



BIURKOWA STACJA POGODOWA

SYSTEMY WBUDOWANE 2021

ADRIAN BEŚCIAK

ALBERT GIERLACH

Spis treści

1 Ogólny zamysł projektu	3
1.1 Hardware	3
1.2 Software	4
1.3 Narzędzia	4
2 Instrukcja użytkownika	5
2.1 Przygotowanie do uruchomienia	5
2.1.1 Podłączenie termometru	5
2.1.2 Podłączenie przewodów	5
2.2 Obsługa urządzenia	6
2.2.1 Pobranie pogody dla wybranego miasta	6
2.2.2 Wyświetlanie informacji pogodowych	7
3 Dokumentacja techniczna	9
3.1 Elementy potrzebne do odtworzenia urządzenia	9
3.2 Schemat połączeń	9
3.3 Drzewo katalogów	11
3.4 Konfiguracja CubeMX	11
3.5 Taski FreeRTOS	12
3.6 Debugowanie na konsolę	12
3.7 DS18B20	12
3.8 Ethernet i lwIP	13
3.9 Weather	14
3.10 TouchGFX	15
4 Postęp prac	20
4.1 Początkowa konfiguracja	20
4.2 Integracja z CLion	20
4.3 Wsparcie dla C++	20
4.4 UART	20
4.5 FreeRTOS	21
4.6 Termometr ds18b20	21
4.7 TouchGFX	22
4.8 Ethernet	23
4.9 Prezentacja danych na wyświetlaczu	23

1. Ogólny zamysł projektu

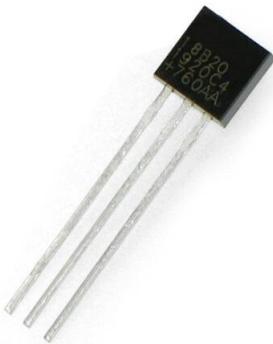
Projekt zakłada stworzenie stacji pogodowej z termometrem pokazującym temperaturę wewnętrz pomieszczenia oraz pobierającej aktualną pogodę z internetu i prezentującą ją na ekranie dotykowym w oparciu o platformę mikrokontrolerową STM32.

1.1. Hardware

- STM32F746G-Disco z wbudowanym wyświetlaczem 4.3" oraz złączem ethernet.
- Termometr DS18B20 do pomiaru temperatury wewnętrz pomieszczenia.



Rysunek 1: STM32F746G-Disco



Rysunek 2: DS18B20

1.2. Software

Planowane jest wykorzystanie systemu FreeRTOS [1]. Do prezentowania informacji zostanie wykorzystana biblioteka graficzna: TouchGFX [2]. W ramach eksperymentu chcemy wykorzystać język C++ oraz integrację środowiska CLion [3] z STM32CubeMX do implementacji tego projektu.

1.3. Narzędzia

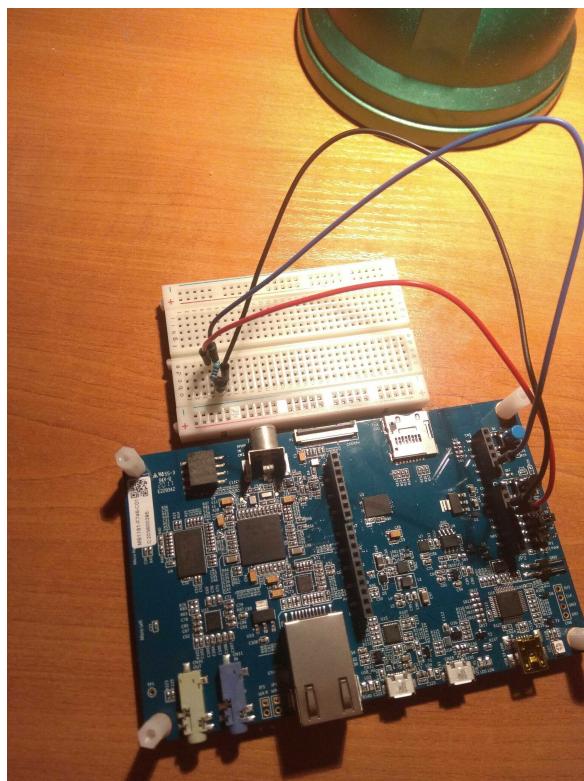
- Github - repozytorium
- Github Projects (Kanban)
- STM32Cube IDE
- CLion
- STM32CubeMX

2. Instrukcja użytkownika

2.1. Przygotowanie do uruchomienia

2.1.1. Podłączenie termometru

Aby podłączyć termometr DS18B20 do płytka, należy podpiąć go do zasilania (odpowiednie piny są wyprowadzone na spodzie płytka - GND oraz 5V), a linię danych termometru podłączamy do pinu oznaczonego jako PA1. Pomiędzy linię danych, a zasilanie wstawiamy rezystor $4.7k\Omega$.



Rysunek 3: Podłączenie termometru do płytka

2.1.2. Podłączenie przewodów

Należy podłączyć urządzenie do sieci Ethernet. Następnie należy podłączyć zasilanie. W tym celu można użyć kabla micro USB i podpiąć go do portu USB HS (obok portu ethernet). Tak jak jest to widoczne na zdjęciu nr 4.



Rysunek 4: Podłączenie zasilania i internetu.

2.2. Obsługa urządzenia

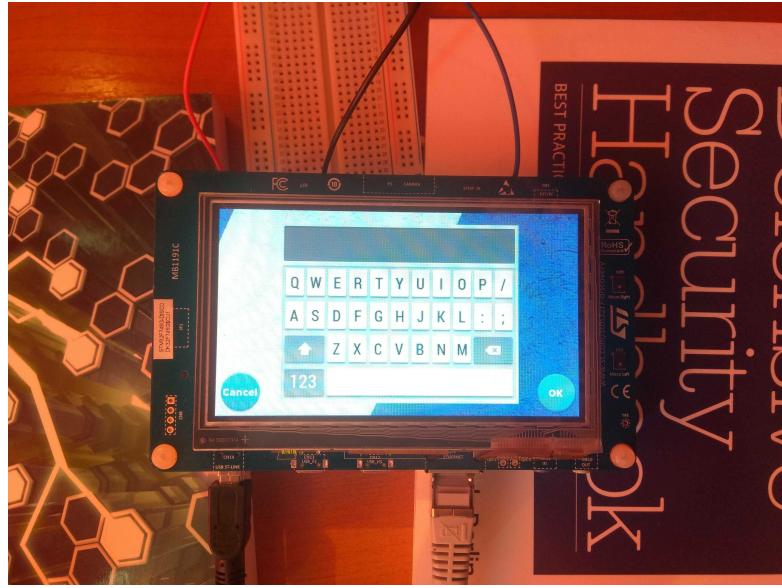
Zaraz po uruchomieniu urządzenia widoczny jest ekran główny (rysunek 5).



