|  |
| --- |
| [Nazwa firmy] |
| Strona Tytułowa |
| Placeholder |

|  |
| --- |
| Szkorla Radosław |

**Spis Treści**

[1. Wstęp 2](#_Toc6582166)

[2. Cel i zakres pracy 3](#_Toc6582167)

[3. Analiza wymagań funkcjonalnych 4](#_Toc6582168)

[4. Technologie realizacji 5](#_Toc6582169)

[4.1. Technologie Sprzętowe 5](#_Toc6582170)

[4.2. Technologie programistyczne 7](#_Toc6582171)

[4.3. Narzędzia 10](#_Toc6582172)

[5. Projekt systemu oraz realizacja 12](#_Toc6582173)

[6. Podsumowanie 23](#_Toc6582174)

[7. Spis ilustracji 24](#_Toc6582175)

[8. Literatura 25](#_Toc6582176)

# Wstęp

Szczepienia ochronne to jedno z najważniejszych osiągnięć w historii ludzkości oraz medycyny. Dziś trudno wyobrazić sobie, że takie choroby, jak świnka, odra czy ospa prawdziwa jeszcze 100 lat temu zbierały milionowe żniwo na całym świecie. Obecnie wiele osób przypomina sobie o istnieniu tychże chorób dopiero w punkcie szczepień. Wydaje się, że wakcynologia to stosunkowo młoda dziedzina nauki. Tymczasem szczepionki mają już ponad 200 lat. Historia szczepień zaczyna się przed 1800 rokiem, kiedy to rozpoczęły się pierwsze masowe szczepienia ochronne przeciw ospie prawdziwiej [6]. Twórcą tej szczepionki był Edward Jenner. W 1796 r. Jenner przeprowadził ryzykowny eksperyment. [6] Zaraził ośmioletniego chłopca krowią odmianą ospy. Po nadzwyczaj łagodnym przebiegu choroby chłopiec wyzdrowiał, a po roku okazało się, że został uodporniony również na ospę prawdziwą. Działania te uratowały miliony ludzi oraz doprowadziły do całkowitego wyeliminowania tego śmiercionośnego wirusa z naszego globu. Od tamtego czasu wiele naukowców i lekarzy prowadzili intensywne badania, mające na celu rozwój nowych rodzajów uodparniania i metod zapobiegania zarażeniom. Efektem tych prac są szczepionki, które obecnie chronią przed 25 chorobami zakaźnymi, między innymi przed wścieklizną, różyczką, WZW A czy grypą.

Obecnie jednak coraz szersza grupa osób kwestionuje sensowność szczepień. Ruchy antyszczepionkowe poruszają szereg argumentów przeciw obowiązkowym szczepieniom ochronnym [4]. Jednym z nich jest argument mówiący o przechowywaniu preparatów do iniekcji w złych warunkach. Kwestia magazynowania szczepionek jest niezwykle istotna dla zapewnienia zdrowia i życia pacjentów, w tym zapobieganiu występowania NOPów, czyli niepożądanych odczynów poszczepiennych [2]. Dlatego w przechowywaniu specyfików ważne są systemy, które stałe monitorują temperatury w jakich przechowywane są szczepionki, aby te mieściły się w zadanym przedziale, zgodnie z zaleceniami producentów i specjalistów, a w razie problemów umożliwić szybką reakcję osób odpowiedzialnych za bezpieczeństwo.

# Cel i zakres pracy

Celem pracy jest stworzenie systemu, który w głównej mierze będzie monitorował panującą temperaturę w magazynie szczepionek oraz powiadamiał wskazane osoby o przekroczonych wartościach. Ma to skutkować podjęciem działań prze te osoby zapobiegających przechowywaniu preparatów do szczepień w niekorzystnych warunkach [2]. Złe składowanie szczepionek może stanowić bardzo poważne zagrożenie dla życia i zdrowia osób, którym je podano, a także dla osób postronnych [3].

System składa się z dwóch podstawowych elementów: serwera monitorującego uruchomionego na Raspberry Pi oraz czujnika temperatury podłączonego do ESP32. Komunikacja między serwerem a czujnikiem odbywa się bezprzewodowo poprzez sieć Wi-Fi i przy użyciu protokołu http. Punkt dostępowy jak i serwer http jest uruchomiony na ESP32.

Podgląd aktualnej temperatury powinien być dostępny dla każdego użytkownika znajdującego się na terenie obiektu. Nie może być on jednak swobodny, dlatego też, aby uzyskać dostęp do strony www, na której owa temperatura jest wyświetlana, należy połączyć się z siecią Wi-Fi stworzoną na ESP32.

System powinien działać niezależnie od stanu zasilania czy dostępności sieci. Dlatego urządzenie, na którym uruchomiony jest serwer monitorujący powinien posiadać niezależne źródło zasilania, najlepiej w postaci urządzenia UPS. Z drugiej strony sam czujnik powinien zostać wpięty do tej samej linii zasilającej urządzenia chłodnicze magazynu, po to, aby awaria zasilania wyłączyła go, co przez serwer zostanie zinterpretowane jako właśnie awaria zasilania i wyśle powiadomienie. Powiadomienia zaś będą wysyłane w formie wiadomości SMS poprzez moduł GSM. Ma to na celu uniezależnienie przekazania informacji od dostępności sieci i Internetu.

# Analiza wymagań funkcjonalnych

# Technologie realizacji

## Technologie Sprzętowe

Raspberry Pi [7]

Sercem systemu jest Raspberry Pi 3 Model B. Raspberry Pi to seria małych komputerów jednopłytkowych opracowanych w Wielkiej Brytanii przez Fundację Raspberry Pi w celu promowania nauczania podstawowej informatyki w szkołach i krajach rozwijających się. Oryginalny model stał się znacznie bardziej popularny, niż się spodziewano. Jego zastosowania wykraczają poza pierwotne założenia i obecnie obok systemów Arduino stanowią podstawę takich gałęzi przemysłu informatycznego jak Internet of Things czy robotyka.

Siłą Raspberry Pi jest możliwość instalacji na nim, w dużym uproszczeniu, klasycznego systemu operacyjnego, najczęściej opartego o system Linux. Umożliwia to użytkowanie go jak klasycznego komputera klasy PC, co jednocześnie daję możliwość tworzenia oprogramowania tak samo jak na tradycyjne platformy.

W projekcie zastosowano Raspberry Pi 3 Model B. Model ten został wydany w lutym 2016 roku. Posiada zintegrowany układ SoC Broadcom BCM 2837, z czterema 64-bitowymi rdzeniami ARM Cortex-A53, taktowanymi zegarem 1,2 GHz. Układ graficzny to Videocore 4 taktowany zegarem 400MHz. Raspberry posiada 1 GB pamięci RAM LPDDR2 taktowanej 900 MHz, która jest dzielona razem z układem graficznym, oraz układ radiowy Broadcom BCM43438, zapewniający połączenie Wi-Fi w standardzie 802.11n na paśmie 2,4 GHz oraz Bluetooth 4.1/LE. Innymi elementami są: kontroler USB/Fast Ethernet (100 Mbit/s) SMSC LAN951, cztery porty USB 2.0, wyjście HDMI, wyjście audio-wideo minijack, złącze kamery CSI, szeregowy interfejs dla wyświetlacza DSI, złącze zasilania microUSB, oraz 40-pinowe złącze GPIO.

ESP32 [8]

ESP32 to seria tanich, niskonapięciowych mikrokontrolerów z wbudowanym Wi-Fi i Bluetooth. Seria ESP32 wykorzystuje mikroprocesor Tensilica Xtensa LX6 zarówno w wersji dwurdzeniowej, jak i jednordzeniowej. Zawiera wbudowane przełączniki antenowe, wzmacniacz mocy, niskoszumowy wzmacniacz odbioru, filtry i moduły zarządzania zasilaniem. ESP32 jest tworzony i rozwijany przez Espressif Systems, i jest produkowany przez TSMC w 40 nm. procesie technologicznym. Jest następcą mikrokontrolera ESP8266.

