

Autorzy	
Oleksii Kostiukov	231972
Uladzimir Lipski	238961
Vikatr Hasiul	231862

---

Prowadzący	Dr inż. Marek Woda
Termin	Środa, godzina: 13:15

**Zastosowanie systemów wbudowanych**  
*Raspberry Pi - Telegram Bot i web-aplikacja dla  
śledzenia klimatu otoczenia*

# 1 Cel projektu

Celem danego projektu jest wykorzystanie platformy *Raspberry* do realizacji *Telegram bot*'u oraz punktu pomiarowego.

**Telegram bot** jest aplikacją wykorzystującą interfejs aplikacji *Telegram* w celu komunikacji z wybranymi użytkownikami (którzy posiadają możliwość komunikacji ze stworzonym *bot'em*).

**Punkt pomiarowy** Platforma *Raspberry* umożliwia podłączenie licznych czujników, dane z których można gromadzić na urządzeniu lub wysyłać do serwerów zdalnych. W danym projekcie zostaną podłączone czujniki:

1. temperatury,
2. wilgotności,
3. światła

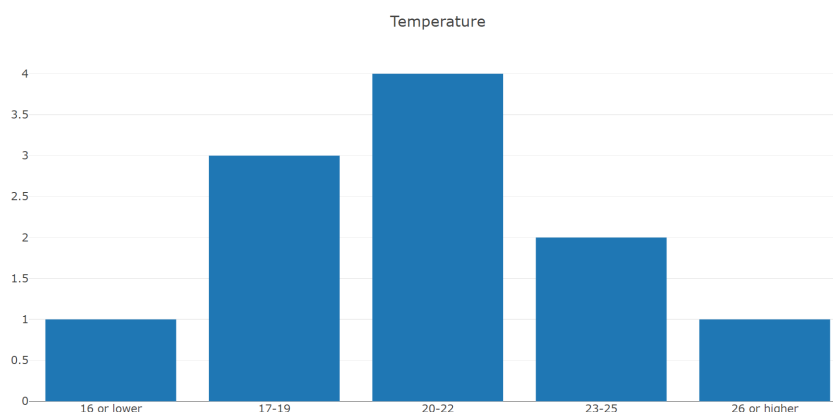
dane z których będą przechowywane na urządzeniu w celu przetwarzania i wyświetlania na stronie **WEB** w postaci interaktywnego wykresu.

Dodatkowo dany zbiór danych zostanie wykorzystany przez *Telegram bot* w celu powiadomienia użytkownika o aktualnych danych.

## 2 Implementacja asynchronicznego wyświetlania wyników pomiarowych

### 2.1 Cel aplikacji internetowej

Celem aplikacji internetowej jest wizualizacja danych, pobranych za pomocą czujników podłączonych do platformy Raspberry Pi. Dane są wyświetlane za pomocą histogramów. Wykresy pokazują jak często występuje każdy z zadanych zakresów temperatury, natężenia światła i wilgotności. Wykresy w przeglądarce muszą się odświeżać asynchronicznie po zmianie pliku z danymi.



Rysunek 1: Histogram temperatury

## 2.2 Wybór platformy web-serwera

Aplikacja internetowa została stworzona za pomocą *frameworku Flask*. *Flask - framework* do tworzenia aplikacji internetowych w języku *Python*, głównie do minimalistycznych aplikacji, które celowo zapewniają tylko podstawowe funkcje.

## 2.3 Generowanie wykresów w przeglądarce

Dla utworzenia wykresów w przeglądarce używana jest biblioteka *Plotly*, napisana w języku *JavaScript*. Biblioteka pozwala na utworzenie różnorodnych wykresów, z których dla danej aplikacji jest typem *bar*, odpowiadającym histogramowi. Dane dla wykresów są przekazywane od serwera do przeglądarki jako odpowiedź na żądanie *HTTP*.

