

POLITECHNIKA ŚLĄSKA W GLIWICACH

WYDZIAŁ INŻYNIERII BIOMEDYCZNEJ



Telemedycyna – Projekt

Prototyp urządzenia służącego do rejestracji i transmisji sygnału EKG

Kamil Suchanek
IiAM 2 sem. 6

Kierujący pracą
Dr inż. Adam Pawlak

Bytom – 24.09.2018

Spis treści

1.	<i>Cel projektu</i>	3
2.	<i>Założenia</i>	3
3.	<i>Teoria</i>	3
3.1.	<i>Zarys elektrokardiografii</i>	3
3.2.	<i>Arduino</i>	4
4.	<i>Realizacja projektu</i>	4
4.1.	<i>Modyfikacje</i>	4
4.2.	<i>Przesyłanie danych na stronę internetową</i>	5
4.2.1.	<i>Aplikacja ZapisEKG – specyfikacja wewnętrzna</i>	8
4.2.1.1.	<i>Kontrolki</i>	8
4.2.1.2.	<i>Obsługa kontrolek</i>	8
4.2.1.3.	<i>Przesyłanie danych</i>	10
4.2.1.4.	<i>Informacje o błędach</i>	12
4.2.2.	<i>Aplikacja ZapisEKG – specyfikacja zewnętrzna</i>	13
4.2.2.1.	<i>Instalacja Heroku CLI i psql</i>	13
4.2.2.2.	<i>Start Aplikacji</i>	13
4.2.2.3.	<i>Wybór folderu i przesyłanie danych</i>	14
4.3.	<i>Rejestracja i zapis rekordów</i>	16
5.	<i>Aneks do sprawozdania</i>	17
6.	<i>Podsumowanie</i>	18
7.	<i>Spis części</i>	19
8.	<i>Źródła</i>	19

1. Cel projektu

Złożenie i zaprogramowanie prototypu urządzenia zdolnego do rejestracji sygnału EKG i przesyłania go na stronę internetową.

2. Założenia

Urządzenie ma rejestrować sygnał EKG przez całą dobę. Przesyłanie go ma następować przynajmniej raz dziennie gdy wracamy do domu i urządzenie ma dostęp do znajomej sieci WiFi. W czasie gdy sieć nie jest dostępna urządzenie zapisuje sygnał na karcie SD. Sygnał zapisywany na karcie znajduje się w plikach tekstowych po 5 minut zapisu. Sygnał rejestrowany jest z częstotliwością 200Hz. Program główny tworzony jest w środowisku Arduino. Zastosowanie modułu zawierającego układ AD8232 służącego do rejestrowania sygnału EKG, oraz modułu ESP-X 8266 służącego do komunikacji poprzez sieć WiFi. Pracą obu modułów ma zarządzać płytka Arduino UNO.

3. Teoria

3.1. Zarys elektrokardiografii

Podstawowym urządzeniem służącym do rejestracji napięć elektrycznych tkanki żywej jest galwanometr. Obwód elektryczny składający się z dwóch zakończeń galwanometru (ujemnej elektrody i dodatniej) oraz dwóch punktów pola elektrycznego nazywa się odprowadzeniem. Rozróżnia się dwa rodzaje odprowadzeń: jednobiegunowe, gdzie jedną z elektrod umieszcza się w miejscu pomiaru potencjału a drugą w miejscu oddalonym od owego, oraz dwubiegunowe, gdzie obie elektrody umieszcza się w miejscach pomiaru potencjału. Pomiar uzyskane za pomocą odprowadzeń jednobiegunowych dostarczają bezwzględne wartości potencjałów. Dopiero zastosowanie odprowadzeń dwubiegunowych umożliwia ocenę zmiennego pola elektrycznego na powierzchni komórki.

Siła elektromotoryczna serca jest wartością wektorową. Rejestracji podlega wypadkowa wartość siły elektromotorycznej serca. Siła ta zmienia się w czasie w sposób cykliczny i odpowiada kolejnym ewolucjom cyklu sercowego. Wielkość mierzona na powierzchni ciała osiąga wartość do około 2 mV.

Krzywa elektrokardiograficzna jest graficznym przedstawieniem elektrycznej czynności serca. Składają się na nią wychylenia wzdłuż linii izoelektrycznej. Szczegóły morfologii tych wychyleń i odległości pomiędzy nimi przekładają się na wartość diagnostyczną. Również ważnym aspektem podlegającym ocenie typowego badania EKG jest częstość i cykliczność wychyleń zapisu.

3.2. Arduino

Arduino jest platformą programistyczną przeznaczoną dla systemów wbudowanych opartą na projekcie Open Hardware przeznaczonym dla mikrokontrolerów montowanych w pojedynczym obwodzie drukowanym, wraz z wbudowaną obsługą układów wejścia/wyjścia oraz standaryzowanym językiem programowania. Arduino stanowią zatem gotowe rozwiązania z pewną ograniczoną swobodą w zastosowaniu i rozwoju tworzonego projektu.

Układy Arduino bazują na mikrokontrolerach AVR. Programowanie odbywa się zazwyczaj za pomocą środowiska Arduino IDE, a język programowania bazuje na języku C i C++. Na projekt Arduino składają się liczne moduły zewnętrzne, kompatybilne z popularnymi płytkami.

Podstawowy program Arduino składa się z dwóch funkcji:

- void setup() – funkcja wykonywana raz,
- void loop() – funkcja wykonywana w trybie ciągłym po wykonaniu funkcji setup().

Programowanie układu Arduino przypomina pisanie prostych programów obsługi systemów wbudowanych w języku C, platforma ta zbliża jednak pracę z kontrolerem do programowania wysokopoziomowego.

