Autorzy	
Kostiukov Oleksii	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.

Telegam 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

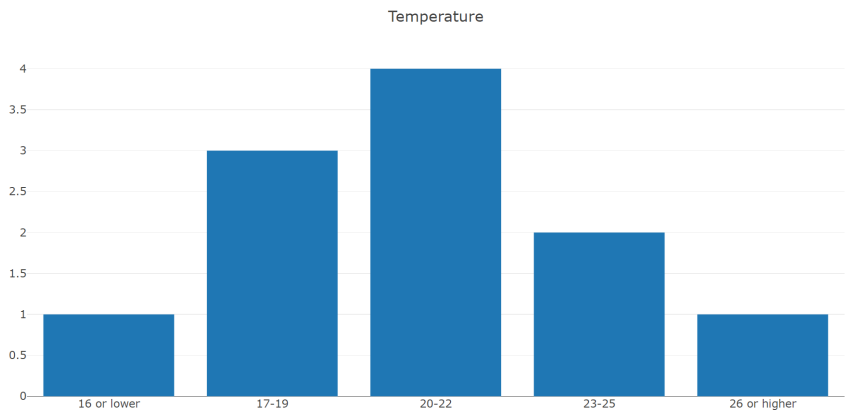
dane z których będą przechowywane na urządzeniu w celu przetwarzania i wyświetlania na stronie **WEB** w postaci interaktywnego wykresu.

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 odpowiedz 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*

```
function drawChart(chart_divId, chart_title) {
  var trace = {
    type: 'bar',
    x: chartRanges[chart_title],
    y: chartData[chart_title],
  };

  var data = [ trace ];

  chart_title = chart_title.charAt(0).toUpperCase()+chart_title.slice(1);
  var layout = {
    title: chart_title,
    font: {size: 18}
  };

  Plotly.react(chart_divId, data, layout);
}
```

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 statyczne i zadeklarowane na samym początku skryptu.

Listing 2: Zdefiniowany zakresy w skrypcie `JavaScript`

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

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 monitorowanie 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 przeglądrce

```
@socketio.on('connect')
def test_connect():
global thread
if thread is None:
  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

```
def background_thread():
  global observer
  event_handler = CsvWatcher()
  observer = Observer()
  observer.schedule(event_handler, './', recursive=True)
  observer.start()
```

Listing 5: Definicja funkcji wysyłającej sygnał i dane

```
def on_modified(self,event):
    global csvFileName
    data = readfile(csvFileName)
    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 funkcję:

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

```
var socket = io.connect('http://'+document.domain+':'+location.port);

socket.on('modified', function(data) {
    var newData = data['data'];
    start(newData)
});
```

3 Telegram Bot

Telegram Bot jest aplikacją bazującą się na `API`[1] 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.

Uługą 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 pomimarowych: Użytkownik posiada możliwość zdefiniowania ilości ostatnich wyników pomiarowych.

  Listing 7: Komedna 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)
  ```

[1] 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: Komedna 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,
- `BHT1750` - czujnik natężenia światła.

Zakresy pomiarowe czujniku `DHT11` stanowią pewne ograniczenie na jego zastosowanie:

- Temeperatura – [0,50] $\pm 2^\circ$ o rozdzielczości 1°,
- 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

4.2 Podłączenie czujników