Dokumentacja aplikacji weatherApp.

Damian Lewandowski

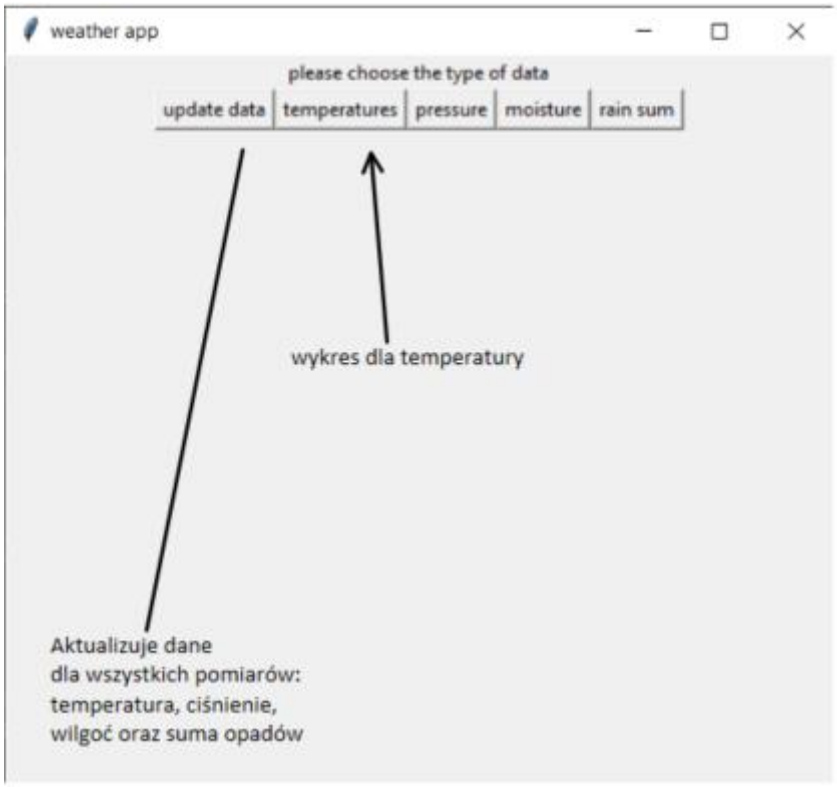
IEiT, Informatyka - Data Science

System zbierania, gromadzenia i prezentacji danych meteorologicznych.

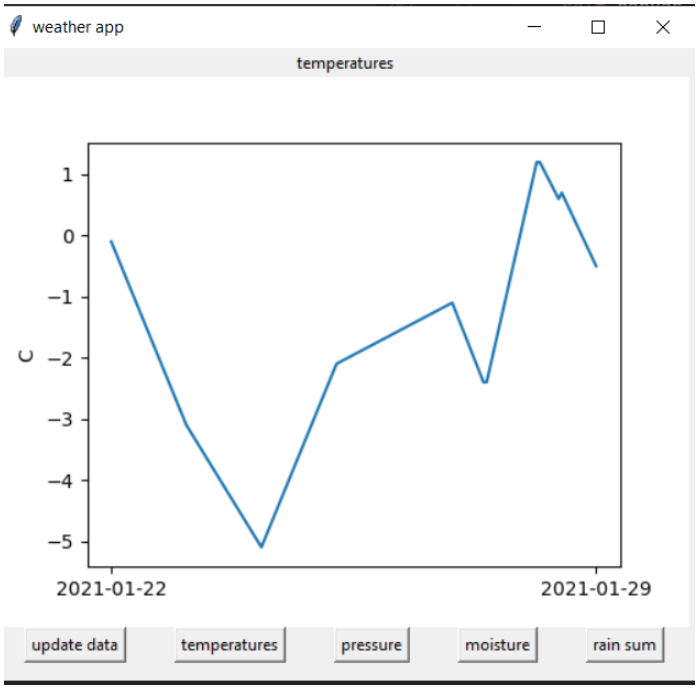
Opis systemu:

Aplikacja weather app monitoruje dane z Krakowa takie jak temperatura, ciśnienie, wilgoć oraz ilość opadów.