4. Realizacja projektu

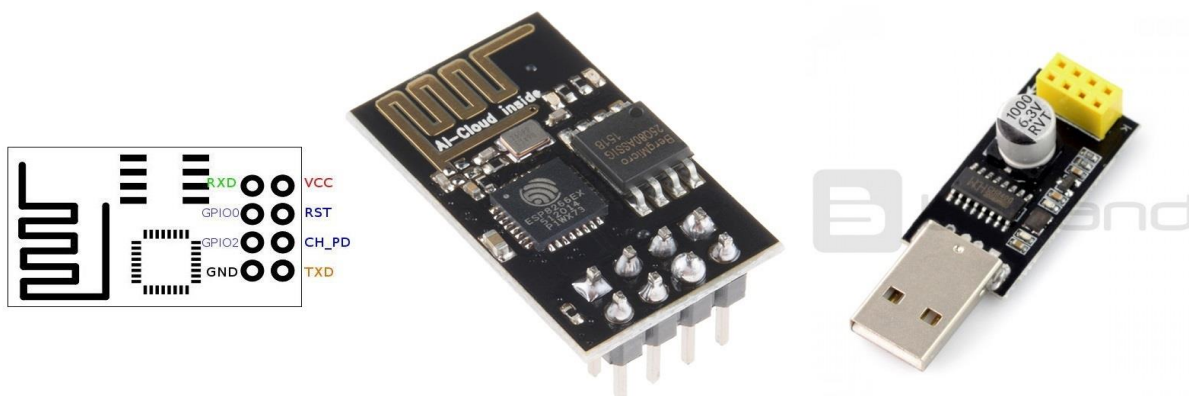
4.1. Modyfikacje

Wbrew założeniom, w celu realizacji projektu dokonano następujących zmian:

1. Dodanie aplikacji okienkowej – realizacja przesyłania zapisów EKG do bazy danych, ponieważ nie udało się komunikacja z bazą danych PostgreSQL należącą do serwisu Heroku,
2. Zmiana podstawowej płytki noszącej oprogramowanie na NodeMCU v3 – układ Arduino okazał się zbędny w zarządzaniu działaniem modułów WiFi i EKG,
3. Zmiana długości poszczególnych rekordów z pięciu minut do minuty – pojedynczy rekord w kolumnie typu numeric[] bazy danych PostgreSQL nie mógł przyjąć zbyt długich tablic wartości,

4.2. Przesyłanie danych na stronę internetową

Płytki ESP-X 8266 posiadają własny kontroler zdolny do zarządzania czujnikiem i modulem czytnika kart SD. Zastosowano moduły ESP-01 (Rys. 1.), ESP-12E oraz NodeMCU (ESP-12).



Rys. 1. Płytki ESP-01 wraz z rozkładem pinów oraz adapter USB – UART.

W przypadku pierwszego ilość wejść/wyjść była niewystarczająca i układ służył do testowania funkcji związanych z połączeniem z siecią WiFi oraz wprowadzaniu poleceń SQL za pomocą kodu zawartego w pliku test_ESP1 (Rys. 2-3.). Zostało tam użyte polecenie programu psql, po którym należy podać host, port, nazwę bazy, nazwę użytkownika i hasło (Rys. 3.).

```
test_ESP1 $
#include <ESP8266WiFi.h>
const char* ssid = "multimedia_LUMKA1";
const char* password = "ATKAMIL1606";
const char* host = "ec2-54-221-237-246.compute-1.amazonaws.com";
const char* database = "d6vbf3bajggsh4";
const char* user = "jdbigwjpuutcu";
const char* privateKey = "8e684ca67665828a0bb7567d03d1faa3bb0875749d88f65af3db0fcb0cfd924d";
String line;

void setup() {
  delay(1000);
  /* Włączenie komunikacji portem szeregowym o prędkości 115200 baud */
  Serial.begin(115200); delay(10); Serial.println('\n');
  WiFi.mode(WIFI_STA); /* tryb działania modułu WiFi jako stacja */
  /* zainicjowanie połączenia z siecią WiFi */
  WiFi.begin(ssid, password); Serial.print("Connecting to "); Serial.print(ssid); Serial.println(" ...")

  int i = 0;
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000); /* Sprawdzanie połączenia i oczekiwanie na owe */
    Serial.print(++i); Serial.print(' ');
  }
  Serial.println("Connection established!");
  Serial.print("IP address:\t");
  Serial.println(WiFi.localIP());
  /* W ten sposób można śledzić działanie programu, wyrzucając różne informacje na port szeregowy
  przy pomocy Serial.print/println() */
  WiFi.printDiag(Serial);
}
```

Rys. 2. Stałe globalne oraz funkcja setup().

```

void loop() {
    delay(5000);

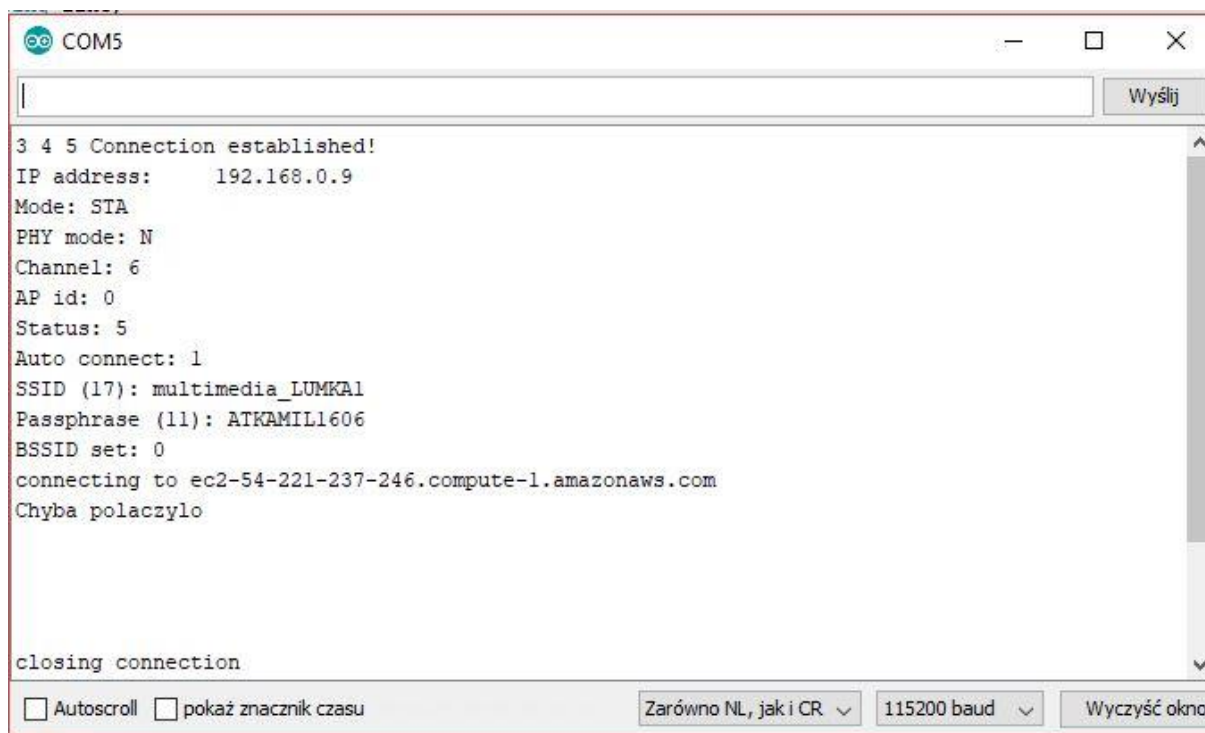
    Serial.print("connecting to "); Serial.println(host);

    WiFiClient client;
    const int httpPort = 5432;
    if (client.connect(host, httpPort)) {/* połączenie z portem bazy danych postgreSQL */
        Serial.println("Chyba polaczylo");
    }
    /* polecenie programu psql -[OPCJE] */
    client.println("psql -U jdbigwjpuutcuu --password 8e684ca67665828a0bb7567d03dlfaa3bb08
    delay(1000);
    line = client.readStringUntil('\n');/* odczytanie odpowiedzi serwera */
    Serial.println(line);/* wrzucenie jej do portu szeregowego */
    delay(1000);
    /* podanie hasła */
    client.println("8e684ca67665828a0bb7567d03dlfaa3bb0875749d88f65af3db0fcb0cfd924d");
    delay(1000);
    line = client.readStringUntil('\n');
    Serial.println(line);
    delay(1000);
    /* przykładowe polecenie SQL zmieniające rolę użytkownika na stronie internetowej */
    client.println("UPDATE users SET role=1 WHERE first_name='Kamil'");
    delay(1000);
    line = client.readStringUntil('\n');
    Serial.println(line);
    |
    while (client.connected())
    {
        if (client.available())
        {
            line = client.readStringUntil('\n');
            Serial.println(line);
        }
    }
    Serial.println();
    Serial.println("closing connection");
}