Rysunek 5: Ekran startowy urządzenia

2.2.1. Pobranie pogody dla wybranego miasta

W pierwszej kolejności należy wpisać nazwę miasta, dla którego chcemy pobrać pogodę. W tym celu klikamy w pole tekstowe z napisem *wpisz nazwę miasta* i na klawiaturze ekranowej (rysunek 6) wpisujemy interesującą nas nazwę miasta, po czym zatwierdzamy wybór przyciskiem *Ok*.



Rysunek 6: Klawiatura do wyboru miasta

Następnie wybieramy przycisk *Sprawdź pogodę*. W razie wpisania błędnej nazwy miasta, w polu *Status* pojawia się komunikat o błędzie tak, jak jest to widoczne na zdjęciu nr 7 w przeciwnym wypadku pojawi się status *Pobrano pogodę*.



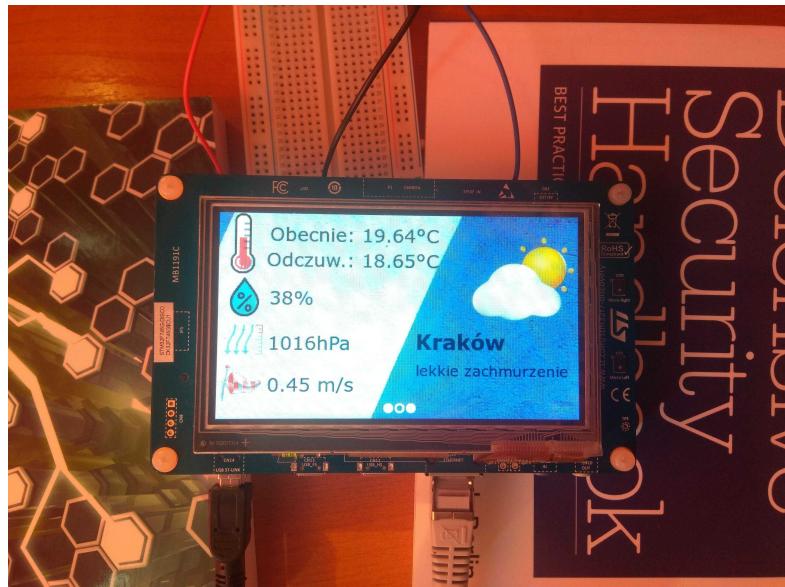
Rysunek 7: Błąd wyboru miasta

Po prawidłowym pobraniu pogody, będzie ona aktualizowana automatycznie co ok. 15s.

2.2.2. Wyświetlanie informacji pogodowych

Aby wyświetlić informacje pogodowe, należy przesunąć palcem w prawo. Zobaczmy wtedy obraz taki jak na zdjęciu nr 8. Na tym ekranie widzimy prognozowaną oraz odczuwalną obecnie

temperaturę w wybranym miejscu, wilgotność powietrza, ciśnienie atmosferyczne oraz prędkość wiatru.



Rysunek 8: Pogoda w Krakowie

Po kolejnym przesunięciu w prawo zobaczymy dodatkowe informacje na temat czasu wschodu i zachodu słońca w dniu dzisiejszym, widoczności oraz temperatury odczytanej z termometru dołączonego do urządzenia (zdjęcie nr 9).



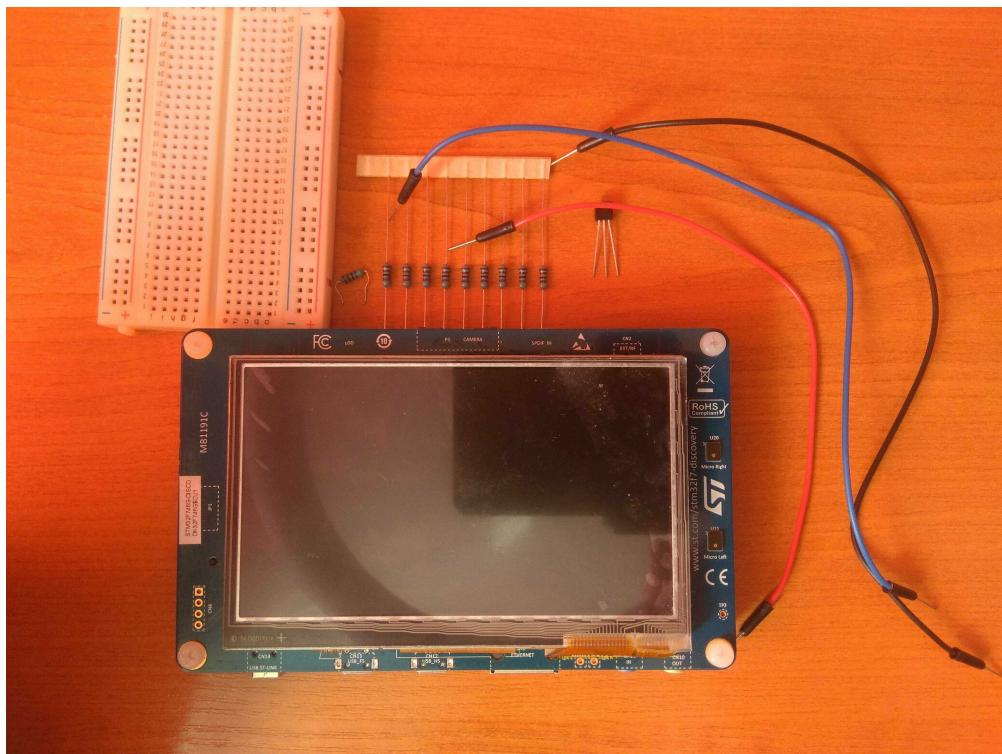
Rysunek 9: Dodatkowe informacje pogodowe

Aby wrócić do wcześniejszego ekranu wystarczy przesunąć palcem w lewo.

