



WYDZIAŁ AUTOMATYKI, ELEKTRONIKI I INFORMATYKI

Projekt

Przemysłowe bazy danych

Temat:

Akwizycja danych meteorologicznych do bazy danych i ich przedstawienie.

Projekt wykonął:

Piotr Malec

Szymon Stolarek

1. Cel projektu

Celem projektu „Akwizycja danych meteorologicznych do bazy danych i ich przedstawienie” jest opracowanie aplikacji umożliwiającej pozyskiwanie aktualnych danych pogodowych z zewnętrznego API (OpenWeatherMap), zapis ich do bazy danych oraz wizualizacja zgromadzonych informacji w czytelny sposób. Projekt skupia się na integracji danych zewnętrznych z systemem bazodanowym, zapewnieniu ich przechowywania i późniejszego przetwarzania oraz prezentacji wyników za pomocą odpowiednich narzędzi graficznych. Całą aplikację napisano w języku Python.

2. OpenWeatherMap

OpenWeatherMap to popularna platforma, która oferuje dostęp do różnych danych pogodowych w czasie rzeczywistym, prognoz, a także historycznych informacji o pogodzie. Jest niezbędna do pozyskiwania szczegółowych informacji o warunkach atmosferycznych w dowolnym miejscu na świecie. Wykorzystanie tej platformy jest niezwykle wygodne, ale niesie też pewne ograniczenia, ponieważ w darmowej wersji pozwala tylko na pobieranie prognozy pogody na 5 dni w odstępach 3 godzin. Link do strony: <https://openweathermap.org>. Zdecydowano się na wykorzystanie takiego API ze względu na łatwość integracji z wszelkiego rodzaju aplikacjami.

3. Program weather.py – pobieranie danych z API OpenWeatherMap

Do zbierania danych pogodowych wykorzystywane jest API OpenWeatherMap. W dostępnej wersji API użytkownik musi podać współrzędne geograficzne (szerokość i długość geograficzną), aby określić lokalizację, z której mają być pobrane dane pogodowe. Dzięki temu możliwe jest uzyskanie szczegółowych informacji o pogodzie w konkretnym miejscu na świecie. API pozwala również na skonfigurowanie jednostek miar, w jakich mają być zwracane dane (np. temperatura w stopniach Celsjusza lub Fahrenheit), a także na wybór formatu danych, najczęściej w formacie JSON, który jest łatwy do przetwarzania w aplikacjach.

API call

```
api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}
&appid={API key}
```



Parameters

lat	required	Latitude. If you need the geocoder to automatic convert city names and zip-codes to geo coordinates and the other way around, please use our Geocoding API
lon	required	Longitude. If you need the geocoder to automatic convert city names and zip-codes to geo coordinates and the other way around, please use our Geocoding API
appid	required	Your unique API key (you can always find it on your account page under the " API key " tab)
units	optional	Units of measurement. <code>standard</code> , <code>metric</code> and <code>imperial</code> units are available. If you do not use the <code>units</code> parameter, <code>standard</code> units will be applied by default. Learn more
mode	optional	Response format. JSON format is used by default. To get data in XML format use <code>mode=xml</code> . Learn more