W projekcie zastosowano układ ESP32-WROOM-32 który posiada dwurdzeniowy, 32 bitowy procesor taktowany na 240 MHz oraz 520 kB pamięci SRAM i 16 MB pamięci Flash. Układ jest wykorzystywany jako przekaźnik pomiędzy czujnikami temperatury a Raspberry Pi. Urządzenie jest programowane przy użyciu Arduino Studio. ESP32 jest serwerem http, który przekazuje dane z czujników za pomocą protokołu http. Ponadto jest także urządzeniem, które tworzy zamkniętą sieć Wi-Fi w standardzie 802.11n, co umożliwia komunikację bezprzewodową. Dzięki niewielkiemu rozmiarowi i niskiemu poborowi prądu może być montowany blisko miejsc zainstalowania czujników.

Czujnik DHT22 [9]

DHT22 to czujnik temperatury i wilgotności. Wykorzystuje pojemnościowy czujnik wilgotności i termistor do mierzenia otaczającego powietrza i wyprowadza sygnał cyfrowy na pin danych. Parametry:

1. Napięcie zasilania: od 3,3 V do 6 V
2. Średni pobór prądu: 0,2 mA
3. Temperatura
   1. Zakres pomiarowy: -40 do 80 °C
   2. Rozdzielczość: 8-bitów (0,1 °C)
   3. Dokładność: ± 0,5 °C
   4. Czas odpowiedzi: średnio 2 s
4. Wilgotność:
   1. Zakres pomiarowy: 0 - 100 % RH
   2. Rozdzielczość: 8-bitów (±0,1 % RH)
   3. Dokładność ±2 %RH\*
   4. Czas odpowiedzi: średnio 2 s

W projekcie wykorzystano tylko czujnik temperatury.

Moduł GSM [10]

Nakładka dopasowana do złącz GPIO Raspberry Pi, która umożliwia komunikację w standardzie GSM, GPRS, GNSS oraz Bluetooth. Pozwala prowadzić rozmowy głosowe, wysyłać wiadomości sms, łączyć się z Internetem, przesyłać dane za pomocą Bluetooth, posiada system nawigacji. Nakładka wyposażona jest w moduł SIM868. Na płytce znajdują się: złącze karty SIM, gniazdo microUSB, złącze jack 3,5 mm, miejsce na baterię oraz złącze anteny u.FL.

Pozostałe elementy

Rezystory 10 kΩ wykorzystywane do połączenia czujników temperatury DHT22 do układu ESP32.

Płytka prototypowa która ułatwia połączenie wszystkich elementów ze sobą za pomocą przewodów.

## Technologie programistyczne

.NET Core [5]

.NET Core jest darmową, wielpolatformową i otwartą platformą tworzoną przez .NET Foundation – organizację utworzoną przez Microsoft w 2014 roku, oraz społeczność. .NET Core umożliwia tworzenie i uruchamianie aplikacji chmurowych, IoT czy klasycznych dla PC lub urządzeń mobilnych. Charakterystyka platformy .NET Core:

1. Wieloplatformowość. Umożliwia uruchamiania aplikacji na systemach operacyjnych Windows, macOS i opartych o Linux.
2. Zgodność między architekturami. Programy działają tak samo na różnych architekturach, w tym x64, x84 i ARM.
3. Narzędzia lini poleceń. Zawiera łatwe w użyciu narzędzia, które można wykorzystać do tworzenia aplikacji.
4. Elastyczne wdrożenie. Aplikacja może być dołączona do innych lub uruchomiona niezależnie oraz z wykorzystaniem kontenerów Docker.
5. Kompatybilność. .NET Core jest zgodny z .NET Framework, Xamarin i Mono, poprzez .NET Standard.
6. Open source. .NET Core wykorzystuje licencje MIT i Apache 2.

Aplikacje i biblioteki .NET Core można tworzyć wykorzystując języki programowani, takie jak C#, Visual Basic, F#.

ASP.NET Core [11]

ASP.NET Core jest wieloplatformowym, wydajnym i otwartym frameworkiem do kreacji nowoczesnych, opartych na rozwiązaniach chmurowych aplikacji. Za jego pomocą można tworzyć aplikacje i serwisy webowe, programy na platformy Internet of Things czy mobilne. Narzędzia do tworzenia rozwiązań przy użyciu ASP.NET są dostępne na wszystkich wiodących systemach operacyjnych, czyli Windows, macOS i Linux. Aplikacje można wrażać do chmury, hostować w usługach IIS, Nginx, Apache, Docker lub na lokalnych serwerach i uruchamiać je za pomocą platform .NET Core lub .NET Framework. Może wykorzystywać wzorzec architektoniczny MVC (Model-View-Controler), który optymalizuje tworzenie dynamicznych aplikacji internetowych, zapewnia czyste rozdzielenie zadań i problemów, wspiera programowanie TDD (Test Driven Development) oraz wykorzystuje najnowsze standardy tworzenia aplikacji webowych. Do kreacji widoków ASP.NET Core używa składni języka Razor, która stanowi połączenie składni HTML oraz języka C#. Oprócz tego umożliwia stosowanie takich frameworków jak Bootstrap, Angular czy React.

SignalR [11]

ASP.NET Core SignalR jest otwartą biblioteką, która zapewnia funkcjonalność w postaci komunikacji dwukierunkowej między klientem a serwerem w czasie rzeczywistym w aplikacji. SignalR dostarcza API do tworzenia zdalnych wywołań procedur w kierunku od serwera do klienta. Serwer stworzony przy użyciu .NET Core wywołuje po stronie klienta funkcje wykreowane za pomocą JavaScript. Najważniejszymi cechami tej biblioteki są: automatycznie zarządza połączeniami, potrafi wysłać polecenia do wszystkich podłączonych klientów, jak również do pojedynczych lub określonych grup klientów, a także posiada mechanizmy skalowania, przydatne zwłaszcza zwiększającej się liczby połączeń. SignalR wspiera kilka technik to tworzenia komunikacji w czasie rzeczywistym: WebSockets, Server-Sent Events i Long Polling oraz autowmatycznie określa, która z nich będzie najlepsza w danym przypadku. Biblioteka wykorzystuje do komunikacji tak zwane huby. Huby umożliwiają klientowi i serwerowi nawiązywanie połączeń między sobą. Klient może wywołać po stronie serwera metody, przesyłając do nich silnie typowane parametry. Do przesyłania parametrów można użyć tekstowego protokołu opartego na JSON lub binarnego protokołu opartego na MessagePack. Serwer natomiast wywołuje kod u klienta, wysyłając wiadomości zawierające nazwę i parametry metody po stronie klienta. Obiekty wysyłane jako parametry są serializowane najczęściej do postaci JSON i po przesłaniu deserializowane. Klient próbuje dopasować nazwę do metody w kodzie po swojej stronie i gdy znajdzie dopasowanie wywołuje tą metodę razem z przesłanymi parametrami.

Chart.js [12]

Chart.js jest otwartą biblioteką, która w łatwy sposób umożliwia tworzenie wykresów na podstawie dostarczonych danych przy użyciu JavaScript. Wspiera osiem różnych typów wykresów, w tym liniowe, słupkowe czy kołowe. Wykresy te są responsywne, co oznacza, że są prezentowane dobrze niezależnie od rozdzielczości ekranu urządzenia, na którym są wyświetlane.