3. Dokumentacja techniczna

3.1. Elementy potrzebne do odtworzenia urządzenia

- płytka deweloperska STM32F746G-Discovery
- termometr DS18B20
- rezystor $4.7k\Omega$
- 3 przewody połączeniowe
- płytka stykowa lub łączenie elementów metodą na pajaka
- kabel ethernet oraz dostęp do sieci z serwerem DHCP
- kabel mini USB do programowania układu
- opcjonalnie kabel micro USB do podłączenia zewnętrznego zasilacza z pominięciem ST-linka

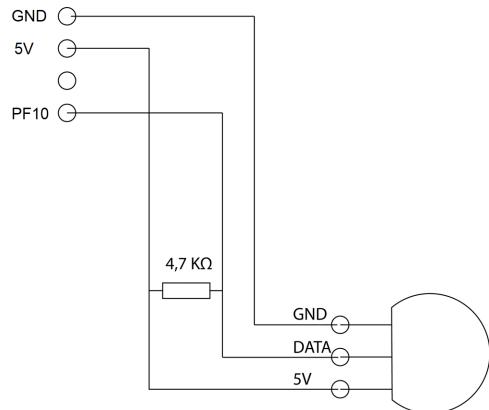


Rysunek 10: Elementy potrzebne do budowy urządzenia

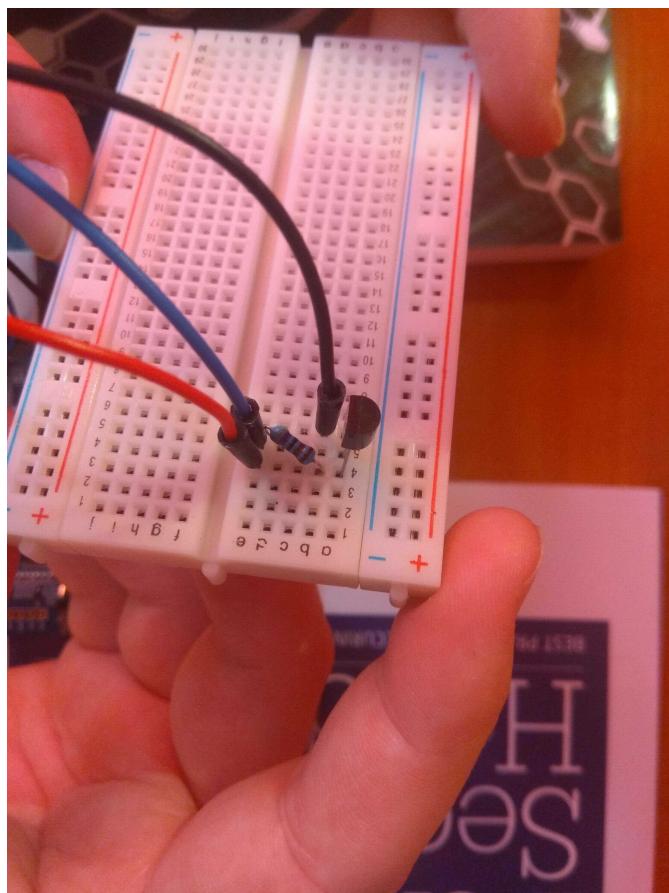
3.2. Schemat połączeń

Termometr DS18B20 podłączliśmy do portu PA1 złącza Arduino na płytce, który odpowiada pinowi PF10 mikrokontrolera, a zasililiśmy go napięciem 5V. Ponadto linię danych

podciągnęliśmy do zasilania przez rezystor $4.7k\Omega$ tak, jak jest to widoczne na rysunku 13.

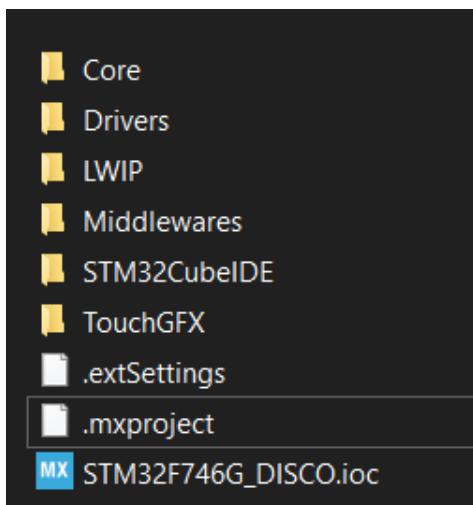


Rysunek 11: Schemat podłączenia termometru



Rysunek 12: Podłączony termometr na płytce stykowej

3.3. Drzewo katalogów



Rysunek 13: Schemat podłączenia termometru

Najważniejsze elementy:

- plik .ioc - zawiera konfigurację CubeMX
- folder TouchGFX - pliki projektu w TouchGFX Designer oraz wygenerowane pliki opisujące interfejs graficzny
- folder STM32CubeIDE - projekt w środowisku STM32Cube IDE, który należy zimportować do środowiska STM32 Cube IDE

3.4. Konfiguracja CubeMX

W programie CubeMX zostały skonfigurowane następujące elementy:

- **FreeRTOS** z podstawą czasu na Timerze 6
- **USART1** służący do wyświetlania logów na konsolę, poprzez nadisaną funkcję `_write()`.
- **Pin PF10** jako wejście GPIO dla lini danych termometru, na którym został później uruchomiony programowy OneWire
- **Ethernet** skonfigurowany w trybie *RMII* w domyślnym trybie, ze zmienionym polem *PHY Address* z wartości 1 na 0 oraz włączonym trybem *Interrupt Mode* w polu *Rx Mode*
- **lwIP** w wersji 2.1.2 z włączoną obsługą DNS w polu *LWIP_DNS*

Ponadto program TouchGFX zainicjalizował dużo peryferiów (np. *QUADSPI* do obsługi zewnętrznej pamięci SDRAM), których nie będziemy wymieniać, ponieważ było to automatyczne działanie programu.

3.5. Taski FreeRTOS

W systemie działają cztery główne taski. Są to:

- **ds18b20_task** - komunikacja z termometrem
- **network_task** - komunikacja z zewnętrznym API OWM
- **TouchGFX_Task** - odpowiada za wyświetlanie obiektów na ekranie
- task związany z **lwIP** - zarządzanie komunikacją TCP (automatycznie tworzony, gdy jest skonfigurowane lwIP)

3.6. Debugowanie na konsolię

Z racji faktu, że w wykorzystywanym przez nas mikroprocesorze istnieje błąd w strukturze krzemowej, uniemożliwiający korzystanie z debugera step-by-step, postanowiliśmy wykorzystać protokół UART do logowania postępów w wykonaniu programu. Najprościej można takie działanie osiągnąć korzystając z standardowych funkcji wyświetlających dane na konsolę, jednak z wiadomych przyczyn, rozwiązań to nie jest od razu dostępne na mikrokontrolerze.