W celu łatwej integracji zbierania danych pogodowych z resztą aplikacji zdecydowano się stworzyć w tym celu klasę. Jak widać poniżej funkcja do odczytywania pogody składa się z dwóch zapytań do API. Pierwsze odpowiedzialne jest za pobranie długości oraz szerokości geograficznej wybranego miasta. Następnie wykonywane jest już właściwe zapytanie o dane pogodowe.

```
class Weather:
    def read_weather(self, city):
        logger.info(f"Rozpaczynam pobieranie pogody dla miasta: {city}")
        try:
            url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}"
            req = requests.get(url)
            req.raise_for_status()
            data = req.json()

            name = data["name"]
            lon = data["coord"]["lon"]
            lat = data["coord"]["lat"]
            logger.info(
                f"Znaleziono miasto: {name}, długość geograficzna: {lon}, szerokość geograficzna: {lat}"
            )

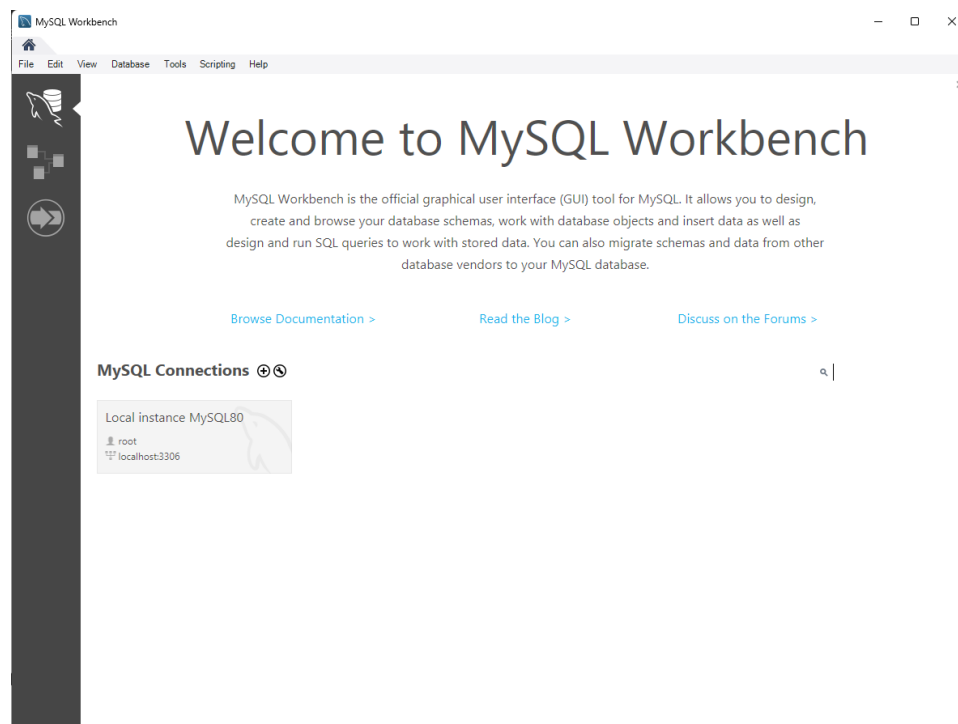
            forecast_url = f"https://api.openweathermap.org/data/2.5/forecast?lat={lat}&lon={lon}&cnt=40&units=metric&appid={API_KEY}"
            req = requests.get(forecast_url)
            req.raise_for_status()
            forecast_data = req.json()

            logger.info("Pogoda została pomyślnie pobrana.")
            return forecast_data

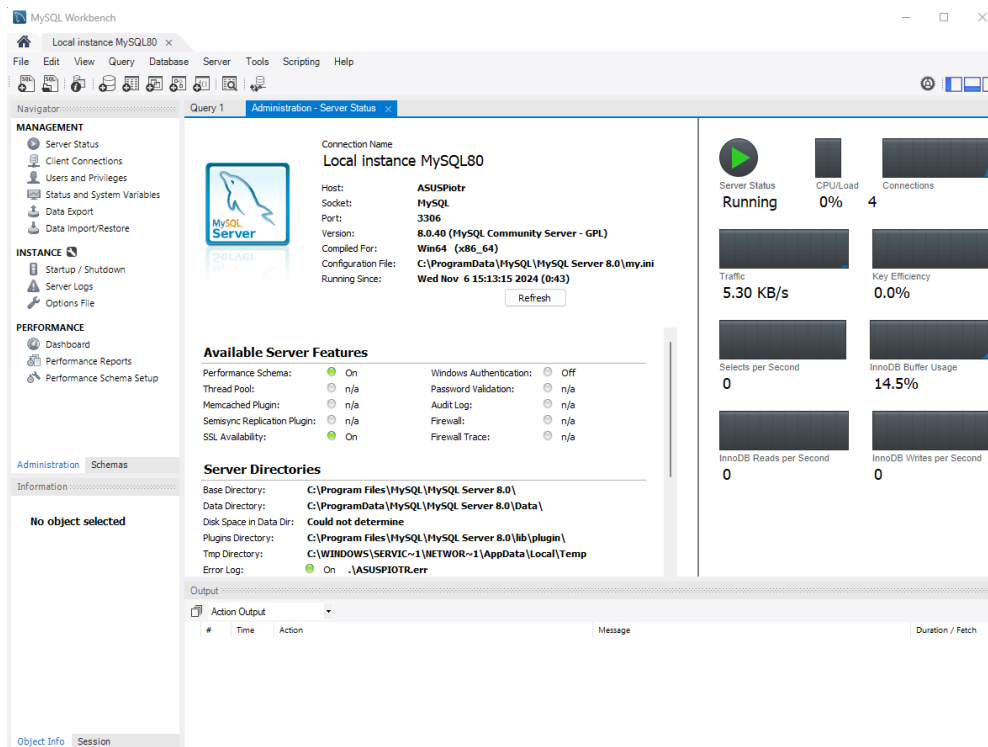
        except requests.exceptions.HTTPError as http_err:
            logger.error(f"Wystąpił błąd HTTP: {http_err}")
        except Exception as err:
            logger.error(f"Wystąpił błąd: {err}")
```

