Przetwarzanie rozproszone

Autor: mgr inż. Szymon Guzik

Kontakt:

email: szymon.guzik@gdansk.merito.pl

Uwaga

Wiadomości email sprawdzam raz w tygodniu – po wysłaniu wiadomości proszę przez tydzień jej nie ponawiać. W przypadku, kiedy nie odpowiem proszę o ponowne przesłanie wysłanie oraz wiadomość na MSTeams (prywatną wiadomość)

lub

Microsoft Teams

Zasady zaliczenia przedmiotu:

- Zaliczenie laboratorium odbywa się z wykorzystaniem platformy Moodle
- Na każdych laboratoriach wykonywane jest zadanie
- Z każdego ćwiczenia można zdobyć maksymalnie 100 pkt.
- Obecność na laboratoriach jest obowiązkowa
- Dodanie pliku z rozwiązaniem zadania z laboratorium (2 tygodnie od laboratorium z zadaniem do wykonania)
- Przesłane pliki będą sprawdzane w miarę na bieżąco - po sprawdzeniu od razu otrzymacie pkt za przesłane zadanie
- Ocena:
    - 150 pkt - 3.0
    - 180 pkt - 3.5
    - 210 pkt - 4.0
    - 240 pkt - 4.5
    - 270 pkt - 5.0

Zagadnienia:

1. **Wprowadzenie do Przetwarzania Rozproszonego**
    a. Definicja przetwarzania rozproszonego
    b. Cele i korzyści z przetwarzania rozproszonego
    c. Architektura systemów rozproszonych
    d. Przykłady aplikacji rozproszonych (Zadanie nr. 1) - 15pkt.
    e. Pobieranie danych z API (Zadanie nr.2) - 35pkt.
    f. Zapisanie danych w bazie (Zadanie nr.3) - 50 pkt.
2. **Komunikacja w systemach rozproszonych**
    a. Protokoły komunikacyjne (TCP/IP, HTTP, RPC)
    b. Model klient-serwer
    c. Mechanizmy przesyłania danych
    d. Synchroniczna i asynchroniczna komunikacja
3. **Zarządzanie zasobami w systemach rozproszonych**
    a. Zarządzanie pamięcią
    b. Zarządzanie procesami
    c. Zarządzanie danymi
    d. Skalowalność i równoważenie obciążenia
4. **Bezpieczeństwo w przetwarzaniu rozproszonym**
    a. Zagrożenia w systemach rozproszonych
    b. Autoryzacja i uwierzytelnianie
    c. Kryptografia w komunikacji
    d. Audyt i monitorowanie
5. **Protokoły i narzędzia do przetwarzania rozproszonego**
    a. Protokoły komunikacyjne w szczegółach
    b. Narzędzia do tworzenia aplikacji rozproszonych (np. Apache Kafka, RabbitMQ)
    c. Rozwiązywanie problemów wydajnościowych
6. **Rozproszone obliczenia**
    a. Modele programowania rozproszonego (MapReduce, MPI)
    b. Przykłady zastosowań rozproszonych obliczeń
    c. Frameworki do rozproszonych obliczeń (np. Apache Hadoop)
7. **Przetwarzanie strumieniowe**
    a. Przetwarzanie strumieniowe vs. przetwarzanie wsadowe
    b. Przykłady zastosowań przetwarzania strumieniowego
    c. Technologie do przetwarzania strumieniowego (np. Apache Kafka Streams, Apache Flink)
8. **Bazy danych rozproszone**
    a. Modele baz danych rozproszonych
    b. Konsystencja i dostępność w bazach danych rozproszonych
    c. Przykłady baz danych rozproszonych (np. Cassandra, MongoDB)