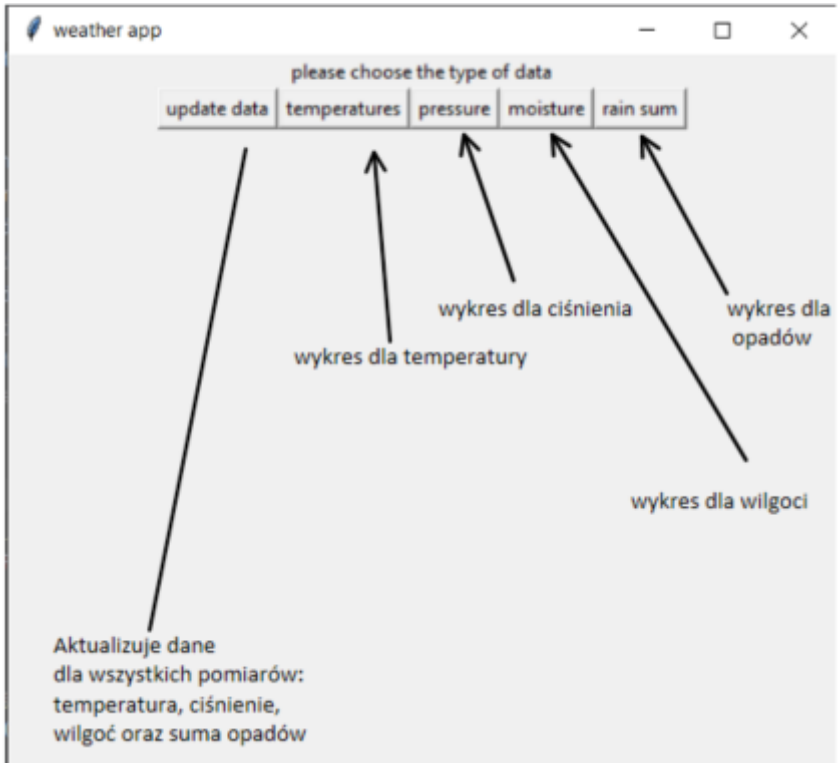
Przewodnik użytkownika dla aplikacji weather app:



Po wykonaniu funkcji update data i wybraniu przycisku dla temperatury uzyskujemy wykres:



Analogicznie dla pozostałych opcji:



Wybierając odpowiedni przycisk dla parametru który nas interesuje uzyskujemy wykres jak zmieniał się dany parametr w danym dniu.

Opis systemu:

Aplikacja wykorzystuje pakiety takie jak requests, json, os, datatime, tkinter oraz matplotlib

Krótki opis wybranych pakietów:

Pakiet JSON umożliwia przechowywanie i wymianę danych oraz pakiet umożliwia przekonwertować plik typu JSON na Python.

Moduł request pozwala na wysyłanie żądań HTTP w Python, dane do aplikacji były pobierane z strony:

https://danepubliczne.imgw.pl/api/data/synop/id/12566

Pakiet Tkinter - jego podstawowym elementem jest okno GUI. Pakiet ten został wykorzystany do stworzenia interfejsu aplikacji. Pola tekstowe, etykiety, przyciski dla aplikacji weather app.

Moduł matplotlib jest narzędziem do rysowania wykresów które zostały stworzone w tej aplikacji za jego pomocą.

Aplikacja została stworzona w celu gromadzenia danych takich jak temperatura, ciśnienie, wilgoć i ilość opadów. Te dane następnie zostały przeniesione na wykresy. Daje nam informacje jak zmieniała się powyższe parametry w poszczególnych dniach.

Przewodnik instalacji i podręcznik administratora:

Aby uruchomić program pobieramy python'a na naszą platformę z strony:

https://www.python.org/downloads/

Potrzebne nam będzie środowisko IDE dla python'a może to być pycharm, visual code, atom lub inne.

Za pomocą odpowiedniego środowiska uruchamiamy skrypt.

