Zaawansowane Techniki Programowania
Analizator danych
Projekt

Wojciech Kur

Prowadzący:
mgr inż. Michał Gandor

9 stycznia 2021

Spis treści

1 Cel i zakres projektu

Celem projektu jest stworzenie aplikacji webowej, która udostępni API umożliwiające wysyłanie, pobieranie oraz agregację danych w określonym standardzie. W części frontendowej zaprezentowana zostanie przykładowa wizualizacja danych z możliwością ich zarządzania. Aplikacja będzie udostępniona w postaci skonteneryzowanej, co umożliwi jej sprawne wdrożenie na własnej maszynie. Projekt realizowany na ocenę 5.0 - jednak bez testów aplikacji.

2 Wymagania

Aplikacja będzie zawierać:

- Część backendową (agregator danych) - jako odrębne API
- Część frontendową - pozwalającą na wyświetlanie danych w postaci wykresów
- Zautomatyzowany deployment - poprzez konteneryzację oraz instalacyjne skrypty bashowe

2.1 Backend

Backend będzie możliwie modularny, do wykorzystania przez zewnętrzne aplikacje. Będzie pewnego rodzaju biblioteką w formie gotowego API, którą można podpiąć do każdego projektu - bądź wykorzystać globalnie do procesowania danych.

Serwisy aplikacyjne będą zawierać CRUD dla ORM, agregację danych do określonego formatu oraz bibliotekę przetwarzania danych dla pewnego utworzonego standardu. Standard będzie zaprojektowany dosyć generycznie by zrozumieć z czego wynika stan, czy zmienność danych - co przekłada się na szybsze zrozumienie swoich błędów i optymalizację wyników, np. w algorytmie. Główne endpointy aplikacji to:

- Uwierzytelnianie za pomocą tokena JWT
- Zapisanie danych (format json) (ustandaryzowany input)
- Wczytanie danych (format json) (ustandaryzowany output - z agregatami)
- Utworzenie zbioru danych
- Listowanie dostępnych zbiorów danych
- Usunięcie elementu/ów ze zbioru danych
- Dodanie elementu/ów do zbioru danych
- Usunięcie zbioru danych
- Edycja konfiguracji zbioru
- Edycja konta użytkownika (tylko dla klientów aplikacji frontendowej)

Niektóre z endpointów mogą zostać oczywiście połączone w celu usprawnienia procesu dla osób technicznych, bądź poprawienia "user experience". Dane będą przechowywane w relacyjnej bazie - co w pewien sposób ogranicza przechowywanie ich w stanie "surowym" (raw data).

2.2 Frontend

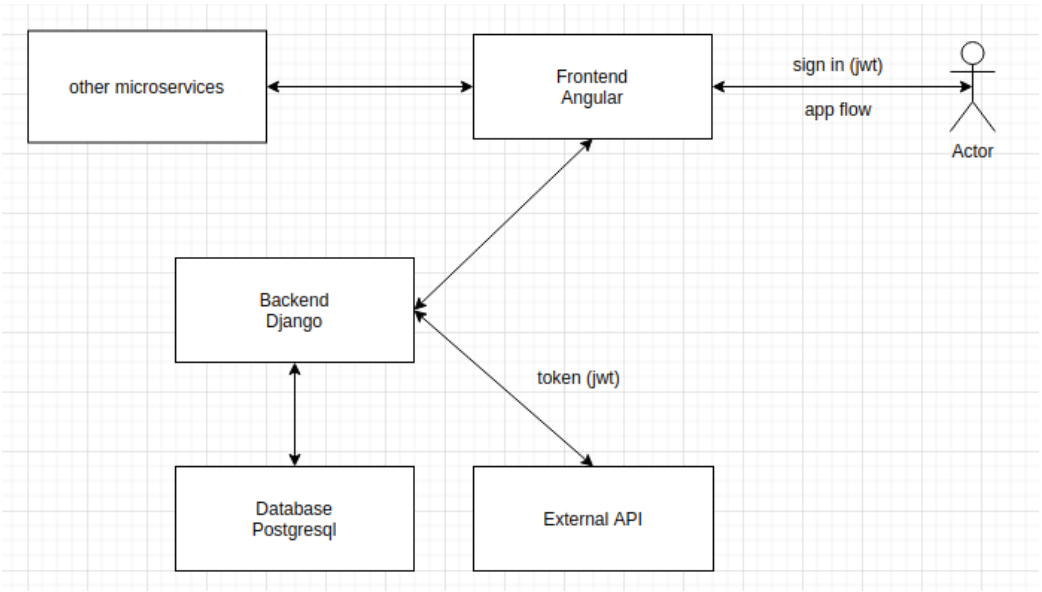
Frontend będzie prostą aplikacją z możliwością:

- Logowania / wylogowania z aplikacji
- Wyświetlania panelu użytkownika
- Wyświetlania wykresów z danymi
- Filtrowania wykresów
- operacji na zbiorze danych opisanych w sekcji "Backend"

Wybór przedstawienia danych (rodzaju wykresu) będzie ograniczony ze względu na wprowadzone dane - również określone w standardzie. Oprócz edycji zbiorów będzie udostępniony podstawowy interfejs do obsługi własnego konta - zmiana hasła i ew. stałe dotyczące wykresów / analizy danych.