Aby uzyskać taką możliwość, skonfigurowaliśmy USART1, aby działał w trybie asynchronicznym, z Baud Ratem 115200 i w trybie 8N1. Dzięki temu mogliśmy przesyłać dane z mikrokontrolera do komputera. Po nadpisaniu funkcji `_write()`, która domyślnie jest zdefiniowana z atrybutem `weak`, funkcja printf zaczęła przekazywać dane na port szeregowy.

```
1 int _write(int file, char *ptr, int len)
2 {
3     HAL_StatusTypeDef status = HAL_UART_Transmit(&huart1, (uint8_t *)ptr, len, 10);
4     return (status == HAL_OK ? len : 0);
5 }
```

3.7. DS18B20

Obsługa odczytu temperatury z termometru została zrealizowana w oparciu o mechanizm tasków. Co określony interwał czasu mikrokontroler odpytuje termometr o bieżące dane. Do komunikacji wykorzystana została biblioteka OneWire, która została przeportowana z innego modelu mikrokontrolera. Zaraz po uruchomieniu tasku szukane są dostępne termometry i zostają one zapamiętane w globalnej tablicy `thermometers` i są dostępne po załączaniu pliku nagłówkowego `ds18b20.hpp`.

Klasa reprezentująca termometr wygląda następująco:

```
1 class Ds18b20;
2 extern std::array<Ds18b20, _DS18B20_MAX_SENSORS> thermometers;
3
4 class Ds18b20 {
5 private:
6     static constexpr float TEMPERATURE_CORRECTION = 4.0;
```

```

7
8     uint8_t address_[8];
9     float temperature_;
10    bool valid_;
11
12 public:
13     Ds18b20() noexcept;
14     ~Ds18b20() = default;
15
16     [[nodiscard]] uint8_t* getAddress();
17     [[nodiscard]] float getTemperature() const;
18     [[nodiscard]] bool isValid() const;
19     void setTemperature(float temperature);
20 };

```

Całość kodu zawierają pliki:

- Modules/include/ds18b20.hpp
- Modules/src/ds18b20.cpp

3.8. Ethernet i lwIP

Do obsługi internetu został stworzony specjalny task *internetConnectionThread*, którego zadaniem jest inicjalizacja biblioteki LWIP, zarejestrowanie urządzenia w usłudze DHCP sieci lokalnej oraz komunikacja z API pogodowym udostępnionym przez OpenWeatherMap. Ponadto po odebraniu danych z serwera w formacie JSON, są one parsowane przy użyciu biblioteki Parson. Przykład danych zwracanych przez API widzimy poniżej:

```

1  {
2      "coord":
3      {
4          "lon":19.9167,
5          "lat":50.0833
6      },
7      "weather":
8      [
9          {
10             "main":"Clear",
11             "description":"bezchmurnie",
12             "icon":"01n"
13         },
14         "main":
15         {
16             "temp":280.48,
17             "feels_like":280.48,
18             "temp_min":277.57,
19             "temp_max":282.34,
20             "pressure":1020,
21             "humidity":97
22         },
23         "visibility":1200,
24     }
25 }

```

```

23         "wind":  
24     {  
25         "speed":1.03,  
26         "deg":0  
27     },  
28     "sys":  
29     {  
30         "country":"PL",  
31         "sunrise":1621824186,  
32         "sunset":1621881095  
33     },  
34     "timezone":7200,  
35     "name":"Kraków",  
36     "cod":200  
37 }

```

Do komunikacji z serwerami korzystamy z Socket API w stylu Unixowym, które jest udostępniane przez bibliotekę lwIP.

Całość kodu zawierają pliki:

- Modules/include/ethernet.hpp
- Modules/src/ethernet.cpp
- Modules/src/network_task.c

3.9. Weather

Struktura reprezentująca pogodę jest kontenerem, który przechowuje pola dotyczące parametrów pogody. Dane, które znajdują się w tej strukturze są wypełniane po sparsowaniu danych odebranych przez task internetowy.

Klasa przechowująca dane wygląda następująco:

```

1 typedef enum Weather_Error_t {  
2     OK = 0,  
3     EMPTY_LOCATION,  
4     GETHOSTBYNAME,  
5     CONNECT,  
6     SEND,  
7     RECV_FAIL,  
8     RECV_EMPTY,  
9     DOWNLOADING,  
10    NO_CITY  
11 } Weather_Error_t;  
12  
13 typedef enum Weather_t {  
14     CLEAR = 0,  
15     CLOUDS,  
16     DRIZZLE,  
17     FOG,

```

```

18     HAZE,
19     RAIN,
20     SNOW,
21     THUNDERSTORM
22 } Weather_t;
23
24 constexpr uint32_t MAX_DESCRIPTION_LEN      = 64;
25 constexpr uint32_t MAX_CITY_NAME          = 64;
26 constexpr uint32_t MAX_MAIN_NAME         = 64;
27
28 typedef struct weather_t {
29     double temperature;                      // in celsius
30     double feels_like;                     // in celsius
31     uint32_t pressure;                     // hPa
32     uint32_t humidity;                    // %
33     double visibility;                   // km
34     double wind_speed;                  // m/s
35     time_t sunrise;                      // linux timestamp
36     time_t sunset;                       // linux timestamp
37     Weather_t status;                   // enum status
38     char main[MAX_MAIN_NAME];           // main weather text
39     char desc[MAX_DESCRIPTION_LEN];       // in PL
40     char city[MAX_CITY_NAME];            // city name
41
42     Weather_Error_t error;
43 } weather_t;

```

Struktura jest widoczna globalnie po dodaniu pliku nagłówkowego *weather.hpp*.