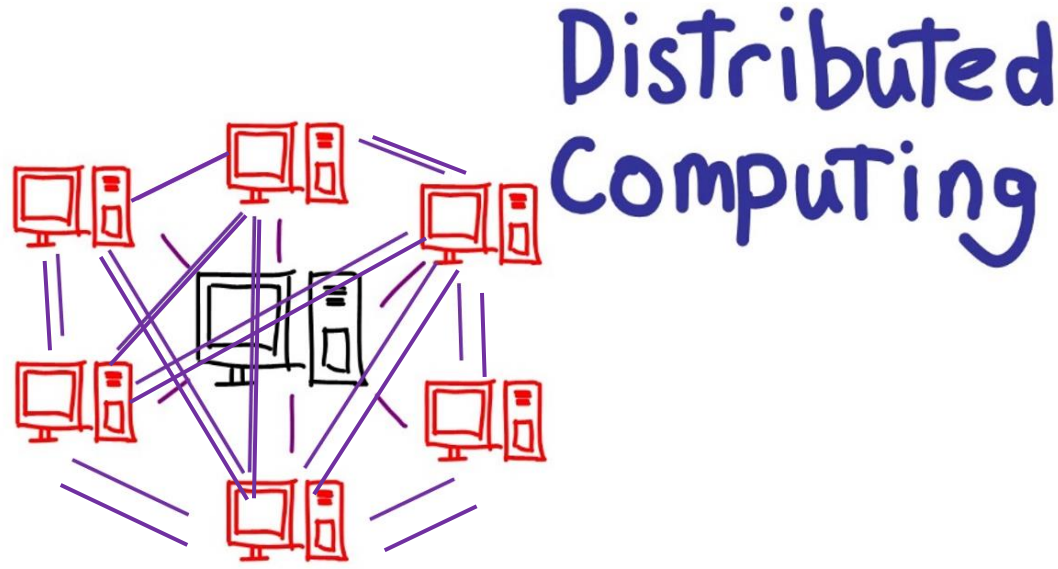
9. **Mikroserwisy i architektura oparta na usługach**
    a. Mikroserwisy vs. monolityczne architektury
    b. Projektowanie mikroserwisów
    c. Rozwiązywanie problemów w architekturze opartej na usługach
    d. Narzędzia do zarządzania mikroserwisami (np. Kubernetes, Docker

10. **Praktyczne zastosowania przetwarzania rozproszonego**
    a. Studium przypadku: Tworzenie systemu rozproszonego od podstaw
    b. Omówienie projektów i zastosowań przetwarzania rozproszonego w rzeczywistych scenariuszach

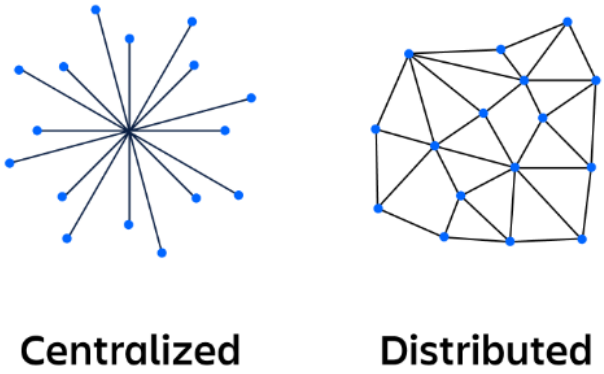
Wprowadzenie do Przetwarzania Rozproszonego

Definicja przetwarzania rozproszonego

Przetwarzanie rozproszone to paradygmat obliczeniowy, w którym zadania przetwarzania danych lub obliczeniowe są wykonywane przez wiele komputerów lub procesorów, które są ze sobą połączone i współpracują w celu realizacji konkretnego celu. Główną różnicą między przetwarzaniem rozproszonym a przetwarzaniem centralnym (lub jednoprocesorowym) jest to, że w przetwarzaniu rozproszonym zadania mogą być wykonywane równolegle na wielu maszynach, co prowadzi do poprawy wydajności i dostępności systemu



[ź



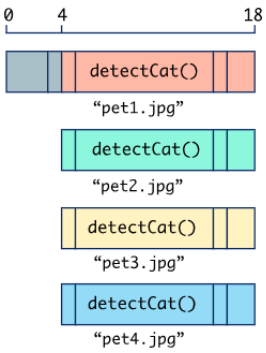
Wprowadzenie do Przetwarzania Rozproszonego

Cele i korzyści z przetwarzania rozproszonego

- **Poprawa wydajności:** Dzięki rozproszeniu obciążenia zadania mogą być wykonywane równolegle, co przyspiesza przetwarzanie danych.
- **Dostępność:** W przypadku awarii jednego komponentu, system może nadal działać, ponieważ inne komponenty mogą przejąć zadania.
- **Skalowalność:** Systemy rozproszone łatwiej jest skalować w górę (dodawanie nowych zasobów) i w dół (zmniejszanie zasobów) w zależności od potrzeb.
- **Niezawodność:** Redundancja i rozproszenie danych i zasobów pomagają w utrzymaniu niezawodności systemu.
- **Lepsza wydajność przy dużych obciążeniach:** Systemy rozproszone są bardziej wydajne w obsłudze dużych obciążeń danych i użytkowników.