```

Rys. 3. Funkcja loop() oraz próby komunikacji z bazą danych.

Wynik działania programu można obserwować na monitorze portu szeregowego środowiska Arduino (Rys. 4.). Do zaprogramowania układu ESP-01 oraz obserwacji jego działania użyto adaptera USB-UART (Rys. 1.). W celu zaprogramowania układu ESP, należało zewrzeć ze sobą piny GPIO0 i GND. Zanim wprowadzono program, bez zwierania pinów układ należało sprawdzić wprowadzając do portu polecenie „AT”, odpowiedź „OK” oznacza sprawność modułu.



Rys. 4. Monitor portu szeregowego, Wynik testu.

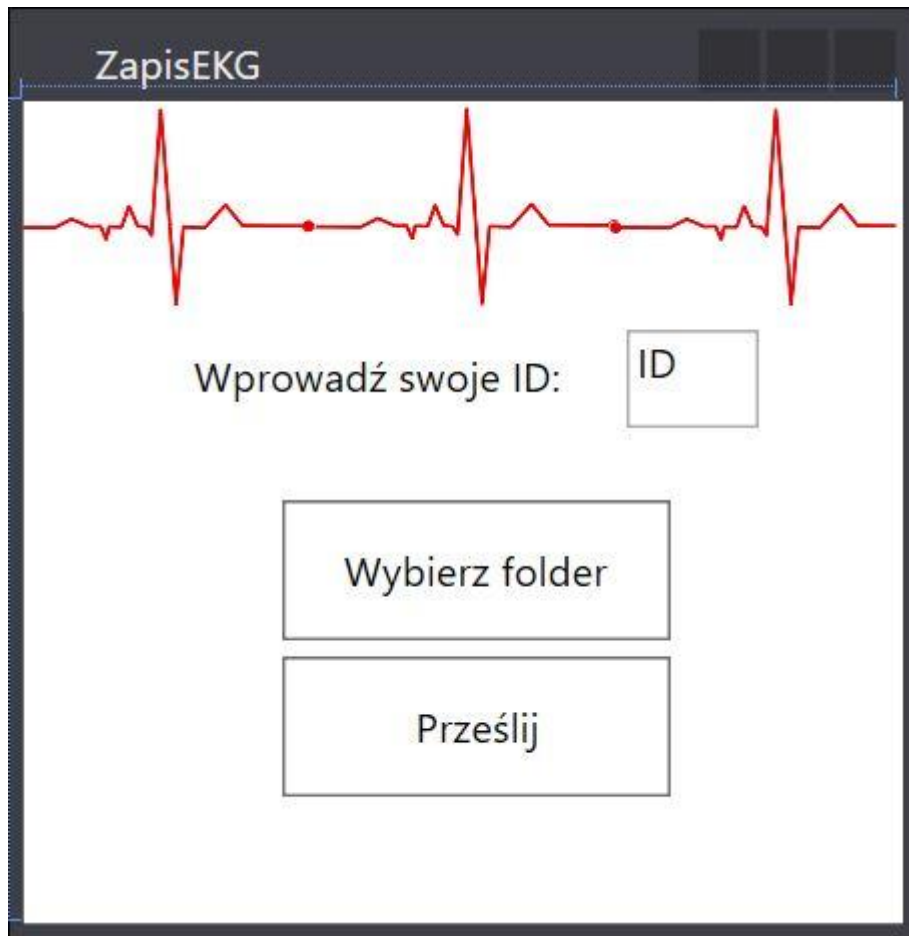
Osiągnięto połączenie z siecią WiFi. Połączenie z portem 5432 bazy danych postgresQL prawdopodobnie również, jednak wykonywane zapytania nie odnosiły skutku, albo samo zalogowanie się do bazy nie powiodło się. Zastosowane różne kombinacje kodu. Ostatecznie została napisana aplikacja okienkowa WPF. Rozwiązanie problemu może leżeć w zmianie źródła bazy danych, serwis Heroku wymaga bezwzględnie klucza SSL w trakcie komunikacji. Wykonywane zapytania były poprawne, sprawdzane wcześniej w konsoli na tej samej bazie danych. Trudno stwierdzić czym konkretne było źródło niepowodzenia połączenia ze względu na ograniczoną ilość informacji uzyskanych w czasie testów. Środowisko Arduino udostępnia opcję dodania do programu wgrywanego fragmentów odpowiedzialnych za debbuging i wyświetlanie logów w porcie szeregowym.

Wybrana Płytką NodeMCU opiera się o układ ESP-12 i to na niej została zrealizowana rejestracja EKG i jego zapis na karcie SD. Płytką ta została wybrana zamiast prostego Arduino UNO pomimo braku elementu przesyłania danych prosto do bazy w wynikowym prototypie ze względu na inne dostępne rozwiązania możliwe do zastosowania dla tego urządzenia w przyszłości. Jednak w tym projekcie należy dostarczyć danych konkretniej stronie internetowej publikowanej za pośrednictwem serwisu Heroku.