Raspbian [13]

Raspbian jest darmowym systemem operacyjnym, opartym na Debianie i zoptymalizowanym specjalnie dla Raspberry Pi. Raspbian jest nieoficjalnym portem Debiana Wheezy – jednej z dystrybucji systemu Linux. Został przygotowany w taki sposób, aby optymalizować operacje zmiennoprzecinkowe wykonywane na Raspberry. To sprawiło, że aplikacje, które korzystają z tego typu obliczeń zmiennoprzecinkowych działają wyraźnie szybciej. Dzięki tej optymalizacji zyskały także inne aplikacje poprzez zastosowanie zaawansowanych instrukcji procesora ARMv6 w Raspberry Pi. Ten system operacyjny umożliwia zainstalowanie na nim środowiska uruchomieniowego .NET Core, uruchomienie aplikacji monitorującej, jak również komunikację poprzez port szeregowy z modułem GSM.

Bootstrap [14]

Bootsrap jest otwartym frameworkiem do tworzenia m. in. responsywnych (dopasowujących wygląd i treść do rozdzielczości wyświetlacza) stron WWW, przy użyciu HTML, CSS i JavaScript. Pierwotnie stworzony na potrzeby portalu Twitter, Bootstrap stał się jednym z najpopularniejszych frameworków front-end i projektów open source na świecie. Początki sięgają połowy 2010 roku, kiedy to nosił nazwę Twitter Blueprint. 19 sierpnia 2011 roku został udostępniony publicznie. Obecnie istnieje już czwarta wersja wydana w 2018 roku. Znaczącą zmianą była wersja trzecia, gdzie przepisano całość tak by współdziałać z podejściem mobile-first (tworzenie strony najpierw na urządzenia mobilne).

## Narzędzia

Visual Studio 2017 [15]

Serwer monitorujący został stworzony przy pomocy zintegrowanego środowiska programistycznego Visual Studio 2017 firmy Microsoft. Został wydany 7 marca 2017 roku. Jest to rozbudowane narzędzie do tworzenia różnego rodzaju aplikacji (klasycznych desktopowych, mobilnych, opartych na chmurze itd.). Posiada wiele przydatnych funkcji jak wspomagające refaktoryzowanie kodu, IntelliSense, czyli zestaw narzędzi wspomagających pisanie (np. poprzez podpowiedzi po wpisaniu części nazwy klasy), możliwość przejścia do lub podglądu definicji danej metody z poziomu menu kontekstowego lub skrótu klawiszowego, obsługa menedżera pakietów Nuget czy wsparcie dla systemów kontroli wersji takich jak Git lub TFVC. Visual Studio wspiera korzystanie z wielu języków programowania takich jak: C#, F#, Visual Basic, C++, Python i JavaScript. Dodatkowo pozwala także na instalacje dodatkowych rozszerzeń wydanych przez Microsoft (np. Xamarin) jak i firm trzecich (np. ReSharper od JetBrains) zwiększających jego możliwości.

Arduino IDE [16]

Do napisania oprogramowania czujników temperatury wykorzystano Arduino IDE w wersji 1.8.5. Jest to darmowe środowisko do tworzenia oraz wgrywania kodu na urządzenia Arduino, jednak z powodzeniem może być wykorzystywane do pracy z innymi mikrokontrolerami jak np. wykorzystane w projekcie ESP32. Arduino IDE zapewnia dostęp do bibliotek przydatnych w tego typu rozwiązaniach, weryfikację kodu, wgrywanie bezpośrednio do urządzeń, dostosowywanie parametrów komunikacji z mikrokontrolera, konsolę informującą o statusie czy błędach oraz monitor portu szeregowego urządzenia, które może być przydatne np. przy debugowaniu.

Git [17] i github.com

Git jest to darmowy otwarty system kontroli wersji, wspomagający tworzenie oprogramowania. Początki sięgają roku 2005, kiedy to twórcy jądra Linux a przede wszystkim Linus Torvaldis zerwali współpracę z firmą rozwijającą inny system kontroli wersji BitKeepera. Tworząc Git autorzy, na bazie doświadczeń pracy z BitKeeperem starali się stworzyć system który miał być szybki i prosty w obsłudze, wspierać nieliniowe tworzenie kodu, być rozproszony oraz umożliwiać pracę z bardzo dużymi projetkami.

Git tworzy repozytoria i wykorzystuje migawki do zapisywania różnic między kolejnymi wersjami, zwanymi commitami. Git umożliwia prowadzenie procesu deweloperskiego wielotorowo poprzez gałęzie (branches). Każda gałąź przechowuje swoją wersję projektu, które można ze sobą później połączyć (merge).

Github.com jest najpopularniejszym serwisem internetowym, który umożliwia przechowywanie repozytoriów Gita w Internecie. Pozwala na dzielenie się kodem źródłowym. Dostarcza przydatne narzędzia do analizy kodu.

VNC Server i VNC Viewer

VNC jest programem do komunikacji między urządzeniami na zasadzie wirtualnego pulpitu. Na Raspberry Pi zainstalowany jest VNC Server a na komputerze, służącym do tworzenia oprogramowania klient VNC Viewer. Program umożliwia również transfer plików.

Etcher

Program, który posłużył do przygotowania karty pamięci z systemem Raspbian.

# Projekt systemu oraz realizacja

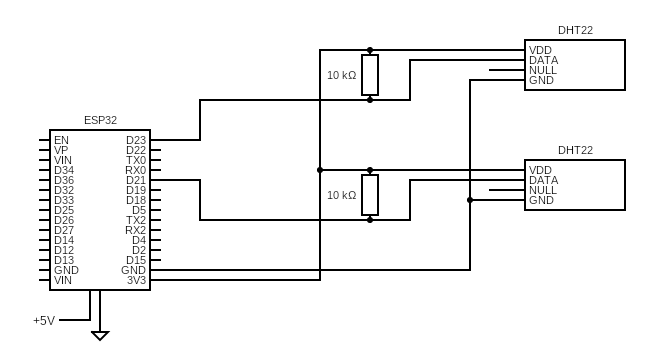
Pierwszym etapem tworzenia systemu było przygotowanie Raspberry Pi do współpracy z środowiskiem .NET Core. Z oficjalnej strony twórców systemu operacyjnego Raspbian pobrano plik obrazu w formacie .iso. Za pomocą programu Etcher przygotowano kartę pamięci z zainstalowanym system.

Po podłączeniu urządzeń peryferyjnych, karty pamięci oraz modułu GSM za pomocą złączy GPIO uruchomiono system, skonfigurowano połączenie do sieci LAN oraz Internet i ustawiono parametry w programie VNC Server do pracy z Raspberry Pi za pomocą komputera deweloperskiego. Praca została przeniesiona na wspomniany komputer.

Następnie z urządzenia usunięto większość niepotrzebnych aplikacji. System Raspbian posiada domyślnie wiele programów edukacyjnych, jak na przykład Wolfram Alpha. Kolejnym krokiem była aktualizacja systemu. W tym celu należało w terminalu wpisać dwa polecenia: sudo apt-get update które sprawdziło jakie elementy i pakiety posiadają dostępne nowe wersje oraz sudo apt-get upgrade, z kolei to polecenie wykonało właściwą aktualizację. Następnie za pomocą polecenia sudo apt-get install curl libunwind8 gettext apt-transport-https zainstalowano właściwe środowisko .NET Core. Aby przetestować poprawność konfiguracji utworzono czysty projekt ASP.NET Core, który został skompilowany poprzez polecenie konsoli dotnet publish -r linux-arm. Jest to wymagane, aby aplikacja mogła zostać uruchomiona w środowisku Linux i na urządzeniu z procesorem stworzonym w architekturze ARM. Następnie za pomocą funkcji File Transfer z programu VNC przeniesiono gotowy program testowy na Raspberry Pi i z powodzeniem uruchomiono za pomocą polecenia terminala dotnet test.dll.