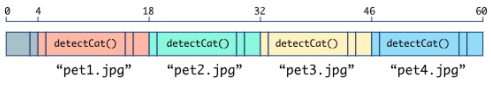
Obliczenia równoległe

Poniższa linia czasu obrazuje operacje komputera, który dzieli obliczenia programu na 4 niezależne procesy:

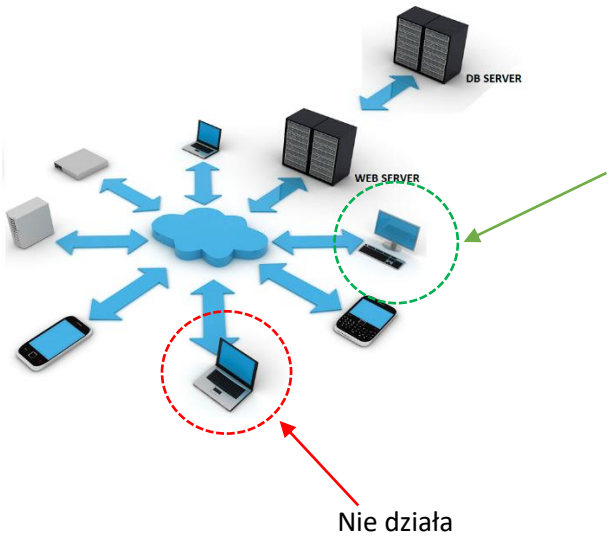


Obliczenia sekwencyjne

Poniższa linia czasu obrazuje operacje komputera w czasie:



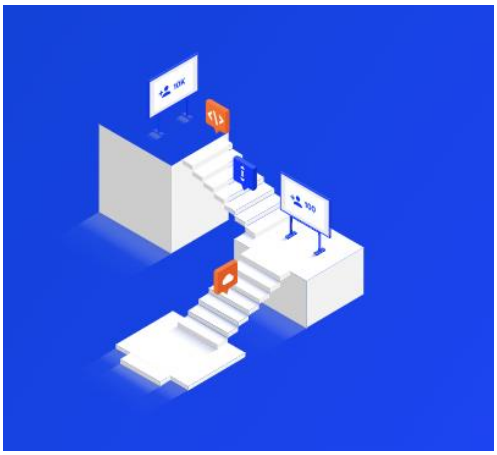
Dostępność:



Przejmuje role uszkodzonej maszyny

(komputera zaznaczonego na czerwono)

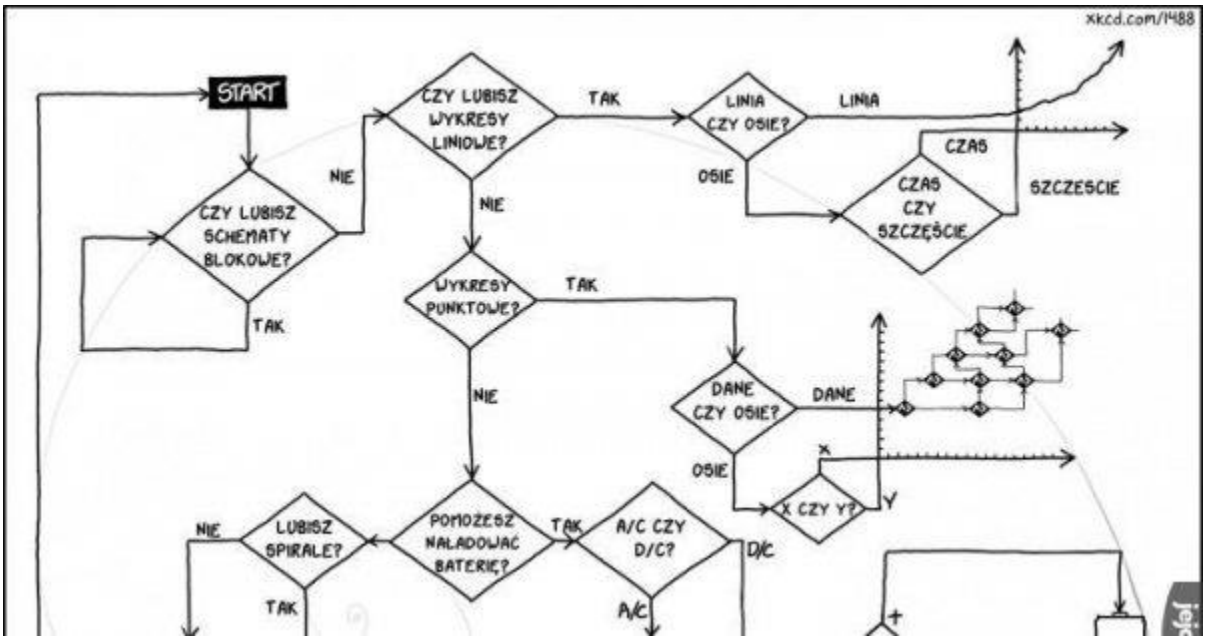
Skalowalność:



Rozwijanie aplikacji we właściwy sposób i zapewnienie możliwości skalowania pozwala przygotować projekt na rosnącą liczbę użytkowników.

Kiedy warto rozważyć skalowanie aplikacji i o czym pamiętać?

Niezawodność: (dobrze rozpisane przypadki, wyjatki itd.)

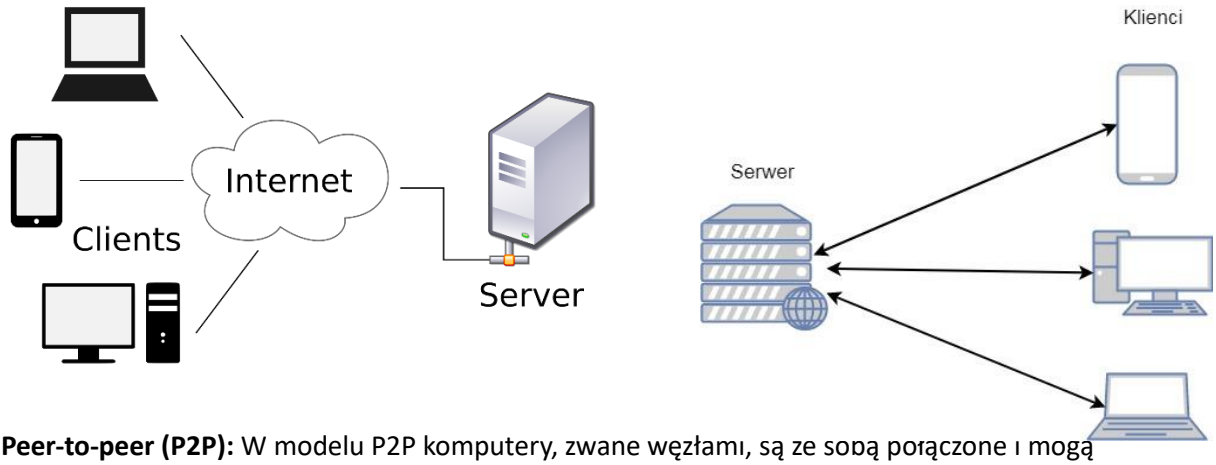


Lepsza wydajność przy dużych obciążeniach (podział na kilka maszyn zwiększa wydajność pracy)

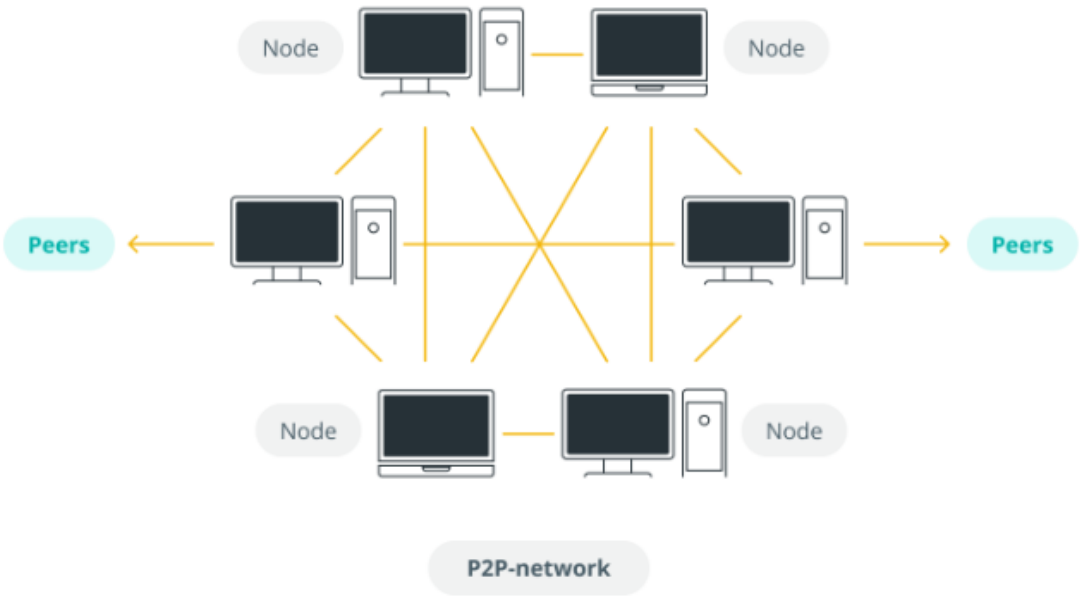
Wprowadzenie do Przetwarzania Rozproszonego