Żądanie jest obsługiwane w języku *JavaScript* za pomocą ciągu funkcji, które zostały omówione w dalszej części dokumentacji.

## 2.4 Tworzenie wykresu

Listing 1: Funkcja do tworzenia histogramu za pomocą biblioteki *Plotly.js*

```
1 function drawChart(chart_divId, chart_title) {
2     var trace = {
3         type: 'bar',
4         x: chartRanges[chart_title],
5         y: chartData[chart_title],
6     };
7
8     var data = [ trace ];
9
10    chart_title = chart_title.charAt(0).toUpperCase()+chart_title.slice(1);
11    var layout = {
12        title: chart_title,
13        font: {size: 18}
14    };
15
16    Plotly.react(chart_divId, data, layout);
17 }
```

Dane z serwera są przekazywane w postaci tablicy. Dodatkowym zadaniem skryptu w języku *JavaScript* jest kategoryzowanie danych dla różnych zakresów, zanim wykres będzie stworzony.

Wartości dla osi *x* są nazwami zakresów, do których może należeć wartość przekazana z serwera. Dane zakresy są zdefiniowane statycznie i zadeklarowane na samym początku skryptu.

Listing 2: Zdefiniowany zakresy w skrypcie *JavaScript*

```
1 chartRanges = {
2     "temperature" :
3         ["16 or lower", "17-19", "20-22", "23-25", "26 or higher"],
4     "light":
5         ["349 or lower", "350-450", "451-550", "551-650", "651 or higher"],
6     "humidity":
7         ["0-20", "21-40", "41-60", "61-80", "81-100"]
8 }
```

## 2.5 Asynchroniczne wczytywanie danych z pliku

Dane, pobrane z czujników za pomocą platformy *Raspberry Pi*, są przechowywane w pliku z rozszerzeniem `.csv`. Dla asynchronicznego wczytywania danych z pliku została użyta moduł `watchdog`. Dany moduł został użyty do monitorowania wybranych plików. Wybór plików został zdefiniowany za pomocą wyrażeń regularnych. Jeżeli dowolny plik o zadanym rozszerzeniu zostaje zmieniony, to do serwera jest wysyłany sygnał. Do obsługi różnego rodzaju sygnałów jest używany moduł `flask.socketio`.

Listing 3: Połączenie gniazdka na serwerze z gniazdkiem w przeglądarce

```
1 @socketio.on('connect')
2 def test_connect():
3     global thread
4     if thread is None:
5         thread = socketio.start_background_task(target=background_thread)
```

Dla każdego sygnału mogą być zdefiniowane osobne funkcje. W danej implementacji sygnał zostaje wysłany jeżeli monitorowany plik został zmieniony, to serwer wysyła sygnał do przeglądarki za pomocą funkcji `on_modified`, która dodatkowo wczytuje dane z pliku, żeby przekazać do przeglądarki.

Listing 4: Definicja wątku, monitorującego zmiany w plikach o zadanych rozszerzeniu

```
1 def background_thread():
2     global observer
3     event_handler = CsvWatcher()
4     observer = Observer()
5     observer.schedule(event_handler, './', recursive=True)
6     observer.start()
```

Listing 5: Definicja funkcji wysyłającej sygnał i dane

```
1 def on_modified(self, event):
2     global csvFileName
3     data = readfile(csvFileName)
4     socketio.emit('modified', {'data': data})
```

## 2.6 Asynchroniczne odświeżanie wykresów w przeglądarce

Asynchroniczne odświeżanie wykresów umożliwia połączenie przeglądarki i serwera za pomocą gniazdek. Celem gniazdek jest asynchroniczna obsługa sygnałów, wysłanych z serwera.

