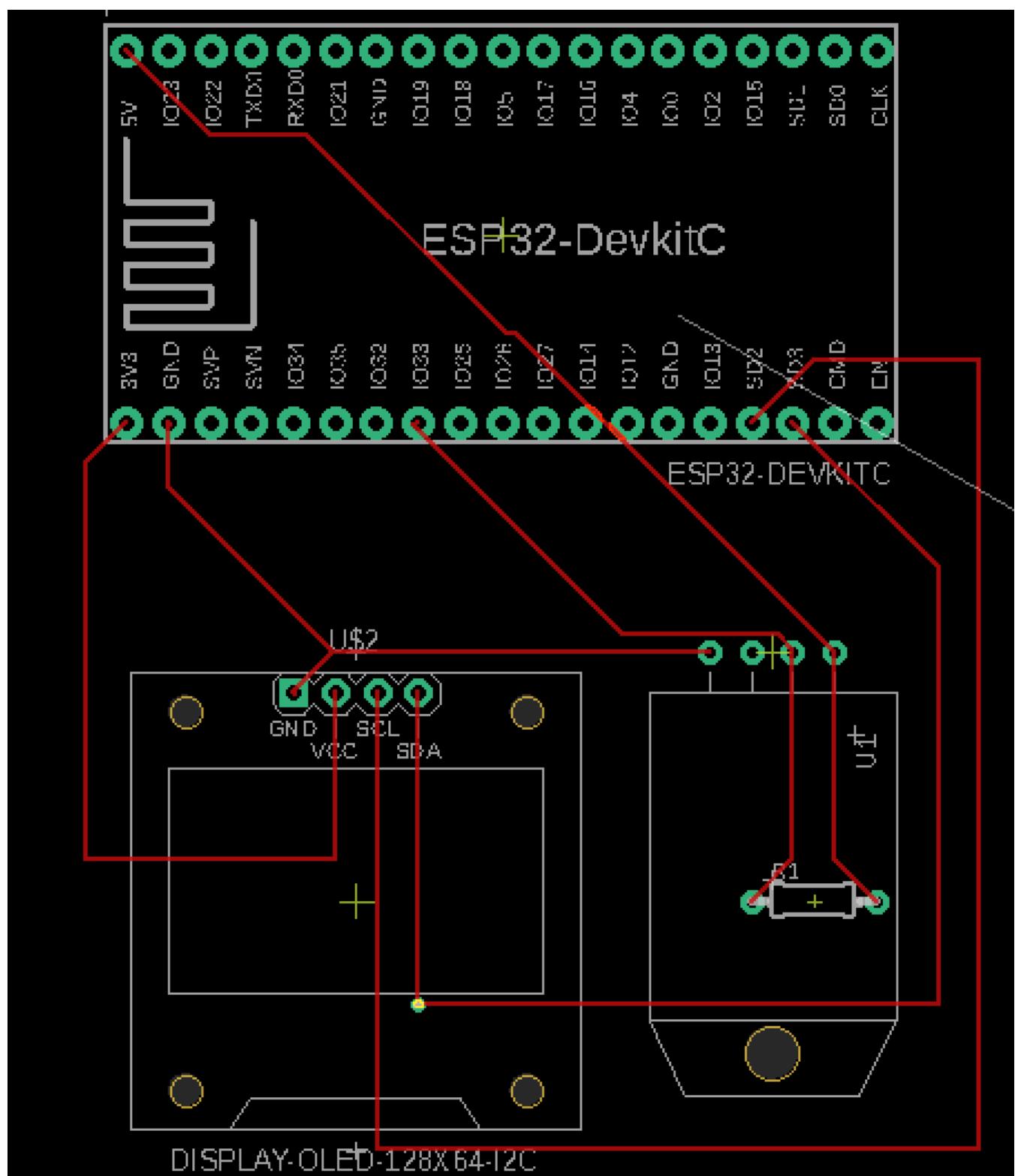
Esp32 zostało wybrane ze względu na wbudowany moduł wifi. Mikrokontroler ten był bardzo dobrym wyborem, nie miałem z nim większych problemów. Sytuacja wyglądała podobnie z czujnikiem wilgotności. DHT22 podaje prawidłowe informacje. Wybrałem go zamiast DHT11 ponieważ gwarantuje lepszą jakość, a jest w podobnej cenie. Wyświetlacz OLED SSD 1306 został wybrany ze względu na rozmiar(w sam raz na pokazanie temperatury i wilgotności w pomieszczeniu)