Architektura systemów rozproszonych

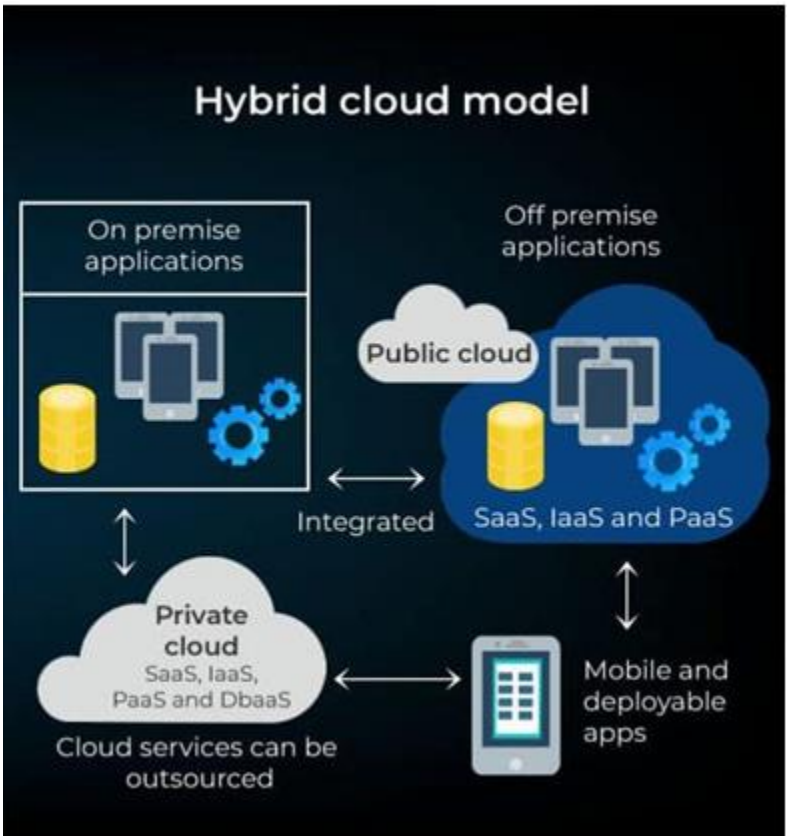
- **Klient-serwer:** W tym modelu, klient wysyła żądania do serwera, który odpowiada na te żądania. Serwer jest odpowiedzialny za przetwarzanie i dostarczanie danych klientowi.



- **Peer-to-peer (P2P):** W modelu P2P komputery, zwane węzłami, są ze sobą połączone i mogą działać zarówno jako klienci, jak i serwery, dzieląc się zasobami i obowiązkami równorzędnie.



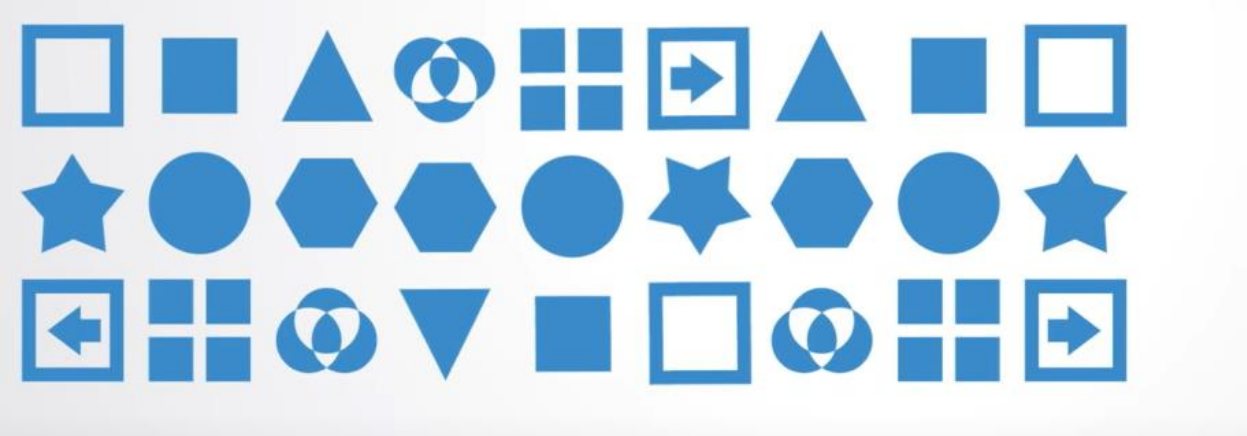
- **Hybrydowe i inne architektury:** Istnieją również hybrydowe modele i bardziej zaawansowane architektury, które łączą różne podejścia w zależności od konkretnych potrzeb systemu.