Do obsługi sygnału w języku JavaScript zostały zdefiniowane następujące funkcje:

Listing 6: Połączenie gniazdek w przeglądarce z serwerem i odebranie danych

```
1 var socket = io.connect('http://' + document.domain + ':' + location.port);
2
3 socket.on('modified', function(data) {
4     var newData = data['data'];
5     start(newData)
6 });
```

### 3 Telegram Bot

*Telegram Bot* jest aplikacją bazującą się na API<sup>1</sup> udostępnionego przez **Telegram** w celu tworzenia **Bot'a** – aplikacji odpowiadającej na określone żądania użytkownika w określony sposób. Jest to pewien rodzaj usługi, którą możemy stworzyć i wdrożyć w danym messengerze.

Usługą danego projektu jest udostępnianie wyników pomiarowych, które są zbierane przez niezależną aplikację, działającą na platformie **Raspberry Pi 3B+**.

Proces tworzenia **Bot'a** jest opisany na stronie – <https://core.telegram.org/bots>

W danej implementacji **Bot'a** zostały zrealizowane następujące funkcjonalności:

- Wgląd do wyników pomiarowych – odczyt ostatnich danych pomiarowych: Użytkownik posiada możliwość zdefiniowania ilości ostatnich wyników pomiarowych.

Listing 7: Komenda do pobierania określonej ilości ostatnich wyników pomiarowych

```
/tail <size>
```

gdzie <size> określa ilość pomiarów.

Listing 8: Odpowiedź na wprowadzoną komendę – /tail

```
def rasp_tail(update, context):
    tail_size = 10
    if len(context.args) != 0:
        tail_size = int(context.args[0])

    if tail_size < 0:
        tail_size *= -1
    data = read_csv('data.csv').tail(tail_size)
    for name in TRUE_ARGS.values():
        msg = name
            +':\n'
            +re.sub('_{2,}', '_',
                    data.loc[:,name].to_string(index=False))
        update.message.reply_text(msg)
```

---

<sup>1</sup>od ang. Application Programming Interface – programistyczny interfejs aplikacji

- Przeglądanie wyników pomiarowych w postaci graficznej – wykresy typu *histogram*: Użytkownik posiada możliwość wyboru typu danych pomiarowych.

Listing 9: Komenda do pobierania wyników określonego typu, w postaci graficznej

```
/pic <type>
```

gdzie <type> określa jeden z możliwych typów pomiarowych:

temp, humid, light – temperatura, wilgotność, natężenie światła.

Listing 10: Odpowiedź na wprowadzoną komendę – /pic

```
def rasp_pic(update, context):
    picture_size = None
    picture_type = None
    if len(context.args) > 1:
        picture_type = context.args[0]
        picture_size = int(context.args[1])
        file_name = create_plot(TRUE_ARGS[picture_type], picture_size)
        context.bot.send_photo(chat_id=update.effective_chat.id,
                               photo=open(file_name, 'rb'))
    elif len(context.args) == 1:
        picture_type = context.args[0]
        file_name = create_plot(TRUE_ARGS[picture_type])
        context.bot.send_photo(chat_id=update.effective_chat.id,
                               photo=open(file_name, 'rb'))
    else:
        update.message.reply_text('Invalid args')
```

## 4 Generowanie i przechowywanie danych

Generowanie danych jest oparte o informacje uzyskiwaną z czujników pomiarowych:

- DHT11 - czujnik temperatury i wilgotności,
- BH1750 - czujnik natężenia światła.

Zakresy pomiarowe czujnika DHT11 stanowią pewne ograniczenie na jego zastosowanie:

- Temperatura –  $[0,50] \pm 2^\circ$  o rozdzielczości  $1^\circ$ ,
- wilgotność –  $[20,90]\%$  RH - wilgotność względna. Jest to stosunek rzeczywistej wilgoci w powietrzu do maksymalnej jej ilości, którą może utrzymać powietrze w danej temperaturze. Rozdzielczość –  $1\%$ .

Z powodu powyższych ograniczeń, dany czujnik można stosować w warunkach, które nie wymagają większej precyzji, na przykład – pomieszczenia biurowe.