Kod źródłowy:

```
import requests
import json
import os
import datetime
import tkinter as tk
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
NavigationToolbar2Tk)
from matplotlib.backend_bases import key_press_handler
from matplotlib.figure import Figure
from matplotlib import pyplot as plt
import matplotlib.dates as mdates



WEATHER_URL = "https://danepubliczne.imgw.pl/api/data/synop/id/12566"
WEATHER_DATA_FILE_NAME = "saved_data.json"



# prosta konwersja daty i godziny podanej w tringach na datetime
def parseDate(date_string, hour_string):

    year_string = date_string[:4]
    month_string = date_string[5:7]
    day_string = date_string[8:10]

    if month_string[0] == 0:
        month_string = month_string [1]

    if day_string[0] == 0:
        day_string = day_string[1]

    if hour_string[0] == 0:
        hour_string = hour_string[1]

    return datetime.datetime(year=int(year_string),
month=int(month_string), day=int(day_string), hour=int(hour_string))


# pobieram dane z serwera, jesli pomiar jest juz w liscie to go ignoruje,
jesli nie to dodaje
def updateWeatherData(weather_data):

    response = requests.get(WEATHER_URL)
    parsed_response = json.loads(response.text)

    if weather_data == []:

        weather_data.append(parsed_response)
        return

    last_saved_date = parseDate(weather_data[-1]["data_pomiaru"],
weather_data[-1]["godzina_pomiaru"])
    received_date = parseDate(parsed_response["data_pomiaru"],
parsed_response["godzina_pomiaru"])

    if received_date > last_saved_date:
        weather_data.append(parsed_response)
```

```
# zwracam tablice dictow z pomiarami pobranymi z pliku
def getWeatherDataFromFile():

    if os.path.exists(WEATHER_DATA_FILE_NAME) == False:
        return []

    with open(WEATHER_DATA_FILE_NAME, "r") as f:

        stringified_data = f.read()
        parsed_data = json.loads(stringified_data)

        return parsed_data



# tablica dictow zapisywana w pliku
def saveWeatherDataToFile(weather_data):

    with open(WEATHER_DATA_FILE_NAME, "w") as f:

        stringified_data = json.dumps(weather_data)
        f.write(stringified_data)



# pobieram dane dla osi x wykresow (daty z godzinami) z listy pomiarow
def getTimeStampsFromWeatherData(weather_data):

    stamps = []
    for datum in weather_data:

        stamp = parseDate(datum['data_pomiaru'], datum['godzina_pomiaru'])
        stamps.append(stamp)

    return stamps



# 4 funkcje pobierajace wartosci pomiarow danego parametru z listy z
pomiarami
def getTemperaturesFromWeatherData(weather_data):

    temps = []
    for datum in weather_data:

        temp = float(datum["temperatura"])
        temps.append(temp)

    return temps

def getPressureFromWeatherData(weather_data):

    pressure_data = []
    for datum in weather_data:

        pressure = float(datum["cisnienie"])
        pressure_data.append(pressure)
```

```
    return pressure_data

def getMoistureFromWeatherData(weather_data):

    moisture_data = []
    for datum in weather_data:

        moisture = float(datum["wilgotnosc_wzgledna"])
        moisture_data.append(moisture)

    return moisture_data

def getSumedRainFromWeatherData(weather_data):

    rain_data = []
    for datum in weather_data:

        _sum = float(datum["suma_opadu"])
        rain_data.append(_sum)

    return rain_data



# klasa tozsama z aplikacja i jej layoutem / abstrakcyjna funkcjonalnoscia
class weatherApp(tk.Frame):

    def __init__(self, master=None):

        super().__init__(master)
        self.master = master
        self.pack()
        self.create_widgets()
        self.current_data_set = "temperature"
        self.weather_data = getWeatherDataFromFile()

    # inicjalizacja elementow aplikacji (controls)
    def create_widgets(self):

        # buttons
        self.update_data_button = tk.Button(self)
        self.update_data_button["text"] = "update data"
        self.update_data_button["command"] = self.updateData
        self.update_data_button.grid(row = 5, column = 1)

        self.temp_graph_button = tk.Button(self)
        self.temp_graph_button["text"] = "temperatures"
        self.temp_graph_button["command"] = self.presentTemperatures
        self.temp_graph_button.grid(row = 5, column = 2)

        self.pressure_graph_button = tk.Button(self)
        self.pressure_graph_button["text"] = "pressure"
        self.pressure_graph_button["command"] = self.presentPresure