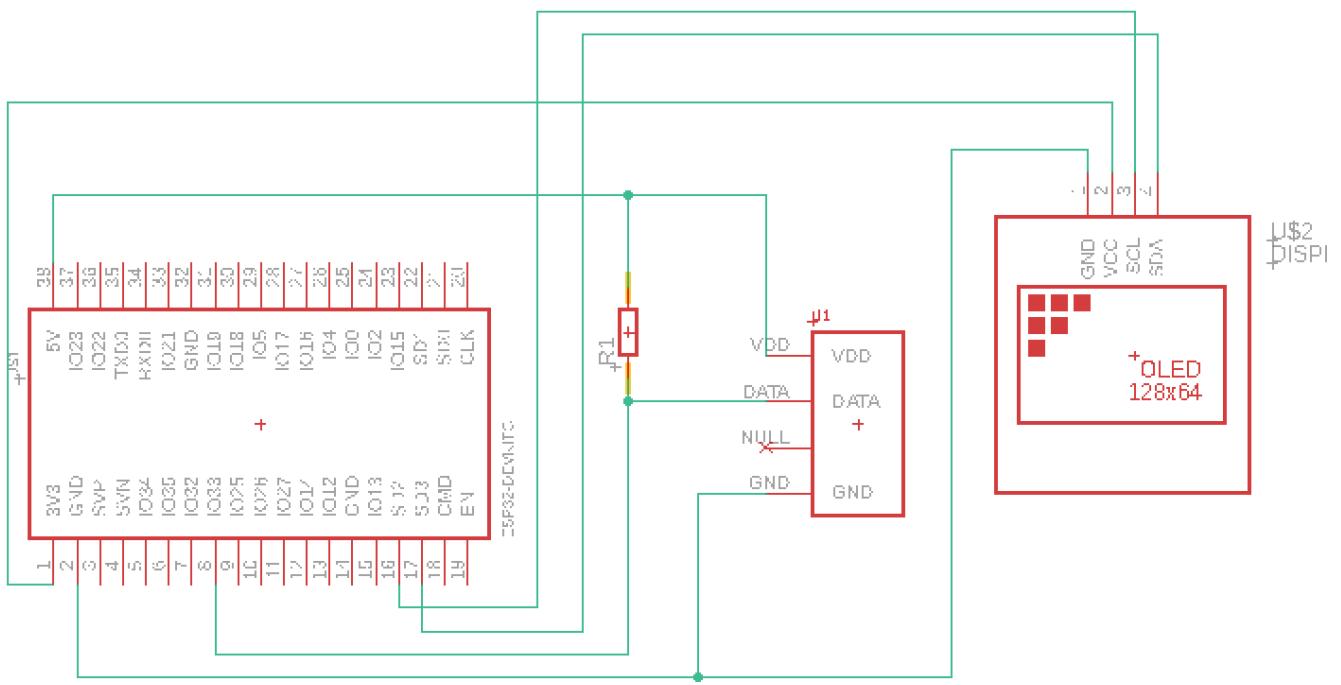
2. Specyfikacja wewnętrzna urządzenia

ESP 32 - mikrokontroler zarządzający całym układem. Po uruchomieniu, próbuje się połączyć z wifi, aby mieć kontakt z głównym serwerem MQTT, następnie próbuje pobrać informacje z DHT22. Jeżeli temperatura i wilgotność zostaną prawidłowo pobrane, mikrokontroler próbuje wysłać dane do głównego serwera oraz wyświetlić je na wyświetlaczu. Dodatkowo mikrokontroler ma za zadanie wysyłanie do serwera publicznego IP, dzięki temu możemy poznać temperaturę i wilgotność która jest w najbliższym otoczeniu układu, przez co możemy porównać informacje pogodowe na zewnątrz i wewnątrz naszego domu.