Czujnik natężenia światła posiada szeroki zakres pomiarowy –  $[1,65535]lx$ , z rozdzielczością 1 lub 4  $lx$  w zależności od wybranego trybu pracy. W danej implementacji została wybrana rozdzielczość w 1  $lx$ .

## 4.1 Przechowywanie danych

Ze względu na małą ilość zbieranych danych, 3 wartości typu `int`, w zakresie `[0, 65535]`, przez długi okres czasowy (do 150 dni, przy okresie pomiarowym 1 minuta), został zdefiniowany plik w formacie `csv`.

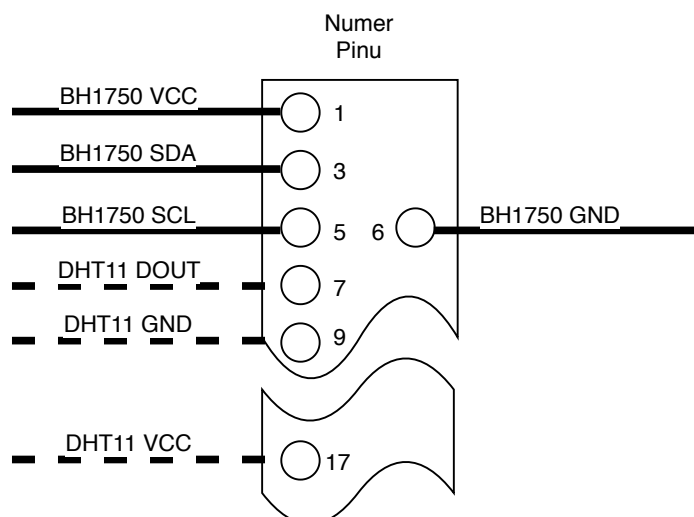
Dane rozwiązanie zostało przyjęte ze względu na przechowywane dane, łatwość w użyciu, w porównaniu do baz danych, oraz brak nadmiarowych informacji – plik przechowuje tylko te dane, które zostały do niego zapisane.

Listing 11: Definicja pliku `csv` z przykładowymi danymi

```
Temperature , Light , Humidity
25 , 380 , 57
```

## 4.2 Podłączenie czujników

Czujniki zostały podłączone do wejść GPIO platformy *Raspberry Pi 3B+* w poniższy sposób:



Rysunek 2: Połączenie pinów czujników z wejściami GPIO

W celu komunikacji i uzyskania danych z czujników zostały użyte moduły:

- `smbus`
- `Adafruit-DHT`

Dla uzyskiwania danych co określony czas (domyślne ustawione na 1 minutę) został napisany proces, który pobiera dane z czujników i zapisuje je do pliku oraz proces monitorujący:

Listing 12: Definicja obiektu nadzorującego proces zbierający dane

```
class Sensors():
    def __init__(self):
        self.thread = None
        self.runner = runner.Runner(60)
        self.is_running = threading.Event()
```

Listing 13: Inicjalizacja procesu zbierającego dane

```
def reset_sensors(self, _sleep=60):
    """
    Function initialize the thread
    which will collect data from sensors
    based on provided 'sleep' timeout
    Timeout = [60,3600]
    """
    if self.thread is None:
        self.is_running.clear()
        self.runner.set_sleep(_sleep)
        self.thread = threading.Thread(target=self.runner.launch,
                                       args=(self.is_running,))
        self.thread.start()
    ...
```

Listing 14: Definicja procesu zbierającego dane

```
class Runner():
    def __init__(self, _sleep):
        self.sleep = _sleep
        self.data = data.Sensors_data()
        self.bus = smbus.SMBus(1)
        self.light = bh1750.BH1750(self.bus)
        self.dht = dht11.DHT()
```

Listing 15: Zbiór i zapis danych

```
def launch(self, is_running):
    while not is_running.is_set():
        _humid, _temp = self.dht.read()
        if _humid == -1:
            time.sleep(self.sleep)
        else:
            _light = int(self.light.measure_high_res())
            self.data.save_average([_temp, _light, _humid])
            time.sleep(self.sleep)
```