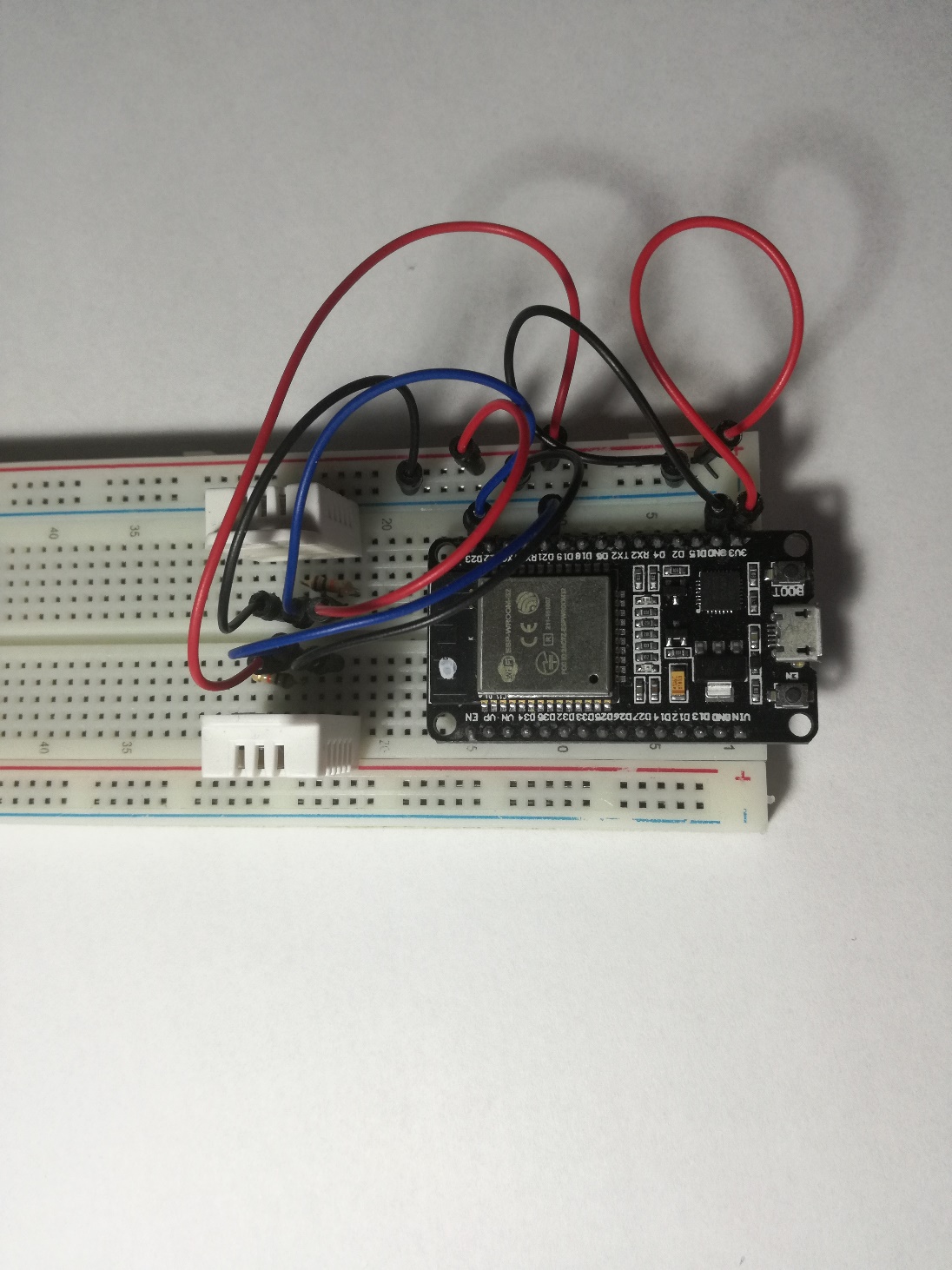
Kolejne etapy, czyli tworzenie oprogramowania serwera i czujników, przebiegały równolegle.

Pierwszym krokiem w projektowaniu czujnika było zapoznanie się z dokumentacją ESP32 oraz DHT22. Na ich podstawie został sporządzony schemat układu (Rys. 5-1). Widoczne na nim rezystory 10kΩ to tak zwane rezystory podciągające. Mają za zadanie wyeliminowanie stanów nieokreślonych, jakie mogłyby powstać na linii danych, gdy urządzenia nie komunikują się między sobą.



Rys. ‑ Schemat układu

Układ został złożony na płytce prototypowej (Rys. 5-2). Dzięki temu połączenia nie wymagały lutowania. W docelowej implementacji czujniki i układ ESP byłyby ze sobą połączone trwale oraz w sposób dostosowany do miejsca pracy.



Rys. 5‑2 Układ na płytce prototypowej

Przewody czerwone to przewody zasilania napięciem 3.3V, niebieskie to linia danych, natomiast czarne to masa.

Układ ESP32 spełnia trzy zadania: pobiera dane z czujników, stanowi punkt dostępu oraz jest hostem usługi http, dzięki której serwer może pobrać dane o temperaturze. Program dla układu ESP posiada dwie podstawowe metody: setup() i loop(). Metoda setup() jest wywoływana tylko raz podczas uruchomienia układu. Natomiast metoda loop(), jak sama nazwa wskazuje, jest wywoływana w pętli przez cały czas działania urządzenia. W metodzie setup() uruchamiane jest punkt dostępu poprzez polecenie WiFi.softAP(ssid, password), które jest częścią biblioteki WiFi, serwer http poprzez: server.on("/get", HTTP\_GET, [](AsyncWebServerRequest \*request){

request->send(200, "text/plain", data);

});   
server.begin();

Obiekt server jest typu AsyncWebServer, który pochodzi z biblioteki ESPAsyncWebServer.h.

Odczyt temperatur dokonywany jest pętli. Każda pętla jest opóźniona 2 sekundy, aby zapewnić poprawny odczyt, co wynika z dokumentacji czujników. Aby odczytać temperaturę z jednego czujnika wywoływana jest następująca sekwencja poleceń:

uint32\_t start = micros();

int chk = DHT.read22(DHT22\_PIN);

uint32\_t stop = micros();

float localTemp=DHT.temperature;

Dla pozostałych czujników należy powtórzyć powyższą sekwencję zmieniając parametr w metodzie DHT.read32() na tą zmienną przypisaną do konkretnego czujnika oraz zmienną, do której zapisuje się wynik DHT.temperature. Zmienne odpowiedzialne za konkretne czujniki są definiowane dyrektywy:

#define DHT22\_PIN 23

#define DHT22\_PIN1 21

Zapis DHT22\_PIN x oznacza, że linia danych z czujnika jest połączona z pinem x ESP32. Klasą odpowiedzialną za komunikację z czujnikami jest dht z biblioteki dht.h. Obiekt tej klasy jest przypisany do zmiennej globalnej DHT. Odczytane wyniki zapisywane są do zmiennej globalnej data w postaci łańcucha znaków uzyskiwany poprzez polecenie. data = String(localTemp)+","+String(localTemp1), a następnie jest wysyłana w momencie zapytania http przez serwer, który to może odczytać pod odpowiednim adresem, domyślnie http://192.168.4.1:80/get

Tworzenie serwera zarządzającego rozpoczęto od utworzenia nowego projektu ASP.NET Core w Visual Studio 2017 oraz założenie w folderze, w którym ów projekt został utworzony, repozytorium Git i połączenie go z repozytorium na portalu github.com. Plik z kodem źródłowym związane z oprogramowaniem czujników również zostały dodane do wymienionego repozytorium. Podczas tworzenia nowego projektu ASP.NET Core został wybrany docelowy framework .NET Core, 2.0 który na tamten moment był najnowszy, a także typ aplikacji Web Application (Model-View-Contoller). Spowodowało to automatyczne dodanie architektury MVC do projektu.