Kod programu (Rys. 2-3.) jest analogiczny dla każdej wersji płytki ESP oraz układu NodeMCU. Zadanie transmisji sygnału bezpośrednio z prototypu zostało przeniesione na oddzielną aplikację okienkową działającą w systemie Windows, napisaną jako WPF technologii .NET. Owa aplikacja przyjmuje folder z danymi uzyskiwanymi przez prototyp. Kartę SD należy przełożyć z urządzenia do laptopa (większość posiada teraz czytniki) oraz włączyć aplikację, wybierając folder karty.

4.2.1. Aplikacja ZapisEKG – specyfikacja wewnętrzna

4.2.1.1. Kontrolki



Rys. 5. Widok kontrolek aplikacji.

Aplikacja posiada jedno pole do wypełnienia wartością całkowitoliczbową oraz dwa przyciski. Przycisk *Wybierz Folder* służy do wybrania folderu z zapisami EKG. Przycisk *Prześlij* powoduje włączenie konsoli i wykorzystanie programów SQL oraz Heroku CLI do przesłania danych do bazy. Na przyciskach znajduje się ukryty obraz oczekiwania, który zostaje odkryty na czas wykonywania zadań przez program po kliknięciu każdego z przycisków.

4.2.1.2. Obsługa kontrolek

```
string HEROKU_CLI = "heroku pg:psql postgresql-clean-81393 --app ekg-app";  
string sciezka;  
int Id;
```

Rys. 6. Zmienne globalne programu.

Zmienna *HEROKU_CLI* zawiera polecenie programu Heroku CLI oraz programu psql. Podaje ono informacje o konkretnej bazie danych – postgresql-clean-81393 oraz o aplikacji której podlega – ekg-app. Serwis Heroku udostępnia szczegółową dokumentację swoich usług

i dostępu do nich z zewnątrz. Zmienna *sciezka* przechowuje ścieżkę do folderu karty SD, którą należy wybrać po wciśnięciu przycisku *Wybierz Folder*. Zmienna *Id* przechowuje ID podane przez użytkownika do pola tekstowego.

```
private void Wybierz_folder(object sender, RoutedEventArgs e)
{
    czekaj.Visibility = Visibility.Visible; UpdateLayout();
    if (int.TryParse(ID.Text, out Id))
    {
        try
        {
            OpenFileDialog op = new OpenFileDialog();
            op.InitialDirectory = "c:\\\\";
            op.RestoreDirectory = true;

            op.ShowDialog();
            FileInfo info = new FileInfo(op.FileName);
            sciezka = info.Directory.FullName;
        }
        catch
        {
            ID.Text = "#1-!";
        }
    }
    else
    {
        ID.Text = "#2-!";
    }
    czekaj.Visibility = Visibility.Hidden; UpdateLayout();
}
```

Rys. 7. Funkcja kontrolki Wybierz Folder

Funkcja *Wybierz Folder* uwidacznia kontrolkę z obrazkiem *czekaj* i przystępuje do otwarcia okna wyboru pliku, domyślnie ustawionym na dysk C. wraz z wybraniem pliku wewnątrz interesującego nas folderu ścieżka doń zostaje zapisana w zmiennej *sciezka*. W programie widoczne są instrukcje warunkowe – pierwsza sprawdza czy ID podane przez użytkownika jest liczbą całkowitą, pozostałe stanowią bloki try-catch zwracające odpowiedni komunikat na przedwczesne zamknięcie okna wyboru lub inne losowe zdarzenie powodujące niepowodzenie procesu wyboru ścieżki. Na koniec działania funkcji obrazek czekania zostaje na powrót ukryty.

4.2.1.3. Przesyłanie danych

```
private void Przeslij(object sender, RoutedEventArgs e)
{
    czekaj.Visibility = Visibility.Visible; UpdateLayout();

    Dispatcher.Invoke(new Action(Function1), DispatcherPriority.ContextIdle, null);

    czekaj.Visibility = Visibility.Hidden; UpdateLayout();
}
```

Rys. 8. Funkcja kontrolki Przeslij.

Kod funkcji *Przeslij* został ukryty z osobnej bezargumentowej funkcji *Function1* w celu obniżenia priorytetu tego zadania względem renderowania widoku aplikacji. Owe rozwiązanie pozwala na wyświetlenie obrazu czekania zanim program wejdzie w czasochłonną operację pozyskiwania tekstu z plików, zapisywania logów i przesyłania rekordów.

```
public void Function1()
{
    if (int.TryParse(ID.Text, out Id))
    {
        for (int i = 1; i <= 1440; i++)
        {
            try
            {
                if (File.Exists(sciezka + "\\EKG" + i + ".txt"))
                {
                    StreamWriter sw = new StreamWriter("logEKG.txt", true);
                    try { sw.WriteLine("Init EKG" + i); sw.WriteLine(DateTime.Now); }
                    catch { }
                    StreamReader sr = new StreamReader(sciezka + "\\EKG" + i + ".txt");
                    if (sr.Peek() >= 0)
                    {

```

Rys. 9. Funkcja Function1 – 1.

Funkcja *Function1* rozpoczyna się sprawdzeniem czy użytkownik na pewno podał poprawny numer ID. Następnie sprawdza czy rekord o nazwie EKG1.txt, EKG2.txt ... aż do EKG1440.txt istnieje. Jeśli owy istnieje tworzy albo nadpisuje istniejący plik logów oraz otwiera dany rekord EKG w celu odczytu. Rysunek nr 9 kończy się sprawdzeniem czy w danym rekordzie jest co odczytywać.

```

if (sr.Peek() >= 0)
{
    Process process = new Process();
    process.StartInfo.FileName = "cmd.exe";
    process.StartInfo.RedirectStandardInput = true;
    process.StartInfo.RedirectStandardOutput = true;
    process.StartInfo.CreateNoWindow = true;
    process.StartInfo.UseShellExecute = false;
    process.Start();
    process.StandardInput.WriteLine(HEROKU_CLI);
    try
    {
        sw.WriteLine("ansfer heroku...");
        sw.WriteLine(process.StandardOutput.ReadLine());
        sw.WriteLine(process.StandardOutput.ReadLine());
    } catch { }
    process.StandardInput.Flush();
    process.StandardInput.Write("insert into przebiegis (user_id, zapis) values (" +
                                                                    ID.Text + ", '");
    process.StandardInput.Write("{");
    while (sr.Peek() >= 0)
    {
        process.StandardInput.Write((char)sr.Read());
    }
    sr.Close();
    process.StandardInput.Write("0}');");
    process.StandardInput.Flush();
    process.StandardInput.Close();
}

```

Rys. 10. Funkcja Function1 – 2.