Całość kodu zawierają pliki:

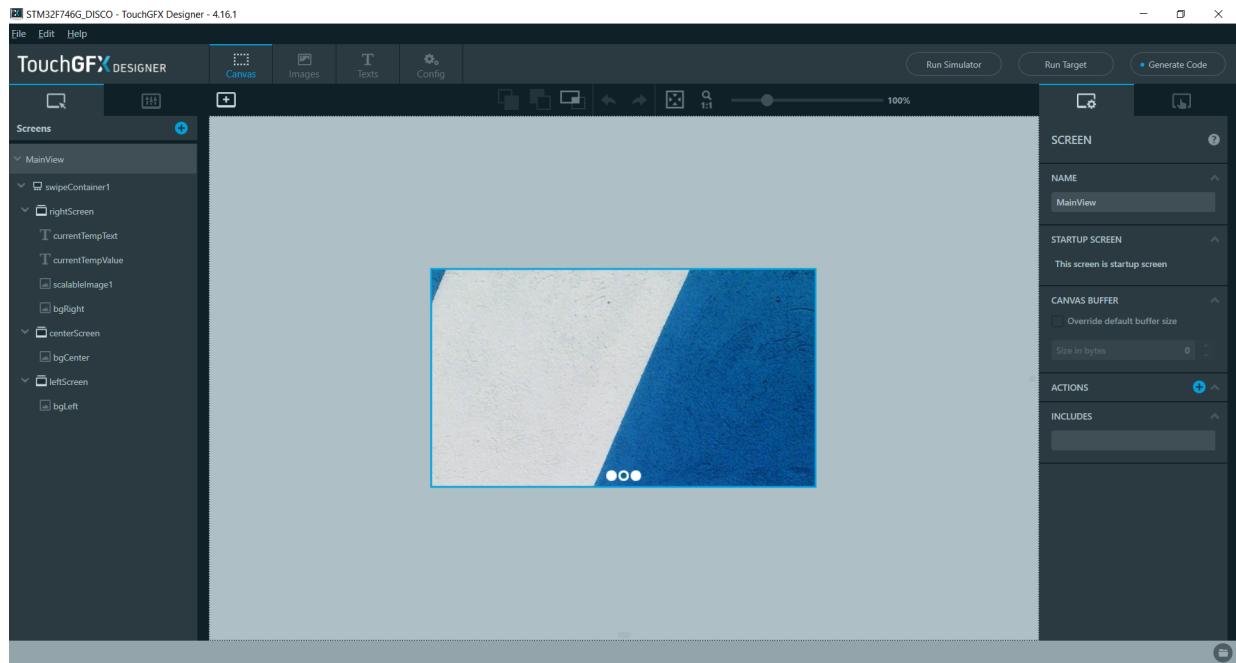
- Modules/include/weather.hpp
- Modules/src/weather.cpp

3.10. TouchGFX

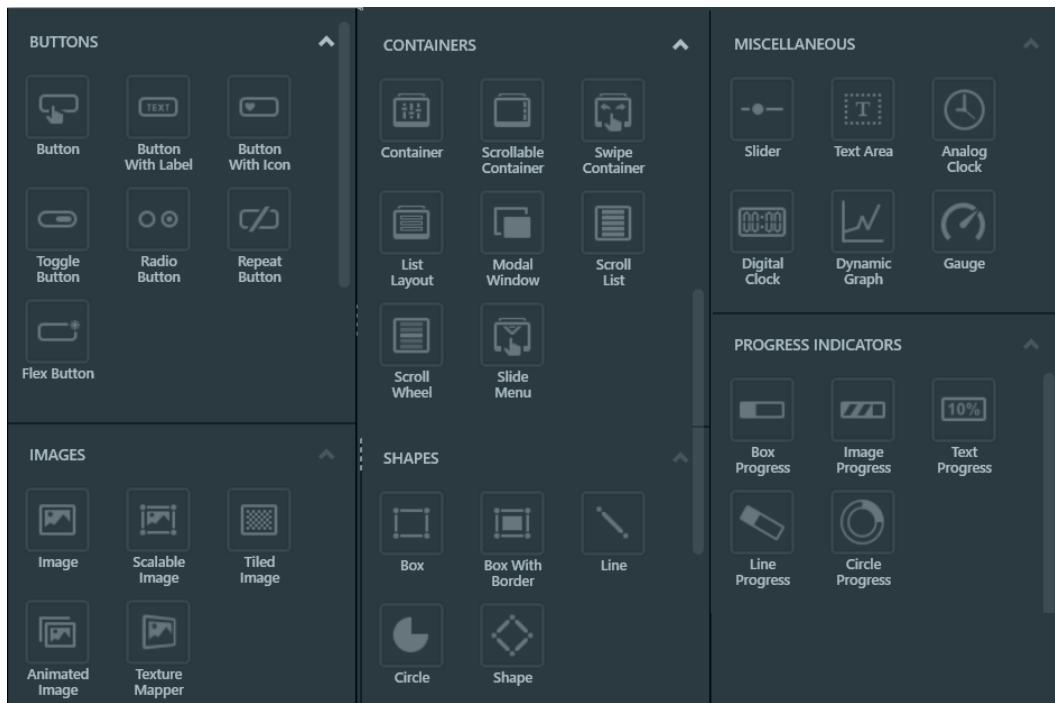
TouchGFX jest zaawansowanym darmowym narzędziem służącym do budowania interfejsów graficznych dla wyświetlaczy mikrokontrolerów STM32. Jego podstawowym zadaniem jest wydajne wspieranie tworzenia interfejsów. Wspiera on zarówno wyświetlacze o niskich rozdzielczościach jak i tych większych oraz wykorzystującą akcelerację sprzętową. Dla zapewnienia płynnego działania aplikacji wymagana jest określona ilość pamięci. TouchGFX jest zintegrowany z całym ekosystemem STM32 i zapewnia łatwy oraz szybki proces rozwoju aplikacji z wykorzystaniem znanych narzędzi projektowych. Sama aplikacja udostępnia kilka pokazowych projektów.

Jedną z aplikacji wchodzących w pakiet TouchGFX jest TouchGFX designer, który umożliwia wygodne tworzenie interfejsów graficznych poprzez wybieranie interesujących elementów i nanoszenie ich na przestrzeń roboczą w projekcie. Kreator graficzny zawiera kilkanaście podstawowych opcji takich jak:

- tworzenie kolejnych ekranów
- zarządzanie czcionkami
- zarządzanie biblioteką obrazów
- generowanie kodu
- komplikacja i uruchomienie projektu na urządzeniu docelowym
- uruchomienie projektu na symulatorze
- definiowanie akcji (*onButtonClick, onScreenStart*)
- definiowanie animacji



Rysunek 14: Okno kreatora graficznego



Rysunek 15: Dostępne widgety

The screenshot shows the TouchGFX Designer interface with the "Resources" tab selected. The top navigation bar includes "Single Use", "Resources" (selected), and "Typographies". Below the tabs, there are sections for "Actions" and "Translations".