4. Baza danych – MySQL

W projekcie baza danych została zrealizowana przy użyciu systemu zarządzania bazą danych MySQL, który zapewnia niezawodne i wydajne środowisko do przechowywania oraz zarządzania danymi. MySQL, będący jednym z najpopularniejszych systemów relacyjnych baz danych, oferuje bogaty zestaw funkcji umożliwiających składowanie i manipulację danymi w sposób zorganizowany i bezpieczny. W kontekście projektu baza danych została zaprojektowana tak, aby przechowywać dane meteorologiczne, z tabelami zawierającymi informacje o pomiarach pogodowych, z polami na datę, godzinę oraz parametry pogodowe, takie jak temperatura, wilgotność i ciśnienie. Zostały zastosowane odpowiednie indeksy, aby zoptymalizować zapytania dotyczące wyszukiwania i filtrowania danych. Pierwsza komunikacja z bazą danych odbywa się za pomocą programu MySQL Workbench.



Jest to zintegrowane środowisko umożliwiające projektowanie, zarządzanie oraz administrowanie bazami danych w sposób intuicyjny i przyjazny dla użytkownika. Funkcje administracyjne w MySQL Workbench obejmują monitorowanie wydajności serwera, zarządzanie użytkownikami i ich uprawnieniami oraz wykonywanie kopii zapasowych i przywracania danych.



	date	temperature	perceived_temp	humidity	pressure
▶	2024-10-29 13:00:00	11.83	11.47	92	1024
	2024-10-29 16:00:00	12.28	11.94	91	1024
	2024-10-29 19:00:00	12.41	12.16	94	1024
	2024-10-29 22:00:00	12.18	11.86	92	1023
	2024-10-30 01:00:00	12.21	11.84	90	1023
	2024-10-30 04:00:00	11.92	11.54	91	1022
	2024-10-30 07:00:00	12.01	11.67	92	1022
	2024-10-30 10:00:00	13.07	12.68	86	1022
	2024-10-30 13:00:00	16.23	15.48	60	1021
	2024-10-30 16:00:00	14.08	13.53	76	1021
	2024-10-30 19:00:00	12.33	11.89	87	1022
	2024-10-30 22:00:00	12.98	12.24	73	1022
	2024-10-31 01:00:00	10.66	9.82	78	1023
	2024-10-31 04:00:00	8.94	6.56	78	1023
	2024-10-31 07:00:00	9.28	7.29	71	1025
	2024-10-31 10:00:00	10.96	9.7	61	1025
	2024-10-31 13:00:00	12.14	10.85	55	1024

Baza danych MySQL jest powszechnie używanym narzędziem do celów przemysłowych, ale i także prywatnych. Jest przejrzysta wizualnie i posiada bogaty zestaw funkcji.

5. Program database.py – zarządzanie bazą danych

Klasa Database została stworzona w celu zarządzania połączeniem z bazą danych MySQL oraz manipulowaniem danymi w tej bazie. Klasa ta jest odpowiedzialna za wszystkie operacje związane z bazą danych, takie jak wstawianie rekordów, pobieranie danych oraz bezpieczne zamykanie połączenia. Jedną z podstawowych funkcji w klasie

jest `_connect`, która umożliwi połączenie z wcześniej utworzoną bazą danych. Poprawne wykonanie tej operacji jest niezbędne do dalszej pracy z bazą.

```
class Database:
    def __init__(self):
        self._connect()

    def _connect(self):
        try:
            self.connection = mysql.connector.connect(
                host="localhost", user="root", password="1234", database="weather"
            )
            self.cursor = self.connection.cursor()
            logger.info("Połączenie z bazą danych nawiązane.")
        except mysql.connector.Error as err:
            logger.error(f"Coś poszło nie tak: {err}")
```