## 5 Łączenie z siecią WiFi

W celu połączenia z siecią użytkownika WiFi o standardzie WPA lub WPA2 została napisana aplikacja, która pobiera dane sieci:

- SSID<sup>2</sup>,
- hasło.

które zostają przekazane do aplikacji:

- `wpa_passphrase` – generowanie klucza dla połączenia,
- `wpa_supplicant` – w celu połączenia z siecią za pomocą klucza i SSID

Listing 16: Pobieranie danych z form za pomocą metody POST

```
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        subprocess.Popen(['./switch_network.sh', request.form['ssid'],
                           request.form['password']], shell=True)
        return render_template('index.html')
    else:
        return render_template('index.html')
```

Generowanie klucza dla połączenia zostało napisane w języku Bash:

Listing 17: Generowanie klucza połączenia

```
#!/bin/bash
#user input for SSID
#creating a config file for network
echo "ctrl_interface=DIR=/var/run/wpa_supplicant_GROUP=netdev" \
| sudo tee /etc/wpa_supplicant/wpa_supplicant.conf
echo "update_config=1" \
| sudo tee -a /etc/wpa_supplicant/wpa_supplicant.conf
wpa_passphrase $1 $2 \
| sudo tee -a /etc/wpa_supplicant/wpa_supplicant.conf
sudo reboot now
```

Po restarcie system będzie próbował połączyć się ze wskazaną siecią.

## 6 Uruchomienie w środowisku *Raspbian*

Algorytm działania całego systemu jest następujący:

1. Sprawdzenie zawartości konfiguracji WiFi,
2. jeżeli istnieje wpis o określonym SSID następuje połączenie z nim,
3. jeżeli nie został przydzielony address IP (sieć nie jest aktywna) następuje uruchomienie własnego AP <sup>3</sup> oraz aplikacji konfiguracyjnej,

<sup>2</sup>od ang. Service set identifier - identyfikator sieci

<sup>3</sup>od ang. Access Point – punkt dostępu do sieci (w danym przypadku WiFi)

4. jeżeli adres został przydzielony – uruchomienie Telegram Bot’u, aplikacji pomiarowej oraz aplikacji WEB wyświetlającej dane.

Powyższy algorytm został napisany w języku Bash.

Listing 18: Uruchomienie procesów w systemie *Raspbian*

```
#!/bin/sh
working_dir="/home/pi/ZWS_rasp_tg/"
sudo service wpa_supplicant restart
sudo wpa_supplicant -D wext -i wlan0 \
  -c /etc/wpa_supplicant/wpa_supplicant.conf -B
WPA_PID=$!
sleep 2
sudo dhclient -1
if [ $? -eq 0 ]
then
    echo "Connected"
    cd $working_dir && python3 ./Telegram_Bot/rasp_bot.py &
    cd $working_dir && python3 ./Web_app/app.py &
else
    echo "No internet"
    sudo kill $WPA_PID
    sudo service wpa_supplicant stop
    sudo iptables-restore /etc/iptables.ipv4.nat
    sudo ifup wlan0
    sudo service hostapd start
    sudo service dnsmasq start
    cd $working_dir && python ./network_auth/app.py
fi
```

W celu uruchomienia danego skryptu podczas startu sytemu bez konieczności ręcznego wywołania, został napisany **service**:

Listing 19: **service** wywołujący skrypt

```
#!/bin/sh
#
### BEGIN INIT INFO
# Provides:          checkInet
# Required-Start:    $remote_fs $syslog
# Required-Stop:     $remote_fs $syslog
# Should-Start:
# Should-Stop:
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Starting script as 'pi' user
# Description:
### END INIT INFO
su pi /home/pi/ZWS_rasp_tg/start.sh
```