Najważniejszą klasą w całej aplikacji jest statyczna klasa Engine. Posiada ona wszystkie niezbędne metody, pola i odwołania do pozostałych klas. Podejście to może się kłócić z zasadą Single Responsibility [1], jednak sprawia. że pozostałe elementy mają dostęp do pozostałych funkcji z „jednego miejsca”. Konstruktor klasy Engine tworzy obiekty pozostałych klas odpowiedzialnych za konfigurację, czujniki, przechowywanie w pamięci odczytów czy powiadomienia. Podczas tworzenia obiektu klasy Engine odczytywane jest pięć pierwszych pomiarów z każdego czujnika, potrzebnych do prawidłowego wyświetlenia wykresów, w przypadku bezpośredniego uruchomienia podglądu temperatury. Również podczas tworzenia do obiektu o nazwie CollectedTimeStamps, który jest listą obiektów typu TimeStamp, dodawany jest obiekt zawierający informacje o starcie aplikacji. Tworzenie obiektów czujników jest wykonywane przy użyciu statycznej metody GetSensors() z klasy EspSensorsInitializer. Zwraca ona obiekt typu List<Esp32Sensor>.

Klasa Engine posiada metody obsługujące najważniejsze funkcje. GetRecentReadings() zwraca listę ostatnich pięciu pomiarów z każdego czujnika, które przechowywane są w polu RecentReadings w postaci jednego łańcucha znaków. Do tego celu wywołuje z klasy statycznej ListExtenstions metodę GetDoubleValuesInOneStringFromTwoDimList(). Metoda ta jest używana do wyświetlania wykresów i aktualnej temperatury. CollectData() co zadany czas zapisuje do zmiennej CollectedTimeStamps nowy TimeStamp z aktualną temperaturą. GenerateReport() tworzy raport dobowy o określonej godzinie na podstawie zebranych danych w CollectedTimeStamps, zapisuje go do pliku .txt oraz zeruje listę CollectedTimeStamps, co ma na celu oszczędzenie pamięci RAM urządzenia. Nazwa pliku zawiera datę, godzinę utworzenia oraz osiem losowych znaków na wypadek generowania raportu w tym samym momencie. Najważniejszą metodą jest Monitoring(). Jej cykliczne wywoływanie powołuje odczytywanie wyników z czujników, obliczanie średniej temperatury i porównanie jej z zadanymi zakresami dopuszczalnej temperatury – ostrzegawczym i alarmowym. Jeżeli średnia temperatura przekroczy zakres ostrzegawczy, lecz nie alarmowy, zostanie wysłany tylko komunikat do podglądu temperatury. Natomiast jeśli zostanie przekroczony zakres alarmowy to oprócz wysłania komunikatu do podglądu, zostanie wysłane powiadomienie SMS o niebezpiecznej temperaturze. Metody CheckWarningRange() i CheckAlarmRange() sprawdzają podaną temperaturę, czy mieści się w zakresach.

Klasa Config odpowiada za ustawienia zakresów dopuszczalnych temperatur, listę numerów telefonów, do których należy wysłać powiadomienia oraz częstotliwości z jakimi należy odświeżać pomiary, zapisywać pomiar czy godzinę o której tworzyć raport dobowy. Zakres jest prezentowany za pomocą czterech publicznych właściwości typu double: LowerAlarmBorder, LowerWarnBorder, UpperWarnBorder oraz UpperAlarmBorder. Numery telefonów są zapisane w liście elementów typu string. Instancja klasy Config jest tworzona za pomocą statycznej metody z klasy ConfigInitializer o nazwie GetConfig(), która to deserializuje plik config.json. Plik ten znajduje się w katalogu /home/pi. Aby zmienić konfigurację należy ręcznie edytować plik poprzez fizyczne podłączenie się do Raspberry Pi. Ma to na celu zwiększenie bezpieczeństwa całego systemu.

Klasą czujników jest Esp32Sensor. Posiada ona prywatne pola \_espIpAdress, które przechowuje adres IP pod jakim jest dostępny czujnik; \_sensorID które to odpowiada za numer czujnika dostępnego pod danym adresem; readings i errorReadings odpowiadają za kontrolę nad błędami pomiarów. Kontrola błędów polega na sprawdzeniu proporcji między błędnymi pomiarami do wszystkich pomiarów. Błędny pomiar objawia się tym że wynik jest równy -999 stopni Celsjusza. Jeśli współczynnik błędnych pomiarów przekroczy 10% zostanie wysłane powiadomienie o uszkodzeniu czujnika. Aby zabezpieczyć się przed przekroczeniem zakresu zmiennej readings jest ona, jak również zmienna errorReadings dzielona całkowicie przez dziesięć. Zmienne te są typu short, której maksymalna wartość wynosi 32,767. Odczyt temperatury jest realizowany przez metodę GetTemperature() i jest przeprowadzany etapami. Pierwszym jest przygotowanie poprawnego linku w następujący sposób: string uri = "http://" + \_espIpAddress + ":80/get"; i przygotowanie klienta http. Następnie wywoływanie zapytania http i sprawdzenie wyniku. Potem zawartość otrzymana poprzez zapytanie jest dzielona na tablicę string i wybierany ten element odpowiadający sensorID. Ta wartość jest konwertowana na typ double i zwracana z metody. Każdy etap jest kontrolowany i w przypadku wystąpienia problemów wysyłany jest alert o uszkodzeniu urządzenia. Jest to wykorzystywane zwłaszcza przy wykrywaniu awarii zasilania. Czujniki są inicjalizowane za pomocą klasy EspSensorsInitializer.

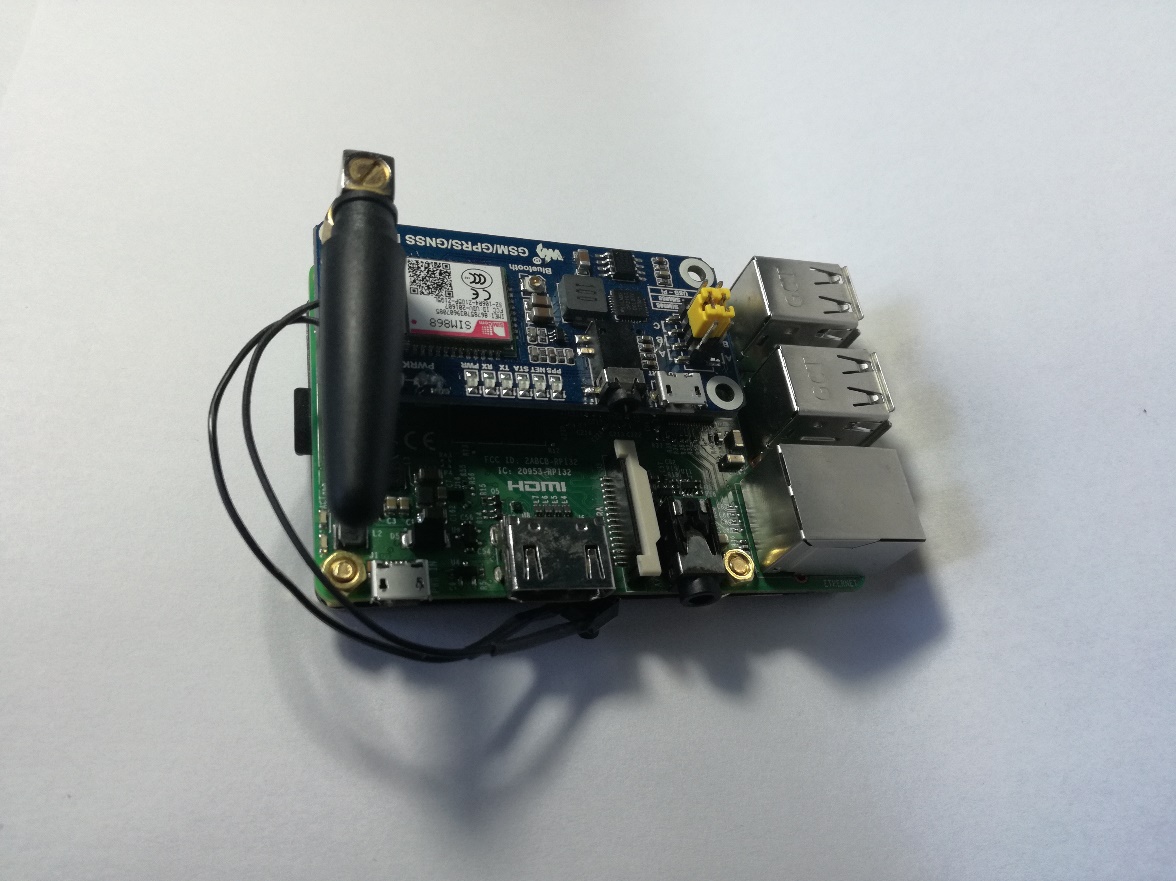
W związku ze specyfiką projektu, na komputerze deweloperskim nie było możliwości testowania czujników oraz wysyłania powiadomień SMS. Dlatego zostały utworzone klasy ProxySensor która implementuje interfejs ISensory oraz ProxyAlerter implementując IAlerter. Umożliwiło to testowanie oprogramowania z poziomu komputera deweloperskiego, bez konieczności przenoszenia gotowego programu za każdym razem na Raspberry Pi. W kodzie wystarczyło podmieniać konstruktory aby dostosować dany wynik kompilacji do konkretnego środowiska. Interfejs ISensory posiada deklaracje trzech metod, w tym najważniejsza jest GetTemperature(). Natomiast IAlerter deklaruje SendAlert() i SendWarning(). Po interfejsie ISensory dziedziczą, poza ProxySensor, klasy GpioSesnor (klasa wykorzystywana we wczesnym etapie prac) oraz Esp32Sensor. Interfejs IAlerter implementują klasy GsmAlerter i ViewAlerter.