DHT22 - Pobiera wilgotność i temperaturę z najbliższego otoczenia, oraz wysyła te dane po wcześniejszym odpytaniu.

OLED SSD1306 - wyświetlanie aktualnej temperatury i wilgotności





Opis programu dla esp32:

Biblioteki z których korzystałem:

WiFi - połączenie z hot spotem oraz informacje o tym połączeniu

dht - pobieranie informacji o temperaturze i wilgotności

HTTPClient - pobieranie informacji o publicznym IP

PubSubClient - połączenie z serwerem MQTT

SSD1306 - biblioteka ułatwiająca pracę z wyświetlaczem

W bloku “setup” na początku znajduje się inicjacja wyświetlacza. Następnie czekamy na połączenie z magistrali i wywoływana jest funkcja pobierająca dane z czujnika wilgotności “loadSensorData”. Potem wywoływana jest funkcja “showSensorData”, która wyświetla dane na SSD1306. Następnie funkcja “connectWiFi” która łączy się z

wcześniej zapisanym hot spotem. Jeżeli uda się połączyć to zostają wywołane funkcje “setServer” - ustawienie danych konfiguracyjnych głównego serwera, oraz “setCallback” - ustawienie akcji która powinna się wywołać po odebraniu informacji z serwera.

W bloku “loop” sprawdzamy czy dalej posiadamy połączenie z MQTT - jeżeli nie to próbujemy je zrealizować ponownie. Następnie sprawdzamy, czy serwer posiada nasze publiczne IP - jeżeli nie to wysyłamy je ponownie. Potem w pewnym odstępie czasu cyklicznie wywołujemy funkcje “loadSensorData” oraz “showSensorData”, tak jak to miało miejsce przy inicjalizacji. Dodatkowo cyklicznie wysyłamy dane do głównego serwera poprzez funkcje “publish” ustawiając temat wiadomości na “esp32/humidity”

3. Specyfikacja zewnętrzna

Co prawda - urządzenie jest niezależne od głównego serwera - ale bez tego nie może się komunikować z urządzeniami mobilnymi. Główny serwer(Broker) jest wystawiony na google cloud platform, oraz jest napisany głównie w node.js - czyli środowisku uruchomieniowym dla Java Script.

Dodatkowo, korzysta z biblioteki mosquito, która implementuje protokół MQTT. W node.js napisana jest obsługa zdarzeń które są kompatybilne ze zdarzeniami dla MQTT(client connected, disconnected, published). Na serwerze przechowywane są również informacje o aplikacjach mobilnych połączonych z systemem. Informacje przechowywane są w taki sposób:

```

Tmax : 100, minTempMin : 0, minTempMax : 55, updatedAt : 2018-12-22T10:00:00Z
esp32 { "temperature": 24.00, "humidity": 64.80}
PUBLISH USER mobileGlaadiss { temperature: 24, humidity: 64.8 }

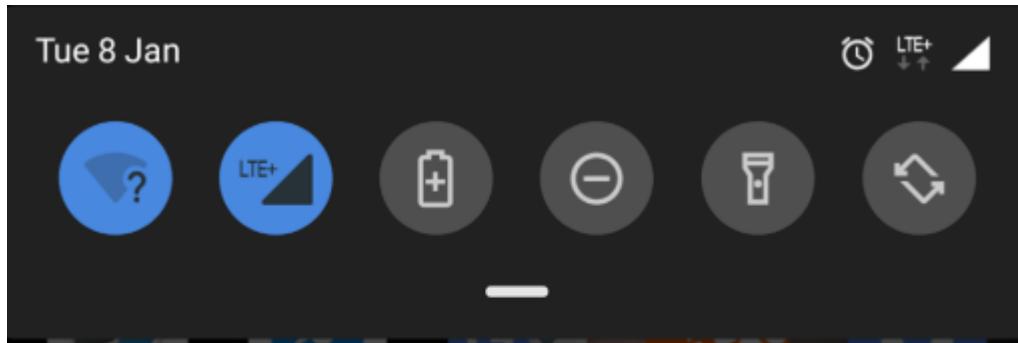
____STATE_____
temperature 24
humidity 64.8
userHumidity { mobileGlaadiss:
  { temperatureMin: 0,
    temperatureMax: 100,
    humidityMin: 0,
    humidityMax: 55,
    updatedAt: 2018-12-22T18:44:08.864Z } }
IP 176.221.122.98
-----
```