Do najistotniejszych funkcji klasy `Database` należą również te odpowiedzialne za manipulację danymi, takie jak wstawianie rekordów (dodawanie nowych danych do tabel), pobieranie rekordów (wyciąganie danych z bazy w odpowiednich formatach) oraz zamykanie połączenia z bazą. Ważnym aspektem jest szczególna dbałość o poprawne zamknięcie połączenia, ponieważ niezamknięcie bazy może prowadzić do problemów z integralnością danych i ich utratą. Poprawne zarządzanie połączeniem z bazą danych oraz operacjami na danych pozwala na stabilną i bezpieczną pracę aplikacji, zapewniając zarówno wydajność, jak i niezawodność w przechowywaniu oraz manipulowaniu danymi.

```
database.py > Database > create_database
You, 2 tygodnie temu | 2 authors (Szymon and one other)
28 class Database:
29 >     def __init__(self): ...
31
32 >     def _connect(self): ...
41
42 >     def show_databases(self): ...
51
52 >     def show_tables(self): ...
61
62 >     def create_database(self, name_db): Szymo
70
71 >     def create_table(self, name_tb): ...
88
89 >     def get_tables(self): ...
102
103 >     def delate_table(self, table): ...
109
110 >     def delete_all_records(self, table_name): ...
124
125 >     def insert_record( ...
141
142 >     def fetch_records(self, table_name): ...
157
158 >     def close(self): ...
165
```

6. Program controller.py – zarządzanie aplikacją

Klasa Controller jest odpowiedzialna za integrację z API OpenWeatherMap oraz zarządzanie komunikacją z bazą danych. Jej głównym celem jest oddzielenie logiki manipulacji danymi od części aplikacji odpowiedzialnej za ich wizualizację. Dzięki temu możliwe jest utrzymanie czystej architektury, w której warstwa kontrolera obsługuje wszystkie operacje związane z pobieraniem danych pogodowych, ich przechowywaniem w bazie oraz przygotowywaniem odpowiednich odpowiedzi do wyświetlenia w interfejsie użytkownika. Klasa ta zapewnia, że dane z OpenWeatherMap są odpowiednio przetwarzane i zapisane w bazie danych, a następnie udostępniane aplikacji w sposób wydajny i uporządkowany. Taki podział umożliwia łatwą rozbudowę aplikacji oraz jej utrzymanie, zapewniając jednocześnie skalowalność i elastyczność.

```
class Controller:
    def __init__(self):
        self.database = Database()
        self.weather = Weather()

    def process(self, city):
        cityTable = self.database.get_tables()

        if city in cityTable:
            existing_records = self.database.fetch_records(city)
            existing_dates = {record[0] for record in existing_records}

        else:
            self.database.create_table(city)
            existing_dates = set()

        data = self.weather.read_weather(city)

        if data and "list" in data:
            for entry in data["list"]:
                timestamp = entry["dt"]
                temp = entry["main"]["temp"]
                perceived_temp = entry["main"].get("feels_like")
                humidity = entry["main"]["humidity"]
                pressure = entry["main"]["pressure"]
                date_time = datetime.fromtimestamp(timestamp).strftime(
                    "%Y-%m-%d %H:%M:%S"
                )

                if date_time not in existing_dates:
                    self.database.insert_record(
                        city, date_time, temp, perceived_temp, humidity, pressure
                    )
                    logger.info(
                        f"Dane pogodowe dla {city} w dniu {date_time} zostały zapisane w bazie."
                    )
                else:
                    logger.info(
                        f"Dane dla {city} w dniu {date_time} już istnieją, pomijam zapis."
                    )

        weather = self.database.fetch_records(city)
        return weather

    def close(self):
        self.database.close()


    def fetch_records(self, city_name):
        records = self.database.fetch_records(city_name)
        return records
```

7. Program gui.py – wizualizacja wyników

Program został stworzony przy użyciu biblioteki Tkinter, będącej standardowym narzędziem do budowania interfejsów graficznych w Pythonie. Tkinter pozwala na łatwe tworzenie okien aplikacji, formularzy, przycisków, menu rozwijanych oraz innych elementów interfejsu, które umożliwiają użytkownikom interakcję z aplikacją.

```
gui.py > Gui > go_back
...
1  import tkinter as tk
2  import tkinter.font as tkFont
3  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
4  import matplotlib.pyplot as plt
5  import numpy as np
6  from controller import Controller
7  from PIL import Image, ImageTk
8  from datetime import datetime, timedelta
9
10
11  ...
12  class Gui:
13  >   def __init__(self, root): ...
26
27  >   def create_widgets(self): ...
75
76  >   def create_weather_plot(self, city_name): ...
165
166  >   def on_closing(self): ...
172
173  >   def open_second_window(self): ...
297
298  >   def go_back(self): | Szymon, 3 tygodnie temu • Deleted dropd
301
302  if __name__ == "__main__":
303     root = tk.Tk()
304     app = Gui(root)
305     root.mainloop()
306
```