3 Schematy

3.1 Komponenty aplikacyjne

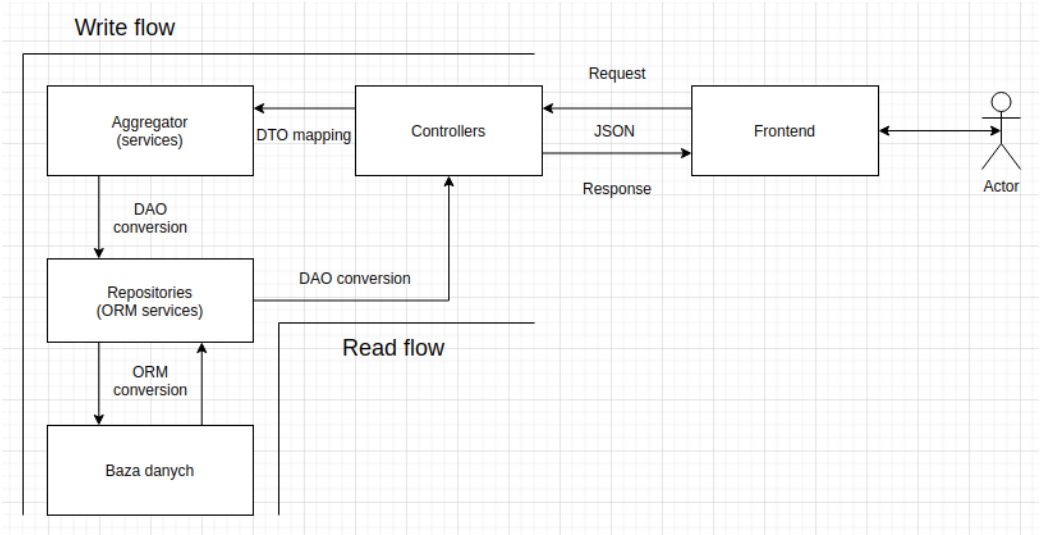


Rysunek 1: Komponenty aplikacyjne

Rysunek przedstawia komunikację między komponentami aplikacji. ”other services” oznacza możliwość podpięcia kolejnych serwisów do działającego frontendu. Nie jest on przypisany do konkretnego backendu. Natomiast potrzebuje zasilenia danymi, które oferuje API.

API umożliwia także komunikację z inną aplikacją, a dzięki generycznemu podejściu do tworzonych agregatów znajdzie wiele innych zastosowań biznesowych.

3.2 Komponenty oraz flow backendu



Rysunek 2: Komponenty backendu

API umożliwia komunikację z uwierzytelnieniem za pomocą JWT. Payload zapytania przekazywany będzie w formacie JSON, a następnie mapowany na kolejne klasy standaryzujące otrzymane dane. Wczytywanie informacji odbywa się bez udziału agregatora - bez wykorzystania DTO (w DAO znajdują się przetworzone dane z agregatami).

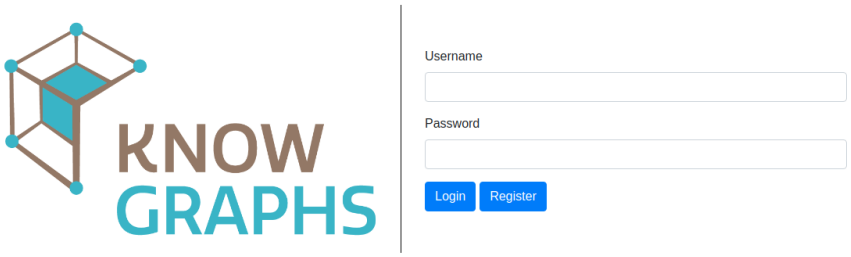
4 Implementacja

4.1 Frontend

Frontend składa się z kilku niezależnych elementów, pomocnych przy korzystaniu z aplikacji.