Na screenie widać, że do każdego telefonu przypisane są parametry (temperatura minimalna i maksymalna, wilgotność minimalna i maksymalna oraz data ostatniej zmiany). Dzięki tym informacjom serwer wysyła na telefon specjalne notyfikacje w momencie kiedy aktualne informacje pogodowe wyjdą poza granicę parametrów podanych wyżej. Dodatkowo, aby nie wysyłać użytkownikowi notyfikacji co 5 sekund, został zaprojektowany mechanizm, który blokuje możliwość wysłania więcej niż jednej wiadomości w ciągu dziesięciu minut.

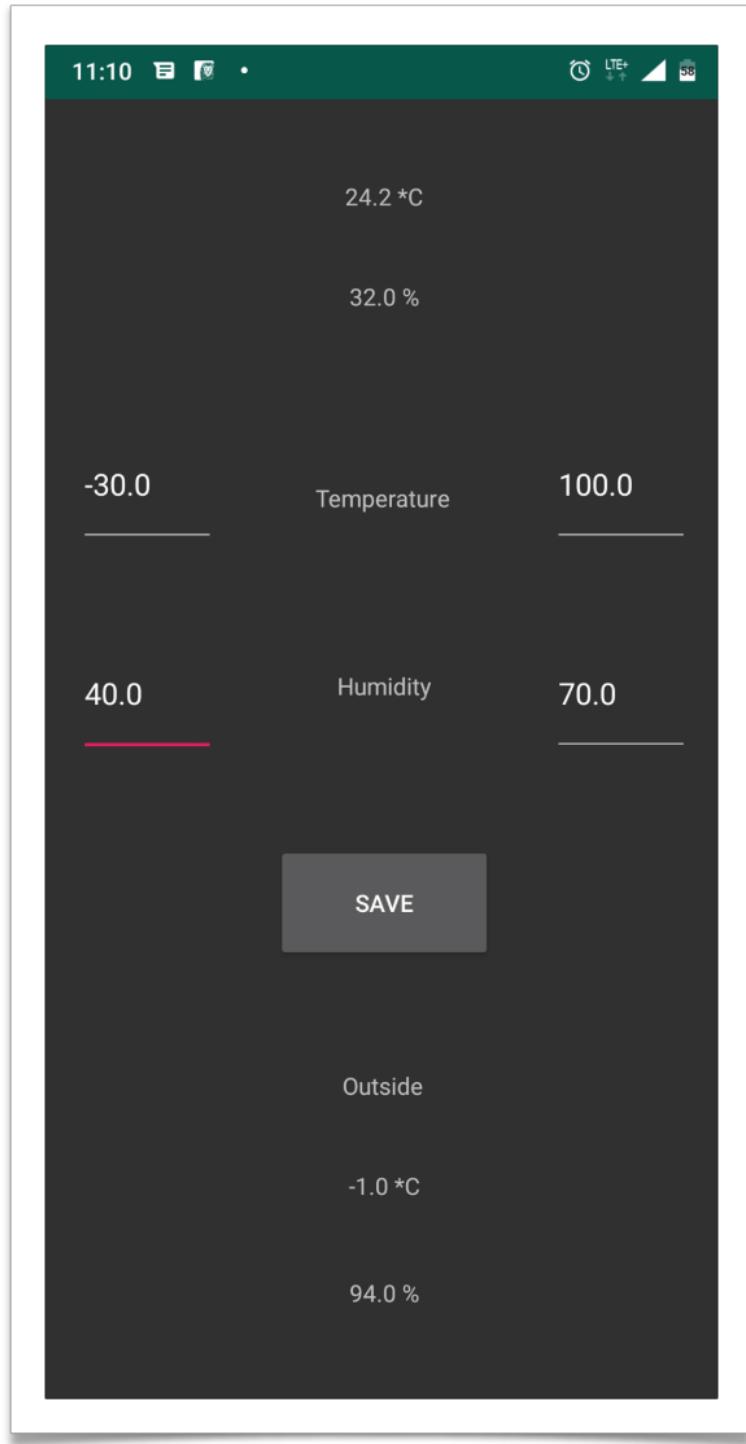


Omówiona została aplikacja dla esp32 oraz aplikacja po stronie serwera. Pozostała aplikacja mobilna, która została napisana na platformę Android.

Do kompatybilności z protokołem został użyty klient wspierający MQTT. Dodatkowo, w środku aplikacji użyty został FireBase - narzędzie od google które umożliwia push notyfikacje(po wyjściu parametrów z zakresu)



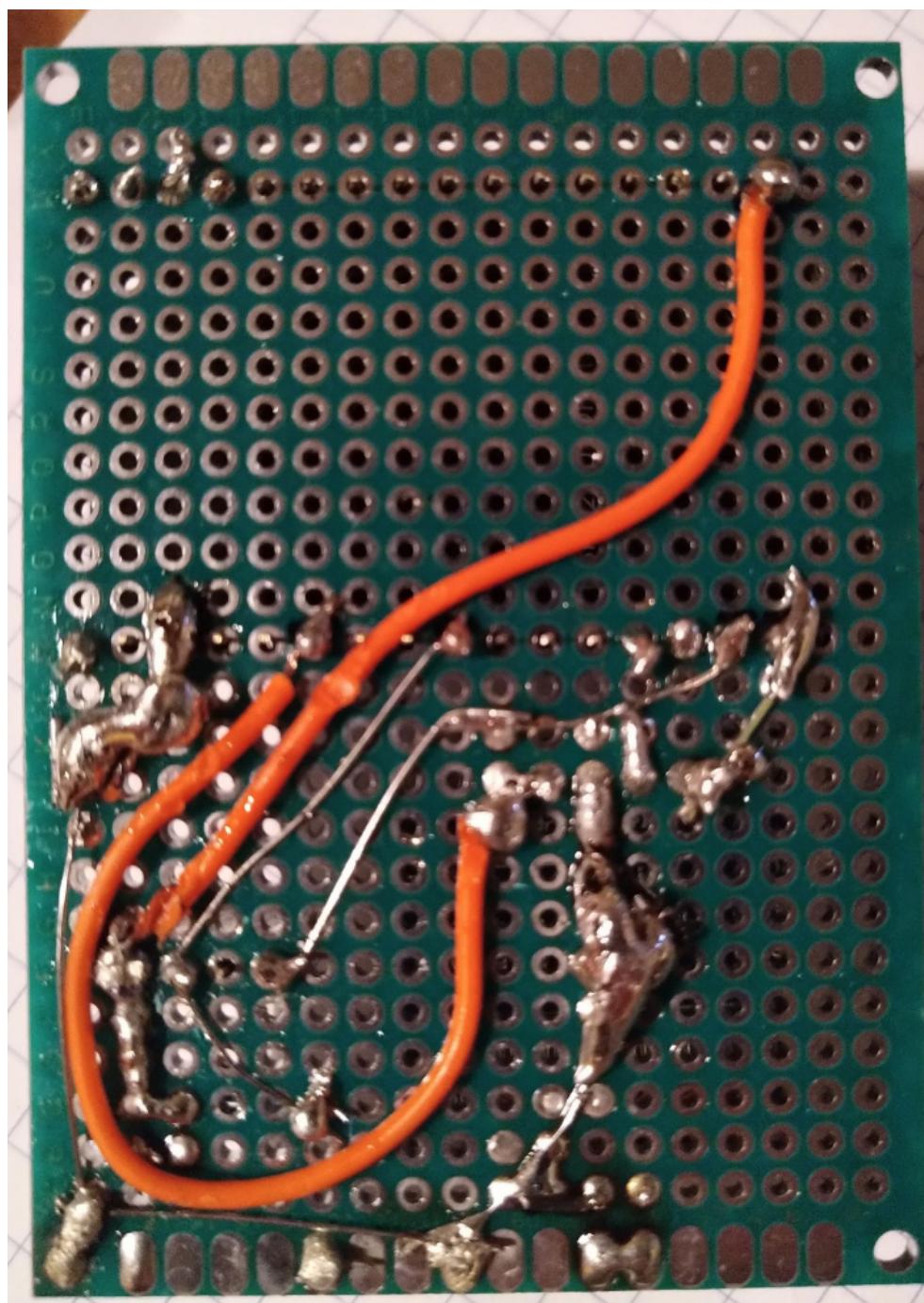
Do aplikacji możemy wejść poprzez naciśnięcie notyfikacji lub z panelu programów zainstalowanych w systemie:

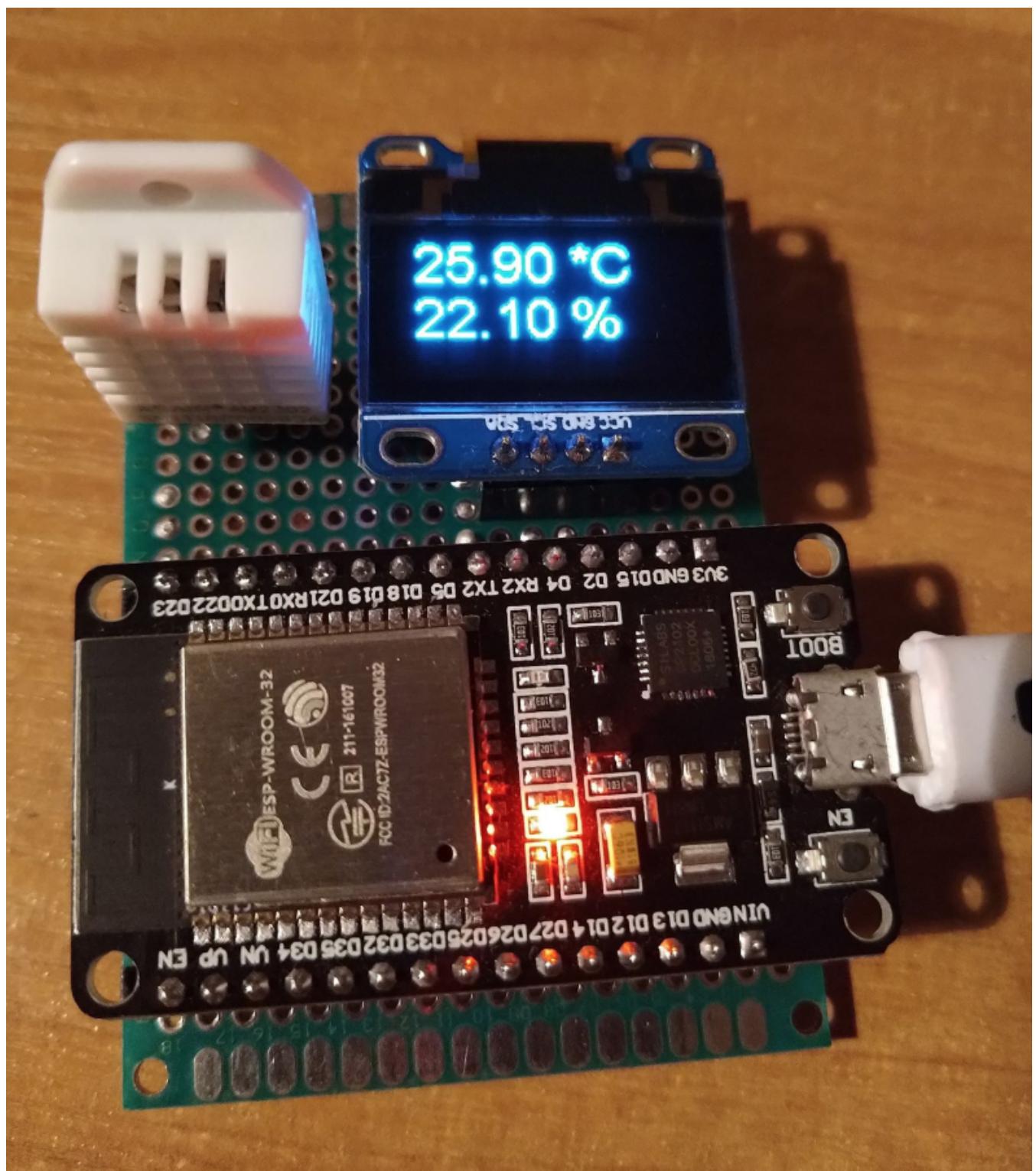


Widok wykonany jest w material design. Poza informacjami o aktualnej temperaturze(na samej górze), zakresem edytowalnym przez użytkownika(na środku), widnieje tu również informacja o pogodzie na zewnątrz(na samym dole aplikacji)

4. Opis montażu układu, sposobu uruchamiania oraz testowania

Układ został złączny na płytce uniwersalnej w sposób przedstawiony na zdjęciach:





Sposób uruchamiania jest bardzo łatwy, wystarczy nam zwykły kabel(usb-micro-usb)

5. Napotkane problemy.

1. Jakość kabla USB - na początku nie wiedziałem, że ma to takie duże znaczenie(brownout detection podczas programowanie esp32 używając kabla słabej jakości)
2. Standardowa biblioteka dla DHT22 często źle analizowała informacje, na początku bałem się, że jest coś nie tak z czujnikiem - na szczęście zmiana biblioteki przyniosła pożądane efekty.
3. Aplikacja bazowo miała być napisana pod system IOS - niestety okazało się, że trzeba posiadać certyfikat dystrybucyjny aby ustawić push notyfikacje dla aplikacji mobilnej.
4. Podczas lutowania początkowo użyłem kabli z plastikową izolacją słabej jakości, która bardzo szybko się topiła podczas przypadkowego dotknięcia grotem lutownicy.