Na rysunku nr 10 funkcja po sprawdzeniu czy rekord jest pusty przystępuje do rozpoczęcia procesu konsolowego, jednocześnie ukrywając konsolę przed użytkownikiem. Przy pomocy funkcji `.StandardInput.WriteLine()` zostaje wprowadzone polecenie ze zmiennej `HEROKU_CLI`. Wewnątrz bloku `try{}` program zapisuje logi do pliku. Dalej program wprowadza początek zapytania SQL, po czym odczytuje zawartość pliku tekstowego z EKG wprowadzając ją znak po znaku do konsoli. Po odczytaniu całego pliku owy jest zamykany, zapytanie jest więczone oraz przesłane funkcją `.StandardInput.Flush()`. Konsola zostaje zamknięta. Na Rysunku nr 11 do pliku logów zostają zapisane ostatnie linijki oraz sprawdzane jest czy wykonanie zapytania zakończyło się powodzeniem.

```

        process.StandardInput.Close();
        try
        {
            sw.WriteLine("ansfer query...");
            string spr = process.StandardOutput.ReadToEnd();
            sw.WriteLine(spr);
            if (!spr.Contains("INSERT 0 1")) { ID.Text = "#4-!"; };
        } catch { }
        sw.Close();
        process.WaitForExit();
    }
}
catch
{
    ID.Text = "#3-!";
}
}
else
{
    ID.Text = "#2-!";
}
}

```

Rys. 11. Funkcja Function1 – 3.

4.2.1.4. Informacje o błędach

Rysunki nr 7-11 zawierają instrukcję wyrzucającą informację o błędach do pola tekstowego aplikacji. Informacje te są następujące:

- #1-! – oznacza niepowodzenie wyboru folderu.
- #2-! – oznacza wprowadzenie niepoprawnego numeru ID.
- #3-! – oznacza niepowodzenie w trakcie odczytu lub przesyłania danych.

W celu dokładniejszego sprawdzenia reakcji konsoli, należy sprawdzić zawartość pliku logów aplikacji.

- #4-! – oznacza nieskuteczność umieszczenia rekordu w bazie danych.

Wystąpienie błędów 3-4 może być spowodowane brakiem zainstalowanych programów psql oraz Heroku CLI oraz zalogowania użytkownika na swoje konto Heroku.

4.2.2. Aplikacja ZapisEKG – specyfikacja zewnętrzna

4.2.2.1. Instalacja Heroku CLI i psql

Aby aplikacja mogła zadziałać należy zainstalować:

- Heroku CLI - <https://devcenter.heroku.com/articles/heroku-cli>

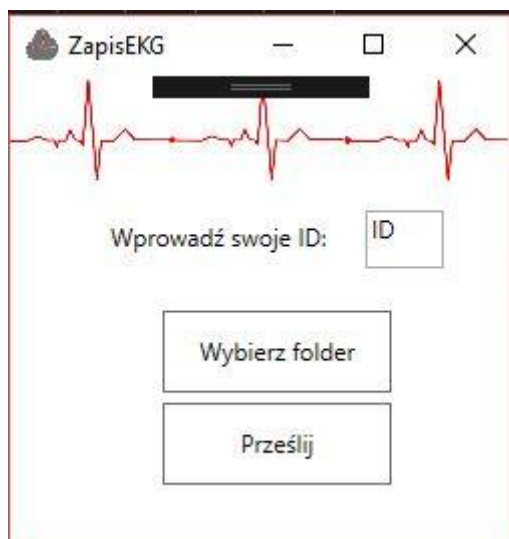
Po zainstalowaniu aplikacji Heroku, należy zalogować się na konto bazy danych przy pomocy konsoli. Należy uruchomić program cmd i po otwarciu się konsoli wpisać *heroku login*, po tym program poprosi o następujące dane:

- Email: kamisuc671@student.polsl.pl
 - Hasło: K0070505033581s
- psql - <https://www.postgresql.org/download/>

4.2.2.2. Start Aplikacji



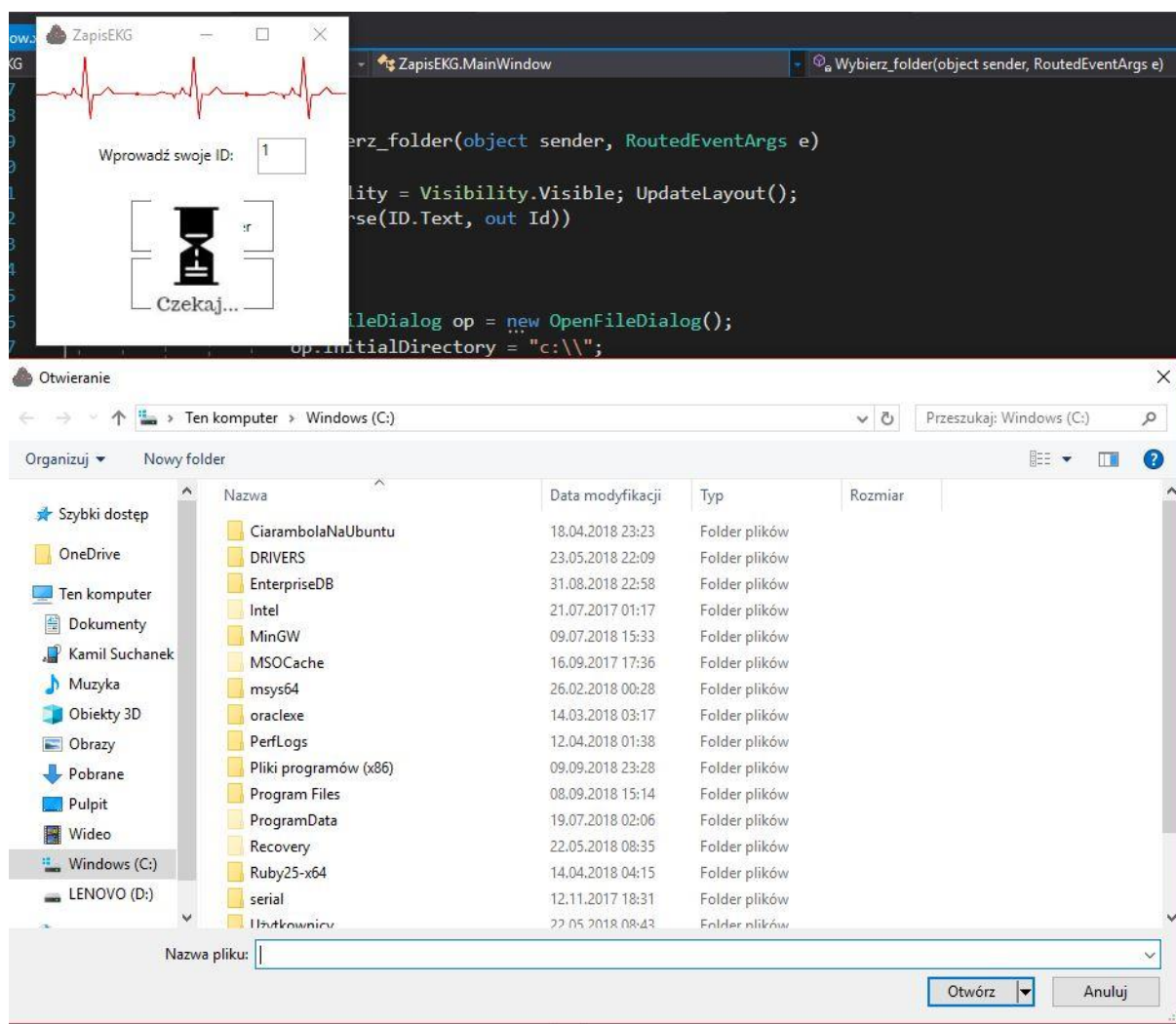
Rys. 12. Ikona aplikacji.