Actions section:

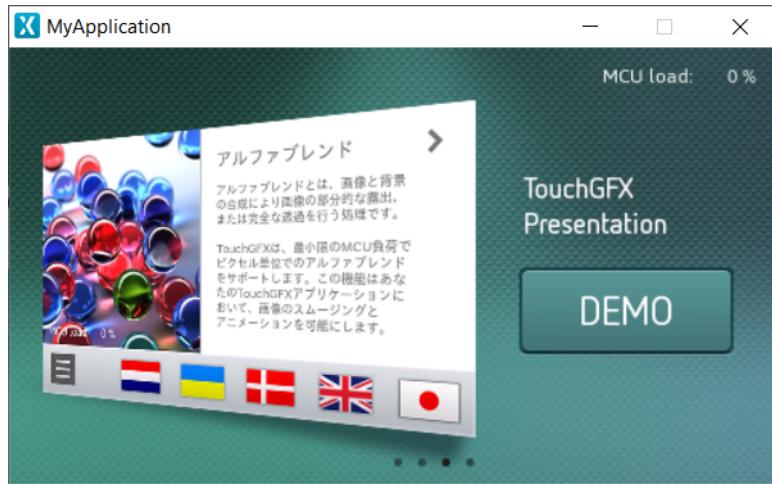
- ADD NEW LANGUAGE
- ADD NEW RESOURCE
- DELETE SELECTED RESOURCES

Translations section:

	Uses	Resource ID	Typography	Alignment	Direction	GB	
<input type="checkbox"/>	1	tempInsideWildcard	Wildcard, 30px	<input type="button" value="A"/>	<input type="button" value="B"/>	<v>	<input type="button" value="Settings"/>
<input type="checkbox"/>	1	tempInsideText	Default, 28px	<input type="button" value="A"/>	<input type="button" value="B"/>	Temperatura wewn:	<input type="button" value="Settings"/>

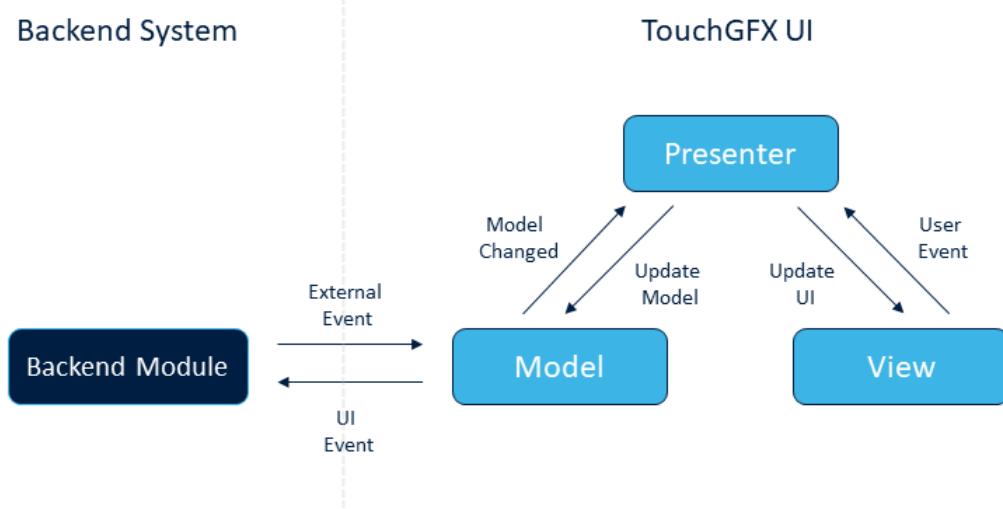
Rysunek 16: Zarządzanie czcionkami

Warto dodać, że narzędzie udostępnia symulator, który pozwala na przetestowanie naszego interfejsu bez konieczności wgrywania programu do mikrokontrolera.



Rysunek 17: Widok symulatora

TouchGFX opiera się w całości o wzorzec MVP (model-view-presenter) [4] co pozwala odseparować część kodu odpowiedzialną za prezentację danych na ekranie od części logicznej, która dostarcza te dane. Dane, które chcemy wyświetlać definiujemy w klasie *Model*, która komunikuje się z *Backendem*, czyli częścią systemu odpowiedzialną za zbieranie i przetwarzanie danych. Klasa *Model* posiada metodę *tick()*, która uruchamiana jest z każdym odświeżeniem (przerysowaniem) ekranu. *Model* odpowiedzialny jest także, za powiadamianie klasy *Presentera* o zmianach danych, który z kolei aktualizuje widok (wywołuje na nim metody zmieniające wewnętrzne dane widoku).



Rysunek 18: Schemat komunikacji we wzorcu MVP

W przypadku projektu *Stacja pogodowa* został zbudowany interfejs oparty o mechanizm *Swipe* (przeciąganie). Dostępne są trzy ekran typu *swipe* oraz jeden zawierający panel z klawiaturą. Klasa *Model* co sekundę odświeża widok i w razie potrzeby odczytuje dane z globalnych zmiennych (*weatherForecast* oraz *thermometers*). Aby otworzyć projekt w programie

TouchGFX należy zainportować plik o nazwie *MyApplication.touchgfx*.

Całość kodu zawierają pliki w folderach:

- TouchGFX/gui/include
- TouchGFX/gui/src

4. Postęp prac

4.1. Początkowa konfiguracja

W początkowej fazie wygenerowany został kod projektu w środowisku STM32CubeMX. Funkcjonalności, które zostały skonfigurowane to:

- FreeRTOS
- LWIP
- USART
- Ethernet

Projekt został skompilowany oraz układ został zaprogramowany za pomocą środowiska STM32CubeIDE. Zweryfikowano także możliwość debugowania programu.

4.2. Integracja z CLion

Następnym krokiem była integracja wygenerowanego projektu z środowiskiem CLion. Po wstępnej analizie okazało się, że środowisko od wersji 2019 posiada wbudowaną wtyczkę umożliwiającą import projektów generowanych przez STM32CubeMX. Konfiguracja wymagała zmiany kilku opcji takich jak:

- ścieżka do kompilatora (i narzędzi z tej rodziny) gcc-arm
- ścieżka do programu OpenOCD
- konfiguracja debuggera (gdb)
- poprawa generowania piku CMakeLists.txt

Po konfiguracji projekt został skompilowany, a następnie układ został zaprogramowany. Sprawdzona została obsługa debugowania bezpośrednio ze środowiska CLion.