5. Podczas tworzenia aplikacji na google cloud platform trzeba uważać na domyślne ustawienia - często są one bardzo zawyżone, ponad nasze potrzeby. Na przykład - GCP proponuje klastry z trzema replikami i procesorem 3 GHZ - gdzie dla małego serwera MQTT wystarczy 1 replika z 1.7 GHZ.

6. Wnioski

Nad projektem spędziłem bardzo dużo czasu, ale muszę przyznać, że nie był to czas stracony. Poznałem dużo nowych rzeczy, które mogą się przydać w wielu miejscach, nie tylko przy systemach mikroprocesorowych. Między innymi poznałem jak działa protokół MQTT używany często przy IOT(ale nie tylko - np. Messenger od Facebook'a). Dodatkowo nauczyłem się lutować. W większym stopniu zapoznałem się z całym IDE od Arduino, programem Eagle, oraz strukturą elementów, które są zapisane w XML.

8. Literatura oraz repozytoria

<https://github.com/Glaadiss/humidity-tracker-broker> - node.js

<https://github.com/Glaadiss/humidity-tracker-client> - java

<https://github.com/Glaadiss/Humidity-and-Temperature-notificator> - c++

<http://androidkt.com/android-mqtt/>

<https://techtutorialsx.com/2018/04/18/esp32-arduino-getting-temperature-from-a-dht22-sensor/>