Rys. 13. Okno startowe aplikacji.

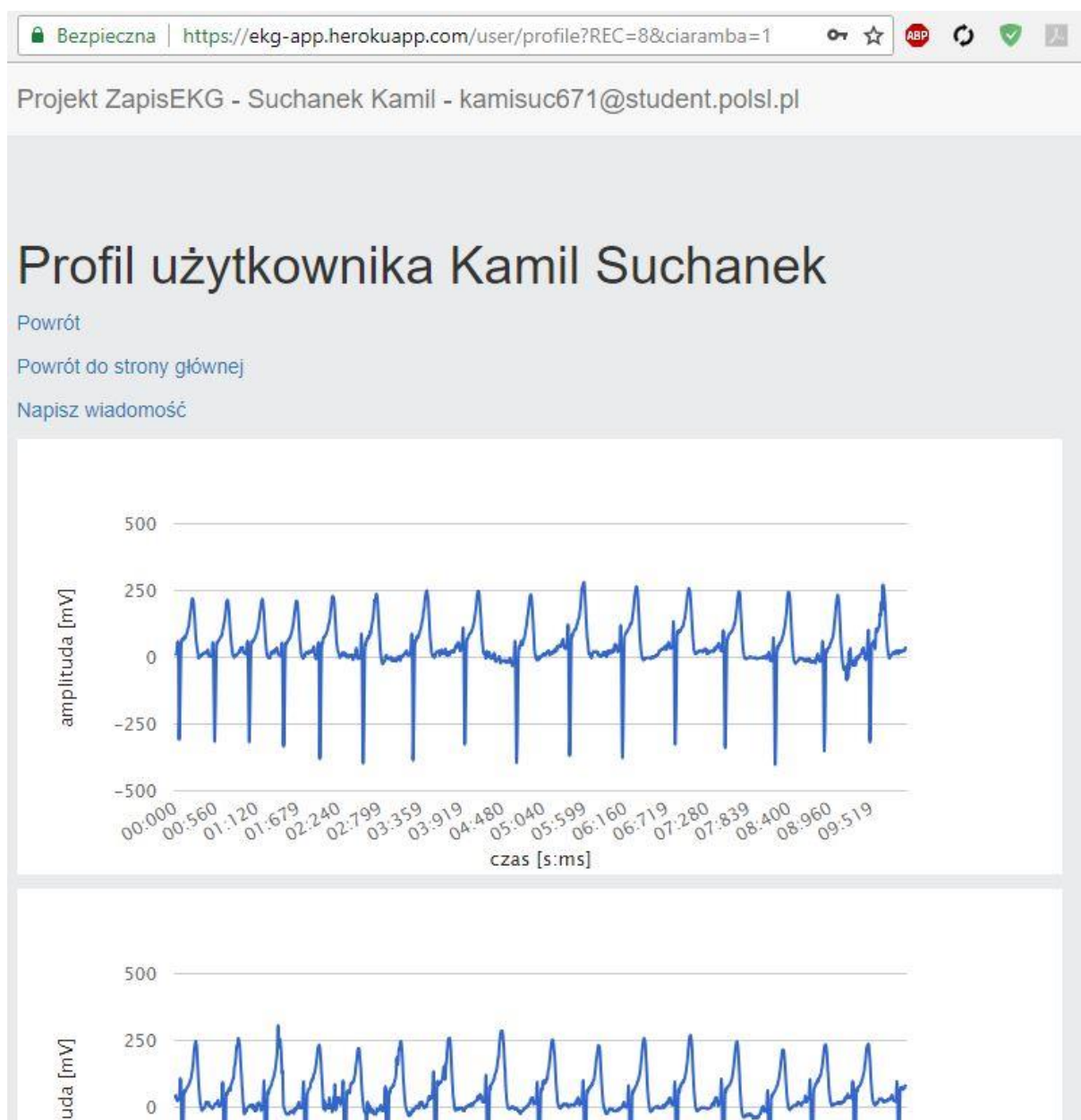
Należy uruchomić aplikację. Otwarte zostanie okno, do którego należy wprowadzić ID swojego konta na stronie internetowej. Musi to być liczba całkowita.

4.2.2.3. Wybór folderu i przesyłanie danych



Rys. 14. Wybór folderu.