4.3. Wsparcie dla C++

Po wstępnych przygotowaniach przetestowane zostało wsparcie dla języka C++. Testy przebiegły pomyślnie i projekt może wykorzystywać kod napisany w języku C++. Warto dodać, że aby można było odwołać się z pliku *.c do funkcji z pliku *.cpp należy zmienić sposób linkowania danej funkcji (*export C*), ponieważ kompilator dokonuje tzw. name-manglingu, co nie występuje w wyjściu generowanym przez kompilator języka C.

4.4. UART

Kolejnym celem była możliwość wypisywania komunikatów na konsolę za pomocą interfejsu UART. Niestety, w takiej konfiguracji po wywołaniu funkcji *printf* system zawieszał się i

nie wypisywał dalszych danych. Okazało się, że trzeba zwiększyć ilość pamięci stosu dla danego zadania, gdyż *printf* w swojej bazowej wersji wykorzystuje sporo pamięci stosu przy formatowaniu argumentów. Alternatywnym rozwiązaniem może być własna implementacja funkcji *printf*, oferująca okrojone możliwości, ale wykorzystująca mniej zasobów stosu.

4.5. FreeRTOS

W ramach testów został stworzony nowy *task*, który wypisywał liczbę tyknięć na konsole, aby zweryfikować poprawość działania systemu. Testy przebiegły pomyślnie.

4.6. Termometr ds18b20

Następnym krokiem była wstępna obsługa termometru ds18b20 oraz wstępne zapoznanie z dokumentacją tego układu [5]. W tym celu wykorzystaliśmy dostępną bibliotekę do obsługi interfejsu 1-Wire, a następnie bibliotekę do obsługi samego termometru. Część kodu musiała zostać poprawiona, gdyż oryginalna wersja biblioteki dotyczyła innego modelu mikrokontrolera. Dodatkowym problemem była integracja obsługi termometru z systemem FreeRTOS. Ostatecznie problem został rozwiązany poprzez mechanizm *tasków*. Biblioteka wymagała konfiguracji licznika, dzięki któremu może ona określać czas, który należy odczekać do momentu odczytania danych z termometru.



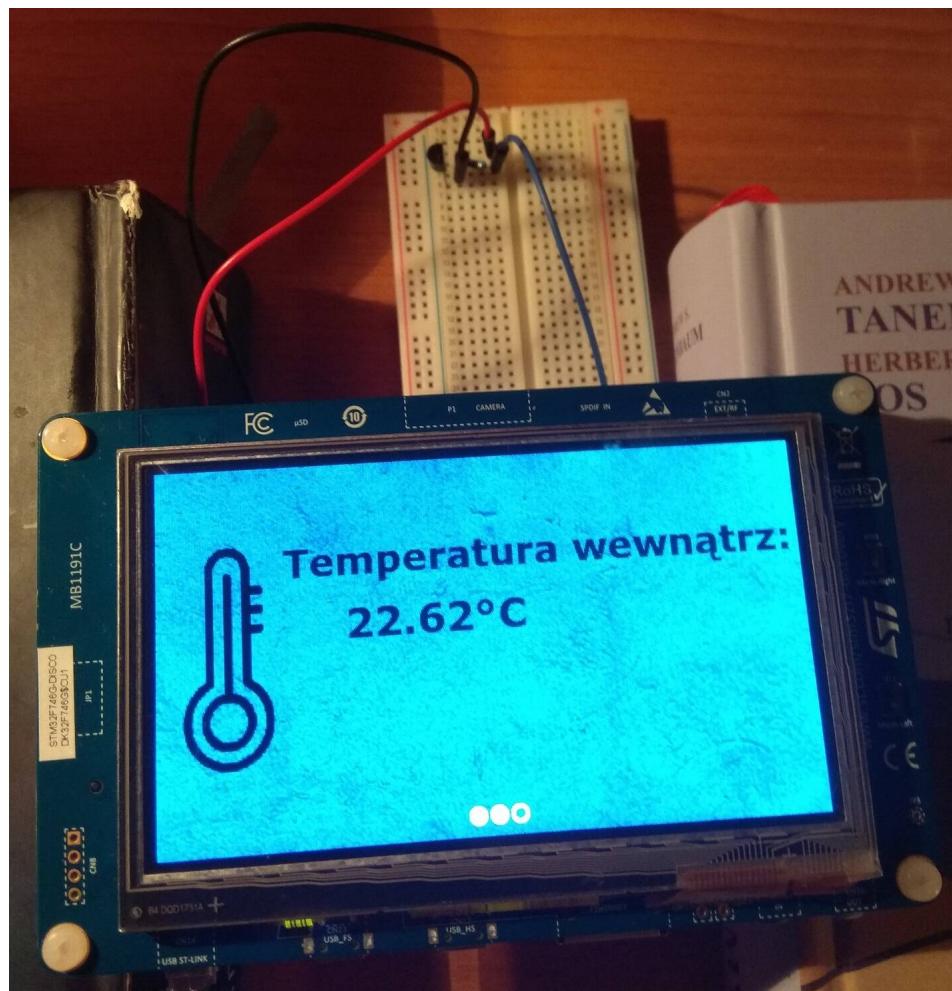
```
Temp: 28.25
Temp: 27.75
Started!
Task Ds18b20
Temp: 27.31
Temp: 27.25
Temp: 27.19
Temp: 27.06
Temp: 27.00
Temp: 26.88
Temp: 26.88
Temp: 26.88
Temp: 26.94
Temp: 26.94
Temp: 26.94
Temp: 26.88
Temp: 26.88
Temp: 26.75
Temp: 26.75
```

Rysunek 19: Uruchomiony program odczytujący temperaturę

4.7. TouchGFX

Kolejnym krokiem było przygotowanie oraz konfiguracja środowiska TouchGFX [2]. Niestety po wielu próbach nie udało się zintegrować projektu, który generuje narzędzie TouchGFX Designer [6], ze środowiskiem programistycznym CLion. Struktura katalogów oraz pliki projektu zbyt mocno odbiegają od tej generowanej przez STM32Cube IDE. Kod został przeniesiony ze starej wersji projektu do tej wygenerowanej przez TouchGFX. Warto dodać, że obie wersje korzystały z FreeRTOS'a co znacznie uprościło migrację.