Do wizualizacji danych w postaci wykresów wykorzystano bibliotekę matplotlib. Dzięki integracji matplotlib z Tkinter poprzez moduł FigureCanvasTkAgg możliwe jest wyświetlanie wykresów bezpośrednio w oknie aplikacji. Matplotlib zapewnia wszechstronne narzędzia do tworzenia wykresów liniowych, słupkowych i innych typów wizualizacji, co umożliwia użytkownikowi zobaczenie szczegółowych danych pogodowych, takich jak temperatura, wilgotność, ciśnienie, w różnych ujęciach czasowych.



Please enter the city name:

Select a day:

Check

Program uruchamia główne okno aplikacji, w którym użytkownik jest proszony o wprowadzenie nazwy miasta i wybór dnia z rozwijanego menu. Po wprowadzeniu danych i naciśnięciu przycisku, aplikacja przelatczy się do nowego okna, które zawiera wykresy z informacjami o pogodzie dla wybranego miasta i dnia. Wykresy prezentują wartości takie jak temperatura, temperatura odczuwalna, wilgotność i ciśnienie. W tym oknie istnieje także możliwość wyświetlenia pogody z przeszłości, które poprzednio zapisano w bazie danych, aby zwiększyć wagę bazy danych w ogóle projektu.



Ponadto program umożliwia użytkownikowi zmianę skali wykresów za pomocą pól tekstowych, co pozwala na dostosowanie zakresu osi do preferencji użytkownika. Po zmianie skali użytkownik może odświeżyć wizualizację, aby zobaczyć dane z nowymi ustawieniami. Do obsługi obrazów, np. logotypu aplikacji, wykorzystano bibliotekę PIL (Pillow), która umożliwia ładowanie i manipulację obrazami. Program jest zaprojektowany tak, aby przełączać między oknami za pomocą funkcji sterujących, co zwiększa interaktywność i umożliwia powrót do głównego okna po zakończeniu pracy z wykresami.

8. Wnioski

- **Skuteczna integracja technologii:**

Projekt udowodnił, że połączenie API OpenWeatherMap, systemu zarządzania bazą danych MySQL oraz narzędzi do wizualizacji danych, takich jak Tkinter i Matplotlib, może być efektywnym rozwiązaniem dla zbierania, przechowywania i prezentowania danych meteorologicznych. Dzięki temu możliwe było stworzenie kompletnej aplikacji, która łączy funkcje akwizycji danych, ich przechowywania oraz wizualizacji.

- **Modularność i skalowalność:**

Podział aplikacji na moduły, takie jak weather.py do pobierania danych, database.py do zarządzania bazą danych, controller.py jako warstwa pośrednicząca oraz gui.py odpowiedzialny za interfejs użytkownika, pozwolił na zachowanie czystości architektury oraz ułatwił przyszłą rozbudowę. Takie podejście sprzyja łatwemu utrzymaniu i aktualizacji aplikacji.

- **Zastosowanie nowoczesnych technologii bazodanowych:**

MySQL okazał się solidnym wyborem do przechowywania danych meteorologicznych. Jego niezawodność oraz szeroka gama funkcji umożliwiły przechowywanie danych w uporządkowanej i łatwo dostępnej formie. Dodatkowo, wykorzystanie narzędzi takich jak MySQL Workbench pomogło w zarządzaniu strukturą bazy oraz optymalizacji zapytań.

- **Intuicyjna obsługa użytkownika:**

Zastosowanie biblioteki Tkinter pozwoliło na stworzenie prostego i przejrzystego interfejsu graficznego, co ułatwiło interakcję użytkownika z aplikacją. Wprowadzenie opcji wyboru dnia i możliwości dostosowywania skali wykresów sprawiło, że aplikacja jest bardziej elastyczna i użyteczna dla użytkowników o różnych potrzebach.

- **Wielozadaniowość aplikacji:**

Dzięki zastosowaniu kontrolera integrującego komunikację z bazą danych oraz pobieranie danych z API, aplikacja może pobierać nowe dane i wyświetlać starsze zapisy z bazy. To podejście zwiększa użyteczność systemu i pozwala na analizę historycznych danych pogodowych.