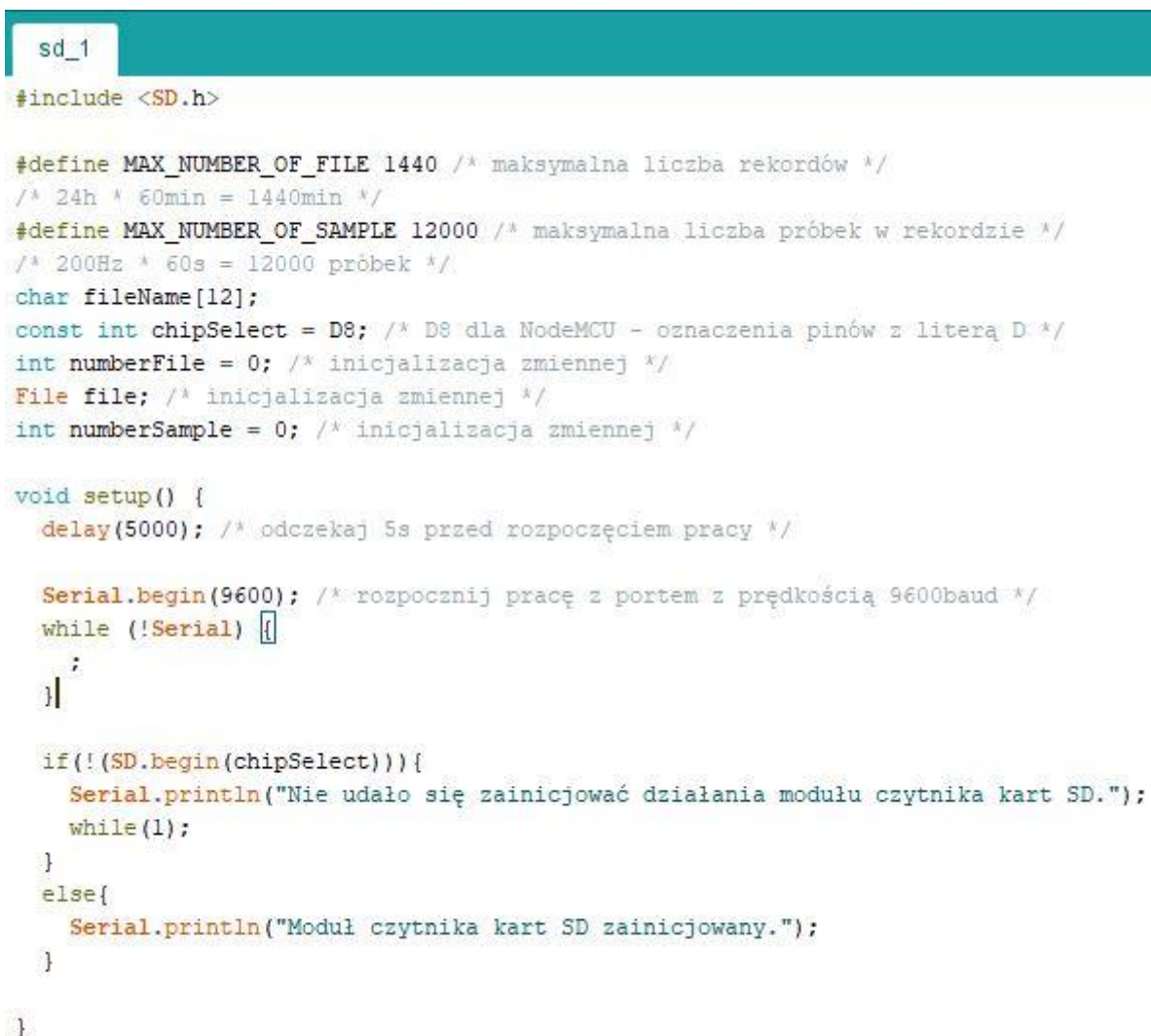
Po wprowadzeniu poprawnego numeru ID, klikamy w przycisk *Wybierz Folder*, przyciski przykrywa informacja o oczekiwaniu, a naszym zadaniem jest znaleźć folder z naszymi rekordami, otworzenie go i wybranie dowolnego pliku tekstowego znajdującego się w środku poprzez dwukrotne kliknięcie myszki albo wybranie pliku i zatwierdzenie przyciskiem *Otwórz*. Po wybraniu folderu aplikacja powróci to wyglądu startowego, upewniając się że w polu tekstowym nie ma informacji o błędzie, tylko nasz numer ID klikamy przycisk *Prześlij*. Ponownie pokazuje się klepsydra z informacją o oczekiwaniu. Po jej zniknięciu można zakończyć działanie programu i sprawdzić obecność rekordów na stronie internetowej Rys. 15.)



Rys. 15. Strona internetowa z rekordami EKG.

4.3. Rejestracja i zapis rekordów

Rejestracja sygnału może być realizowana przez układ NodeMCU ze względu na wbudowany dzielnik napięcia przy przetworniku ADC. Klasyczne moduły ESP nie znosiły dobrze napięcia powyżej 1V na owym przetworniku, gdzie moduł EKG dostarcza sygnał do 3V. Program dla tej czynności zawiera się na rysunku nr 16-17.



```
sd_1

#include <SD.h>

#define MAX_NUMBER_OF_FILE 1440 /* maksymalna liczba rekordów */
/* 24h * 60min = 1440min */
#define MAX_NUMBER_OF_SAMPLE 12000 /* maksymalna liczba próbek w rekordzie */
/* 200Hz * 60s = 12000 próbek */
char fileName[12];
const int chipSelect = D8; /* D8 dla NodeMCU - oznaczenia pinów z literą D */
int numberFile = 0; /* inicjalizacja zmiennej */
File file; /* inicjalizacja zmiennej */
int numberSample = 0; /* inicjalizacja zmiennej */

void setup() {
    delay(5000); /* odczekaj 5s przed rozpoczęciem pracy */

    Serial.begin(9600); /* rozpocznij pracę z portem z prędkością 9600baud */
    while (!Serial) {}
}

if(!SD.begin(chipSelect)){
    Serial.println("Nie udało się zainicjować działania modułu czytnika kart SD.");
    while(1);
}
else{
    Serial.println("Moduł czytnika kart SD zainicjowany.");
}
}
```

Rys. 16. Zmienne globalne i funkcja setup().

Zgodnie z założeniami, rekordy mają być umieszczane w bazie danych przynajmniej raz dziennie, więc maksymalna liczba rekordów wynosi 1440, gdzie każdy trwa minutę. Każdy z rekordów zawiera 12000 amplitud sygnału. Funkcja setup() uruchamia port szeregowy oraz inicjuje działanie modułu czytnika kart SD.


```

void loop(void) {

if(numberFile<MAX_NUMBER_OF_FILE){

    numberFile +=1; /* nowy numer rekordu */
    sprintf(fileName,"EKG%d.txt",numberFile); /* utworzenie nazwy rekordu */

    Serial.print(fileName);Serial.println(" ");/* informacja o pracy z danym plikiem na port */

    if(!SD.exists(fileName)){ /* Jeśli taki plik jeszcze nie istnieje na karcie */

        file = SD.open(fileName,FILE_WRITE); /* Stwórz ten plik */

        if(file){ /* Jeśli utworzenie pliku się powiodło */

            Serial.println("Utworzono plik: ");Serial.println(fileName);
            //int kontrolka = 0; int sekunda = 1;
            while (numberSample < MAX_NUMBER_OF_SAMPLE){
                numberSample +=1;
                file.print(String(analogRead(A0)));file.print(",");delay(5);
                /* delay[ms] da częstotliwość 200Hz */
                //kontrolka++;
                //if(kontrolka>=200){kontrolka=0;Serial.print(sekunda++);}
            }
            numberSample = 0;
            file.close();

            }else{Serial.println("Utworzenie pliku się nie powiodło!");}
        }else{Serial.println("Taki plik już istnieje!");}
    }else{Serial.println("Zapis na dzień zakończony!");ESP.deepSleep(0);}
}

```

Rys. 17. Funkcja loop().