Integracja bieżącego postępu prac z nowym projektem przebiegła pomyślnie. Został zbudowany prototyp interfejsu oparty o mechanizm *Swipe* (przeciąganie). Dostępne są trzy ekranы - główny, panel z bieżącą temperaturą wewnętrzną oraz z informacjami nt. pogody w ustalonej lokalizacji. Zaimplementowano także automatyczne odświeżanie temperatury na ekranie (prototyp uwzględniał ręczne odświeżanie poprzez wcisnięcie przycisku na ekranie).



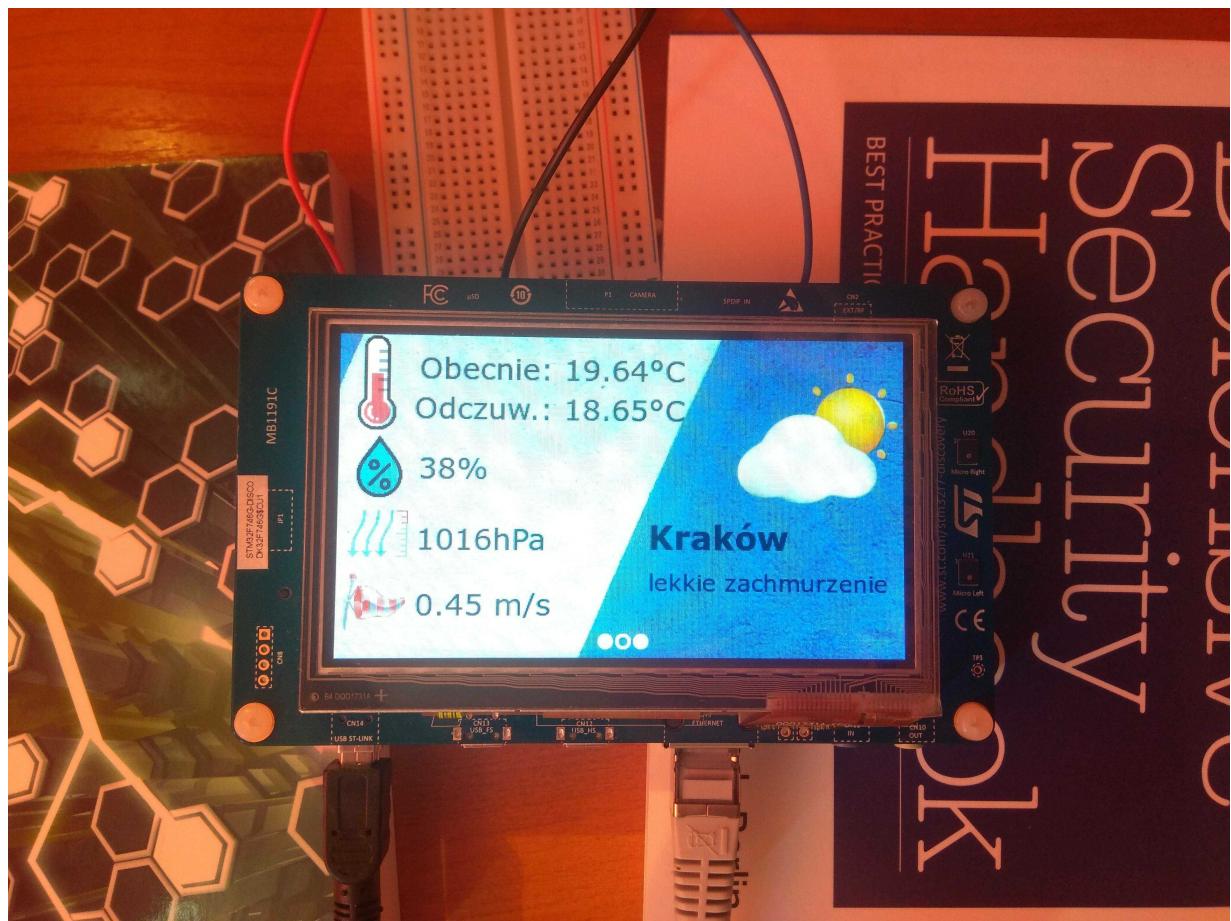
Rysunek 20: Zaktualizowana wartość temperatury

4.8. Ethernet

Kolejnym krokiem było zaimplementowanie obsługi połączenia z internetem. Po kilku nieudanuych próbach wykorzystania API udostępnianych przez LWIP, zdecydowaliśmy się na Socket API, działające tak jak w systemie UNIX. Dzięki temu zabiegowi udało się połączyć z internetem i pobrać dane o pogodzie z serwisu Open Weather Map [7]. Cała logika jest zawarta w tasku FreeRTOS i działa asynchronicznie. Dane przychodzące z API OWM są w formacie JSON. Do parsowania tych danych wykorzystaliśmy bibliotekę Parson [8].

4.9. Prezentacja danych na wyświetlaczu

Ostatnim etapem było zaprezentowanie danych na wyświetlaczu mikrokontrolera. Zostały stworzone komponenty odpowiadające za wyświetlanie danych za pomocą ikon i tekstu. Dodano także obsługę wprowadzania nazwy miasta za pomocą klawiatury. Dodatkowo została wprowadzona funkcjonalność, która poinformuje użytkownika jeśli wystąpi jakiś błąd w komunikacji z internetem i dane o pogodzie nie zostaną pobrane pomyślnie. Etap ten zajął najwięcej czasu.



Rysunek 21: Wyświetlone dane pobrane z API

Bibliografia

- [1] *FreeRTOS*. URL: <https://www.freertos.org/>.
- [2] *TouchGFX Documentation*.
URL: <https://support.touchgfx.com/docs/introduction/welcome>.
- [3] *CLion*. URL: <https://www.jetbrains.com/clion/>.
- [4] *Model-view-presenter*.
URL: <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter>.
- [5] *DS18B20 Programmable Resolution 1-Wire Digital Thermometer*.
URL: <https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf>.
- [6] *TouchGFX Designer*.
URL: <https://www.st.com/en/development-tools/touchgfxdesigner.html>.
- [7] *Open Weather Map API*. URL: <https://openweathermap.org/api>.
- [8] *Parson*. URL: <https://github.com/kgabis/parson>.