Wprowadzenie do Przetwarzania Rozproszonego

Przykłady aplikacji rozproszonych (Zadanie nr.1) – czas na zadanie 30 minut. (15pkt.)

Proszę wyszukać przykłady rozproszonych aplikacji (min. 5) – uzasadnić w kilku zdaniach
Plik z rozwiązaniem proszę umieścić na Moodle



Pobieranie danych z API (Zadanie nr.2) – czas na zadanie 1 godzina (35 pkt.)

Proszę wyszukać w zasobach sieci API (REST API), na którym będzie można zastosować metody

- GET
- POST
- PUT
- DELETE

Następnie napisać program z użyciem tych metod.

Poniżej znajduje się kod napisany w języku Python

Języki dozwolone (Python, Java, C#, PHP, JS)

Kod proszę spakować (kompresor ZIP) oraz przesłać na Moodle

Przykładowy kod.

```python
import requests

class WebClient:
    def __init__(self, base_url):
        self.base_url = base_url

    def get(self, endpoint, params=None):
        """Wykonuje żądanie GET na podanym endpointzie."""
        url = self.base_url + endpoint
        response = requests.get(url, params=params)
        if response.status_code == 200:
            return response.text
        else:
            raise Exception(f'Błąd {response.status_code}: {response.text}')

    def post(self, endpoint, data=None):
        """Wykonuje żądanie POST na podanym endpointzie."""
        url = self.base_url + endpoint
        response = requests.post(url, json=data)
        if response.status_code == 200:
            return response.text
        else:
            raise Exception(f'Błąd {response.status_code}: {response.text}')

    def put(self, endpoint, data=None):
        """Wykonuje żądanie PUT na podanym endpointzie."""
        url = self.base_url + endpoint
        response = requests.put(url, json=data)
        if response.status_code == 200:
            return response.text
        else:
            raise Exception(f'Błąd {response.status_code}: {response.text}')

    def delete(self, endpoint):
        """Wykonuje żądanie DELETE na podanym endpointzie."""
        url = self.base_url + endpoint
        response = requests.delete(url)
        if response.status_code == 200:
            return response.text
        else:
            raise Exception(f'Błąd {response.status_code}: {response.text}')

# Przykład użycia
if __name__ == "__main__":
    web_client = WebClient("https://example.com/api")

    try:
        response = web_client.get("/resource")
        print("Odpowiedź GET:", response)

        data_to_post = {"key": "value"}
        response = web_client.post("/resource", data_to_post)
        print("Odpowiedź POST:", response)

        data_to_put = {"updated_key": "updated_value"}
        response = web_client.put("/resource/1", data_to_put)
        print("Odpowiedź PUT:", response)

        response = web_client.delete("/resource/1")
        print("Odpowiedź DELETE:", response)
    except Exception as e:
        print("Wystąpił błąd:", str(e))
```

Zapisanie pobranych danych w bazie oraz stworzenie API z obsługą wymienionych metod z zadnia nr. 2 (Zadanie nr. 3) - Czas na zadanie do końca laboratoriów + praca w domu Termin oddania tego zadania : Od laboratoriów + 2 tygodnie. (50pkt.)

Zadanie polega na wykorzystaniu pozyskanych danych (zadanie nr. 2) oraz stworzeniu endpointów do obsługi „lokalnej bazy”. Pozyskane dane powinny zostać zapisane w lokalnej bazie.

Może być to baza typu SQLite

Link do dokumentacji bazy: SQLite Home Page