W pętli programu urządzenia sprawdzana jest liczba utworzonych plików, w razie niespodziewanego restartu urządzenia, zamiast nadpisywać poprzednie rekordy program przeskoczy każdy istniejący aż będzie mógł utworzyć nowy plik by zapisywać na nim sygnał. Nawet jeśli dany rekord pozostał niedokończony to kontynuowanie go nie ma sensu, gdyż to jest już inny czas i inny zapis. Każde wykonanie pętli zatrzymuje się na minutę by zapisać 12000 próbek sygnału. Gdy urządzenie zapisze 1440 rekordów odpowiadających za 24 godziny urządzenie zostanie uśpione, kartę będzie można wyjąć, zgrać rekordy, włożyć ponownie do urządzenia i zresetować płytkę NodeMCU (posiada przycisk resetu).

5. Aneks do sprawozdania

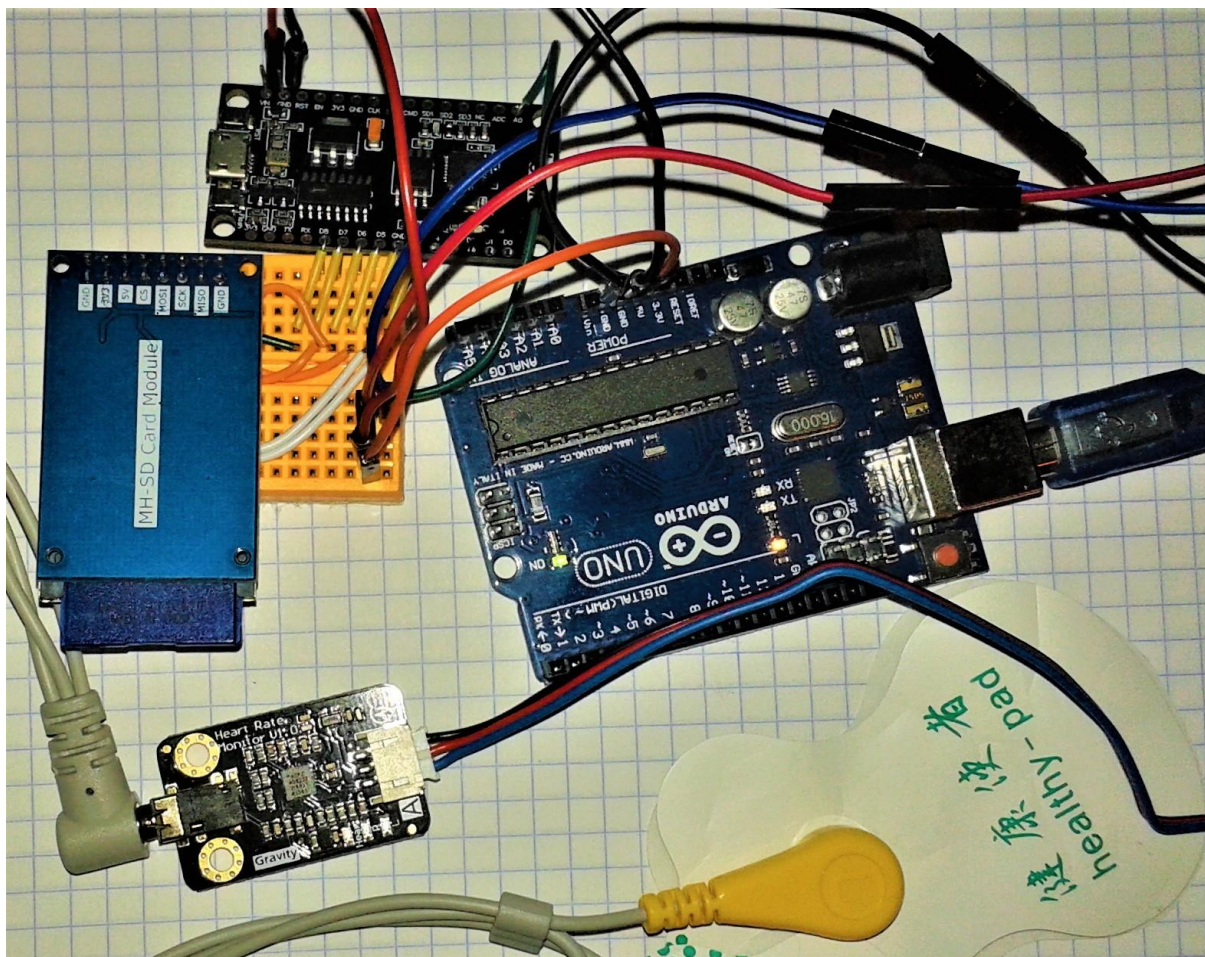
Projekt w ramach kursu Programowanie Urządzeń mobilnych polegał na stworzeniu strony internetowej w technologii Ruby on Rails. Owa strona służy do przechowywania i wyświetlania zapisów EKG pochodzących z zewnątrz, np. z urządzenia mobilnego, którego prototyp jest tematem tego projektu. Na stronie możliwe jest utworzenie swojego konta. Stworzona strona została pokazana na Rysunku nr 15. Ostatecznie, projekt w ramach PUM był raczej całkiem niezależny od tego projektu, jednak utworzone rozwiązanie w tym przypadku miało za zadanie dostarczać rekordy EKG na tą konkretną stronę internetową.

Repozytorium projektu z PUM (link prowadzi do sprawozdania zawartego w owym repozytorium) :

<https://github.com/KamilSuchanek95/ZapisEKG/blob/new/Suchanek-Kamil-Projekt-PUM.pdf>

6. Podsumowanie

Powstałe urządzenie jest zdolne do rejestracji sygnału przez całą dobę. Płytkę Arduino na zdjęciu została użyta jako zasilacz dla układu (została wpięta do komputera). Na samej górze zdjęcia (Rys. 18.) znajduje się płyta NodeMCU, poniżej moduł czytnika kart SD, a na samym dole moduł EKG z przypiętymi elektrodami. Po dniu rejestracji danych można wyjąć kartę i wpiąć do laptopa w celu zgrania sygnałów na stronę internetową.



Rys. 18. Prototyp urządzenia do rejestracji EKG.