Podczas tworzenia obietku GsmAlerter do jego pola PhoneNumberList poprzez referencje jest dołączana lista telefonów do których należy wysyłać powiadomienia oraz tworzony jest obiekt klasy SerialPortCommunication. Klasa ta jest w odpowiedzialna za komunikację z portem szeregowym Raspberry Pi, do którego z kolei podłączony jest moduł GSM. GsmAlerter posiada jedną metodę SendAlert() przyjmujący za parametr typ alertu. Do wyboru są na tym etapie dwa komunikaty: o przekroczeniu zakresu dopuszczalnej temperatury i o problemie z czujnikiem. Dla każdego numeru wywoływana jest metoda z klasy SerialPortCommunication SendSMS() przyjmująca właśnie numer telefonu i treść wiadomości SMS. W metodzie tej na samym początku otwierane jest połączenie z portem szeregowym złączy GPIO do którego podłączony jest modem GSM, który w systemie Raspbian jest dostępny pod /dev/ttyS0. Następnie do urządzenia przesyłane są komendy AT [18]. Komendy rozpoczynają się zazwyczaj od ciągu znaków AT po których następuje polecenie. Do wysłania wiadomości SMS należy wprowadzić następujący ciąg komend:

1. AT+CMGF=1 – ustalenie trybu na wysyłanie wiadomości SMS.
2. AT+CMGS=numer\_telefonu – wprowadzenie numeru odbiorcy.
3. wiadomość \x1A – treść wiadomości zakończona znakiem \x1A (odpowiada Ctrl-Z).

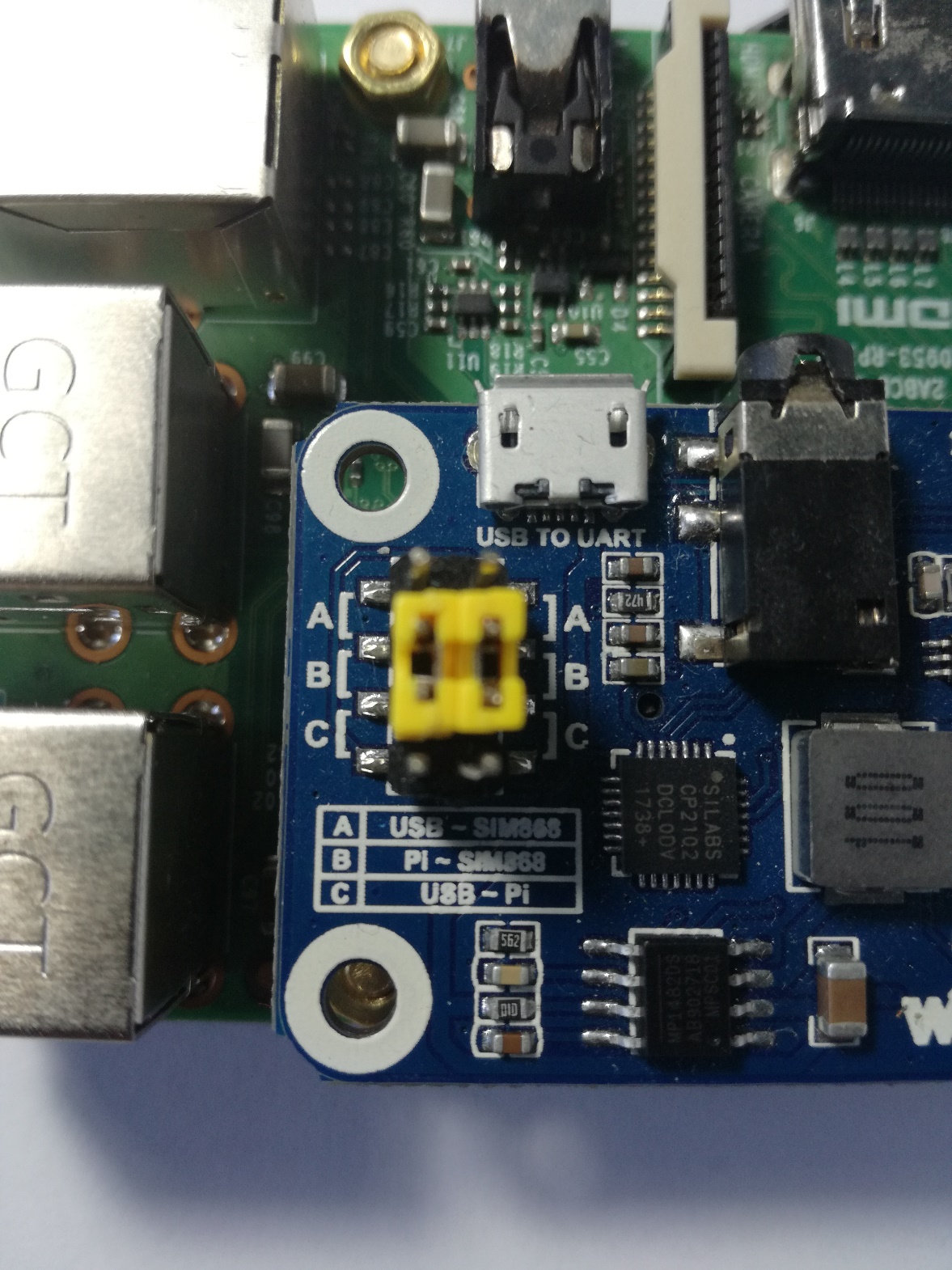
Między każdą komendą następuje 1 sekundowa przerwa, po to aby dać czas urządzeniu na przetworzenie zapytania. Na samym końcu połączenie z portem szeregowym jest zamykane.

Moduł GSM jest podłączony do złączy GPIO Raspberry Pi (Rys. 5-3).