        self.pressure_graph_button.grid(row = 5, column = 3)

        self.moisture_graph_button = tk.Button(self)
        self.moisture_graph_button["text"] = "moisture"
        self.moisture_graph_button["command"] = self.presentMoisture
        self.moisture_graph_button.grid(row = 5, column = 4)

        self.rain_graph_button = tk.Button(self)
```

```
        self.rain_graph_button["text"] = "rain sum"
        self.rain_graph_button["command"] = self.presentRainSum
        self.rain_graph_button.grid(row = 5, column = 5)

        # labels
        self.graph_title_label = tk.Label(self, text = "please choose the
type of data")
        self.graph_title_label.configure(anchor="center")
        self.graph_title_label.grid(row = 1, column = 1, columnspan = 5)

    # aktualizacja stanu aplikacji zwiazanego z przechowywaniem listy
pomiarow, dodatkowo aktualizacja bierzacego wykresu
    def updateData(self):

        updateWeatherData(self.weather_data)
        saveWeatherDataToFile(self.weather_data)
        self.graphCurrentData()

    # okreslenie bierzacego wykresu
    def graphCurrentData(self):

        if self.current_data_set == "temperatures":
            self.graphTemperatures()

        elif self.current_data_set == "pressure":
            self.graphPresure()

        elif self.current_data_set == "rain_sum":
            self.graphRainSum()

        elif self.current_data_set == "moisture":
            self.graphMoisture()

    #  odswiezenie tego co jest aktualnie narysowane
    def presentTemperatures(self):

        self.current_data_set = "temperatures"
        self.graphCurrentData()

    def presentRainSum(self):

        self.current_data_set = "rain_sum"
        self.graphCurrentData()

    def presentPresure(self):

        self.current_data_set = "pressure"
        self.graphCurrentData()

    def presentMoisture(self):

        self.current_data_set = "moisture"
        self.graphCurrentData()

    # 4 funkcje rysujace wykresy poszczegolnych parametrow
    def graphTemperatures(self):

        self.graph_title_label["text"] = "temperatures"
        x = getTimeStampsFromWeatherData(self.weather_data)
        y = getTemperaturesFromWeatherData(self.weather_data)
        fig = Figure(figsize=(5, 4))
```

```
        axes = fig.add_subplot(111)
        axes.plot(x, y)
        axes.set_ylabel("C")
        axes.set_xticks([x[0], x[-1]])

        self.canvas = FigureCanvasTkAgg(fig, master=self)
        self.canvas.draw()
        self.canvas.get_tk_widget().grid(row=2, column = 1, columnspan = 5)

    def graphPresure(self):

        self.graph_title_label["text"] = "pressure"
        x = getTimeStampsFromWeatherData(self.weather_data)
        y = getPressureFromWeatherData(self.weather_data)
        fig = Figure(figsize=(5, 4))
        axes = fig.add_subplot(111)
        axes.plot(x, y)
        axes.set_ylabel("hPa")
        axes.set_xticks([x[0], x[-1]])

        self.canvas = FigureCanvasTkAgg(fig, master=self)
        self.canvas.draw()
        self.canvas.get_tk_widget().grid(row=2, column = 1, columnspan = 5)

    def graphMoisture(self):

        self.graph_title_label["text"] = "moisture"
        x = getTimeStampsFromWeatherData(self.weather_data)
        y = getMoistureFromWeatherData(self.weather_data)
        fig = Figure(figsize=(5, 4))
        axes = fig.add_subplot(111)
        axes.plot(x, y)
        axes.set_ylabel("%")
        axes.set_xticks([x[0], x[-1]])

        self.canvas = FigureCanvasTkAgg(fig, master=self)
        self.canvas.draw()
        self.canvas.get_tk_widget().grid(row=2, column = 1, columnspan = 5)

    def graphRainSum(self):

        self.graph_title_label["text"] = "rain sum"
        x = getTimeStampsFromWeatherData(self.weather_data)
        y = getSumedRainFromWeatherData(self.weather_data)
        fig = Figure(figsize=(5, 4))
        axes = fig.add_subplot(111)
        axes.plot(x, y)
        axes.set_ylabel("cm")
        axes.set_xticks([x[0], x[-1]])

        self.canvas = FigureCanvasTkAgg(fig, master=self)
        self.canvas.draw()
        self.canvas.get_tk_widget().grid(row=2, column = 1, columnspan = 5)



# stworzenie okna i inicjalizacja klasy aplikacji
if __name__ == "__main__":

    root = tk.Tk()
    root.minsize(510, 460)
```

```
    root.maxsize(510, 460)
    root.winfo_toplevel().title("weather app")
    win = weatherApp(root)
    win.mainloop()
```