Rys. ‑ Raspberry Pi z podłączonym modułem GSM

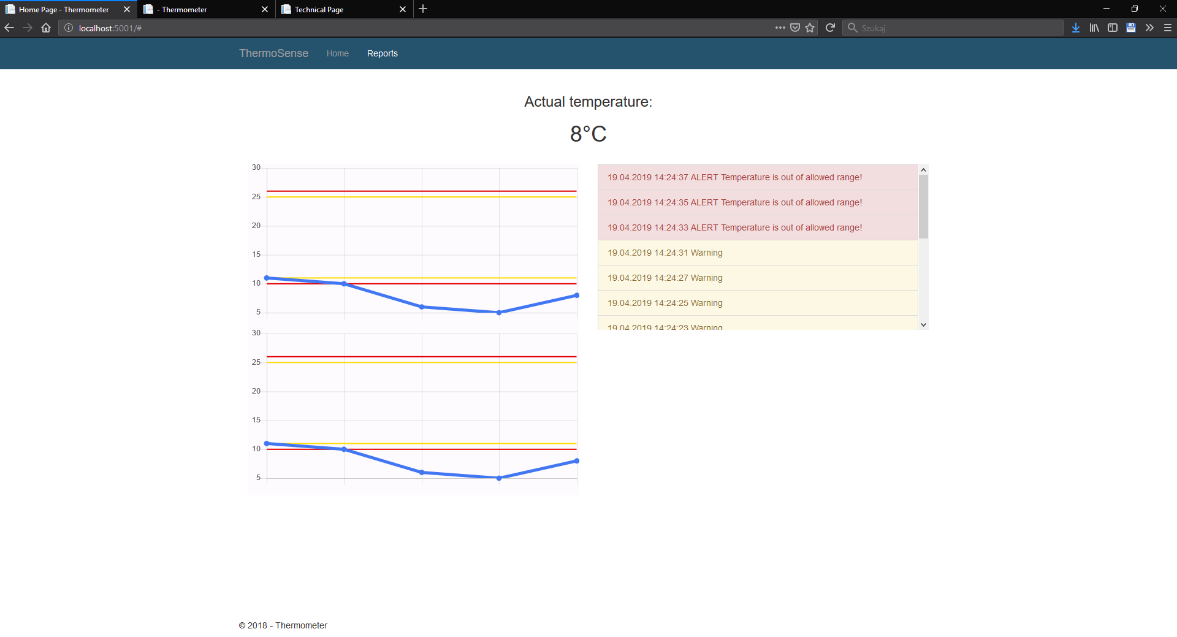
Aby moduł poprawnie komunikował się poprzez port szeregowy należy ustawić poprawnie żółte zworki na układzie w pozycji B (Rys. 5-4). Oprócz tego konieczne jest podłączenie aktywnej karty SIM. Na potrzeby projektu zastosowano kartę sieci Play, opłaconą w modelu prepaid. Slot na kartę sim znajduje się pod spodem modemu, obok miejsca na baterie podtrzymującą zegar czasu rzeczywistego. Podczas uruchamiania całego zestawu należy na 5 sekund przycisk na module aby połączył się do sieci GSM. Poprawne działanie jest sygnalizowane ciągłym świeceniem diod PWR i STA oraz miganiem diody NET.



Rys. ‑ Poprawna konfiguracja zworek

Klasą która wywołuje bezpośrednio lub pośrednio jest UpdateHub. Dziedziczy ona po klasie Hub, która jest częścią biblioteki SignalR. Metody tej klasy są asynchroniczne i wywołują właściwe metody z klasy Engine. Komunikacja w stronę klienta odbywa się poprzez wywołanie metody SendAsync(), której pierwszy parametr typu string to nazwa metody, co pozwala na identyfikację informacji pochodzącej ze źródła. Pozostałe parametry są opcjonalne i są typu object. Klient potrzebuje własnych metod do obsługi informacji otrzymywanych z serwera. Można zarządzać na różne sposoby, kto powinien otrzymać dane z serwera. Istnieją trzy podstawowe grupy odbiorców: do wszystkich połączonych, tylko do klienta, który wywołał daną metodę i do wszystkich poza wywołującym. Podczas tworzenia obiektu klasy UpdateHub referencja do niej jest przesyłana do klasy Engine, dzięki czemu metody tej klasy są dostępne zarówno po stronie klientów jak i serwera.

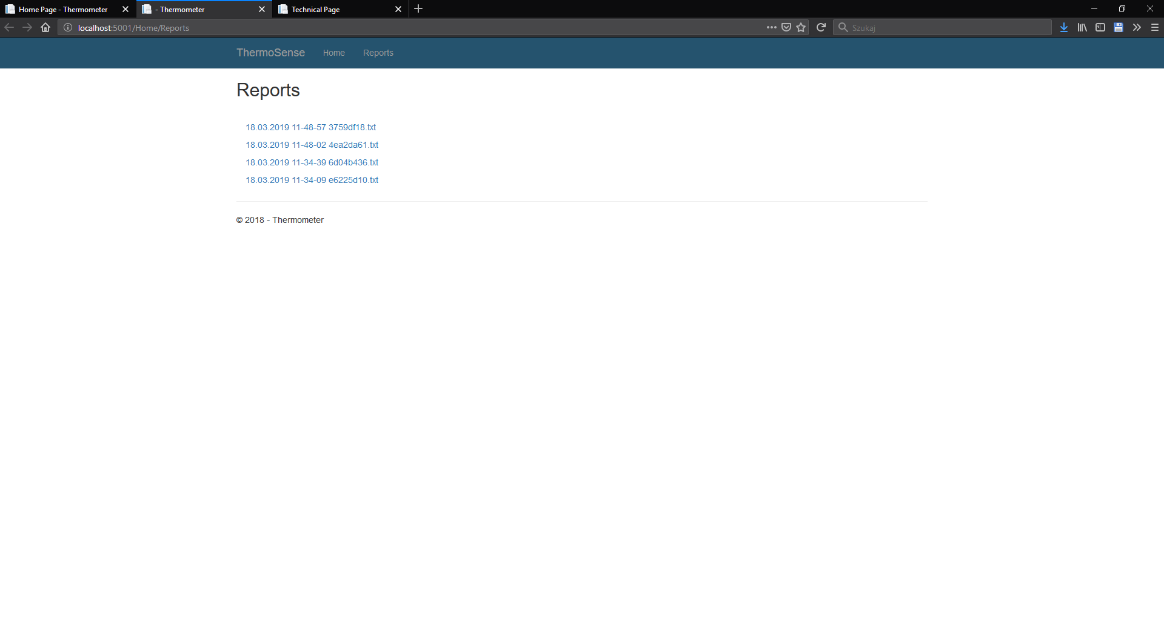
Prezentacja aktualnej temperatury jest tworzona przy użyciu silnika widoków i języka Razor dostępnych w ASP.NET. Użytkownik ma dostęp do podstawowego widoku po wpisaniu adresu IP serwera przez port 5000 np. http://192.168.4.2:5000. Po wpisaniu użytkownik jest wstanie zobaczyć aktualną średnią temperaturę, wykresy temperatur z każdego czujnika oraz ewentualne powiadomienia o przekroczeniu zakresów (Rys. 5-5).



Rys. ‑ Widok Home

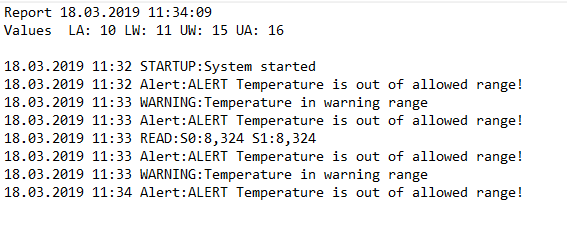
Powiadomienia po prawej stronie są wywoływane po stronie serwera i przy pomocy SignalR są przesyłane do widoku natychmiast. Funkcje JavaScript identyfikują rodzaj powiadomienia i w odpowiedni sposób go wyświetlają. Wykresy są tworzone w podobny sposób przy pomocy chart.js, a częstotliwość, z jaką są odświeżane jest dostarczona poprzez obiekt ViewBag. Jest to obiekt typu dynamic, co sprawia że do niego można dołączyć dowolną ilość dowolnych obiektów. Używając typu dynamicznego trzeba mieć na uwadze, że aby odwołać się do danego obiektu, należy wpisać dokładną nazwę.

Kolejnym widokiem Dostępnym dla użytkownika jest Report. Jest to prosta strona umożliwiająca pobranie wygenerowanych raportów dobowych (Rys. 5-6).



Rys. ‑ Widok Reports

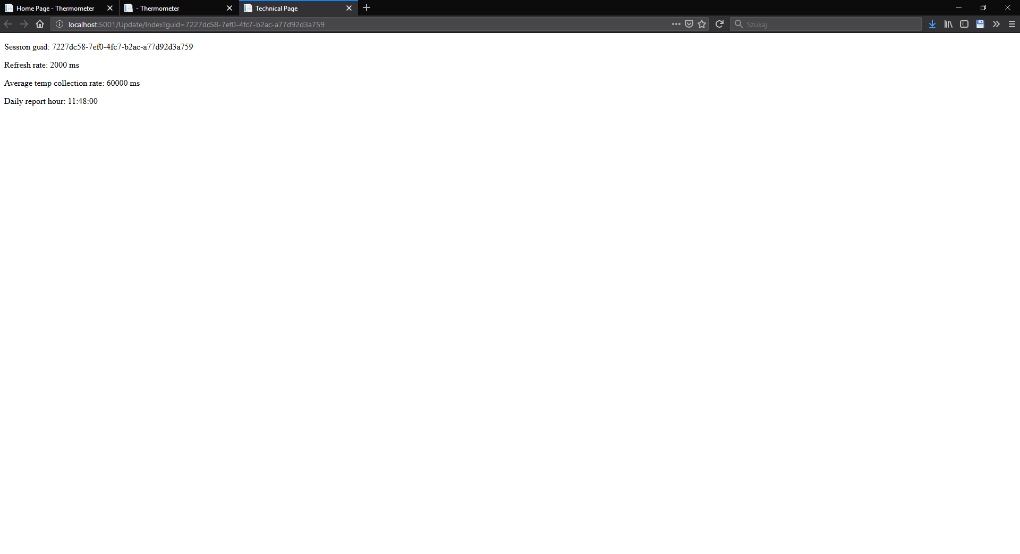
Raporty są tworzone o ustalonej godzinie w oparciu o CollectedTimeStamps. Klasa TimeStamp posiada trzy pola: Time odpowiada za czas wykonania, Type które określa typ komunikatu i Value czyli właściwa treść komunikatu. Ponadto klasa nadpisuje metodę ToString() po to, aby uzyskać gotowy wpis do raportu. Poniżej przykład wygenerowanego raportu:



Rys. ‑ Przykład raportu

Na samym początku raportu jest jego data i godzina utworzenia. poniżej niego znajduje się informacja na temat zakresów temperatur. Obecnie możliwe są cztery typy wpisu: Startup, Warning, Alert i Read. Nazwa raportu składa się z daty i godziny jego utworzenia a także fragmentu losowego GUIDu, co ma na za zadanie zabezpieczać przed problemami, związanymi z nazwami plików. Raporty są posortowane od najnowszego. Aby pobrać konkretny raport wystarczy nacisną jego nazwę.

Najważniejszym widokiem jest jednak Technical Page (Rys. 5-8). Ma za zadanie cykliczne wywoływanie najważniejszych funkcji (monitorowanie, zapis znacznika czasowego, generowanie raportów) za pomocą SignalR i JavaScript. Jest on uruchamiany zaraz przy starcie całego systemu.



Rys. ‑ Strona Techniczna

Dla prawidłowego funkcjonowania systemu, strona techniczna musi być tylko raz uruchomiona. Aby to zapewnić podczas uruchomienia aplikacji ASP.NET tworzony jest plik zawierający losowo wygenerowany GUID. Następnie ten GUID jest odczytywany przez skrypt uruchamiający całość systemu i przekazywany jako parametr w adresie strony technicznej. W momencie zapytania parametr ten jest porównywany z utworzonym GUIDem. Jeśli się różni nastąpi przekierowanie na stronę z informacją o błędzie. Jeśli są takie same strona uruchomi się normalnie. Uzyskanie tego GUIDa jest możliwe jedynie poprzez fizyczne podłączenie się do urządzenia.

Skrypt uruchamiający został napisany w języku skryptowym bash, dostępnym w systemach operacyjnych z rodziny Linux. Jest uruchamiany wraz ze startem systemu Raspbian. Wykonuje on cztery polecenia:

1. Uruchamia aplikację.
2. Czeka jedną minutę. Daje to czas na uruchomienie aplikacji.
3. Odczytuje GUID z pliku.
4. Uruchamia przeglądarkę i wchodzi na stronę internetową localhost:5001/Update/Index?guid=”odczytany\_guid”.

# Podsumowanie

# Spis ilustracji

[Rys. 5‑1 Schemat układu 13](#_Toc6582157)

[Rys. 5‑2 Układ na płytce prototypowej 13](#_Toc6582158)

[Rys. 5‑3 Raspberry Pi z podłączonym modułem GSM 18](#_Toc6582159)

[Rys. 5‑4 Poprawna konfiguracja zworek 19](#_Toc6582160)

[Rys. 5‑5 Widok Home 20](#_Toc6582161)

[Rys. 5‑6 Widok Reports 21](#_Toc6582162)

[Rys. 5‑7 Przykład raportu 21](#_Toc6582163)

[Rys. 5‑8 Strona Techniczna 22](#_Toc6582164)

# Literatura

1. Martin R.: Czysty kod. Podręcznik dobrego programisty. Wydawnictwo Helion, Gliwice, 2014.
2. Jacek Wysocki, Hanna Czajka: Szczepienia w pytaniach i odpowiedziach. Wydanie V, Wydawnictwo „Help-Med” s.c., Kraków 2014.
3. Ustawa z dnia 5 grudnia 2008 r. o zapobieganiu oraz zwalczaniu zakażeń i chorób zakaźnych u ludzi, (Dz.U. 2008 nr 234 poz. 1570).
4. Rozporządzenie Ministra Zdrowia z dnia 18 sierpnia 2011 r. w sprawie obowiązkowych szczepień ochronnych, (Dz.U. 2011 nr 182 poz. 1086)
5. https://docs.microsoft.com/en-us/dotnet/core/ z dnia 04.01.2019r.
6. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1069029/ z dnia 23.01.2019r.
7. https://www.raspberrypi.org/documentation/ z dnia 04.11.2018r
8. Espressif Systems: ESP32 Series Datasheet.
9. https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf z dnia 23.01.2019r.
10. https://www.waveshare.com/wiki/GSM/GPRS/GNSS\_HAT z dnia 23.01.2019r
11. https://docs.microsoft.com/pl-pl/aspnet/core/ z dnia 23.01.2019r.
12. http://tobiasahlin.com/blog/introduction-to-chartjs/ z dnia 23.02.2019r.
13. https://www.raspbian.org/ z dnia 23.02.2019r.
14. https://getbootstrap.com/docs/4.3/about/overview/ z dnia 03.03.2019r.
15. https://docs.microsoft.com/pl-pl/visualstudio/ z dnia 07.03.2019r.
16. https://www.arduino.cc/en/Guide/Environment z dnia 07.03.2019r.
17. https://git-scm.com/ z dnia 13.03.2019r.
18. https://www.waveshare.com/w/upload/2/20/SIM800\_Series\_AT\_Command\_Manual\_V1.09.pdf z dnia 02.03.2019r.