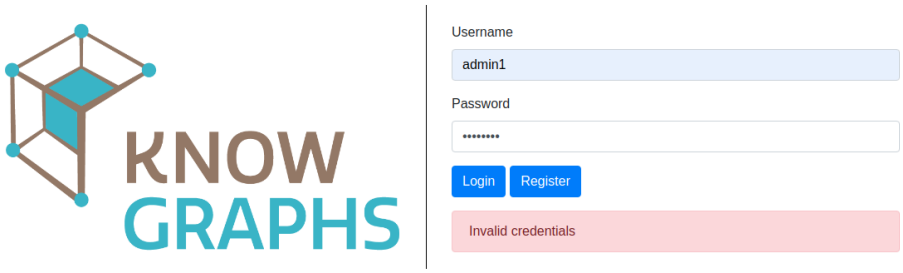
- Dashboardu z wykresami
- Swaggera ukazującego możliwości aplikacji
- Panelu administratora

Aplikacja posiada dwa niezależne ekrany logowania. Pierwszy z nich służy do uwierzytelnienia użytkownika na dashboardzie oraz w swaggerze. Drugi natomiast do panelu administracyjnego.



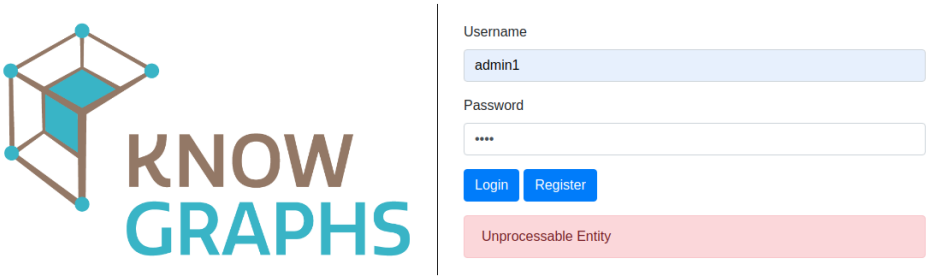
Rysunek 3: Panel logowania do dashboardu

Przed prezentacją wykresów, użytkownik zostaje sprawdzony czy login i hasło znajduje się w bazie.



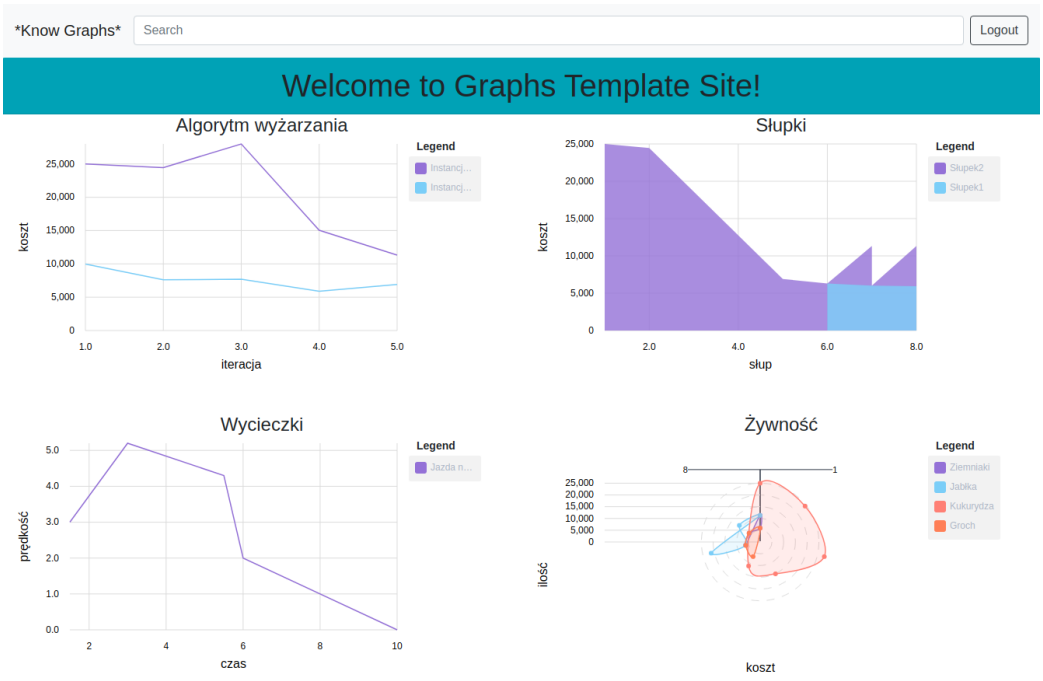
Rysunek 4: Błędne logowanie

Przy rejestracji następuje sprawdzenie czy użytkownik występuje już w bazie.



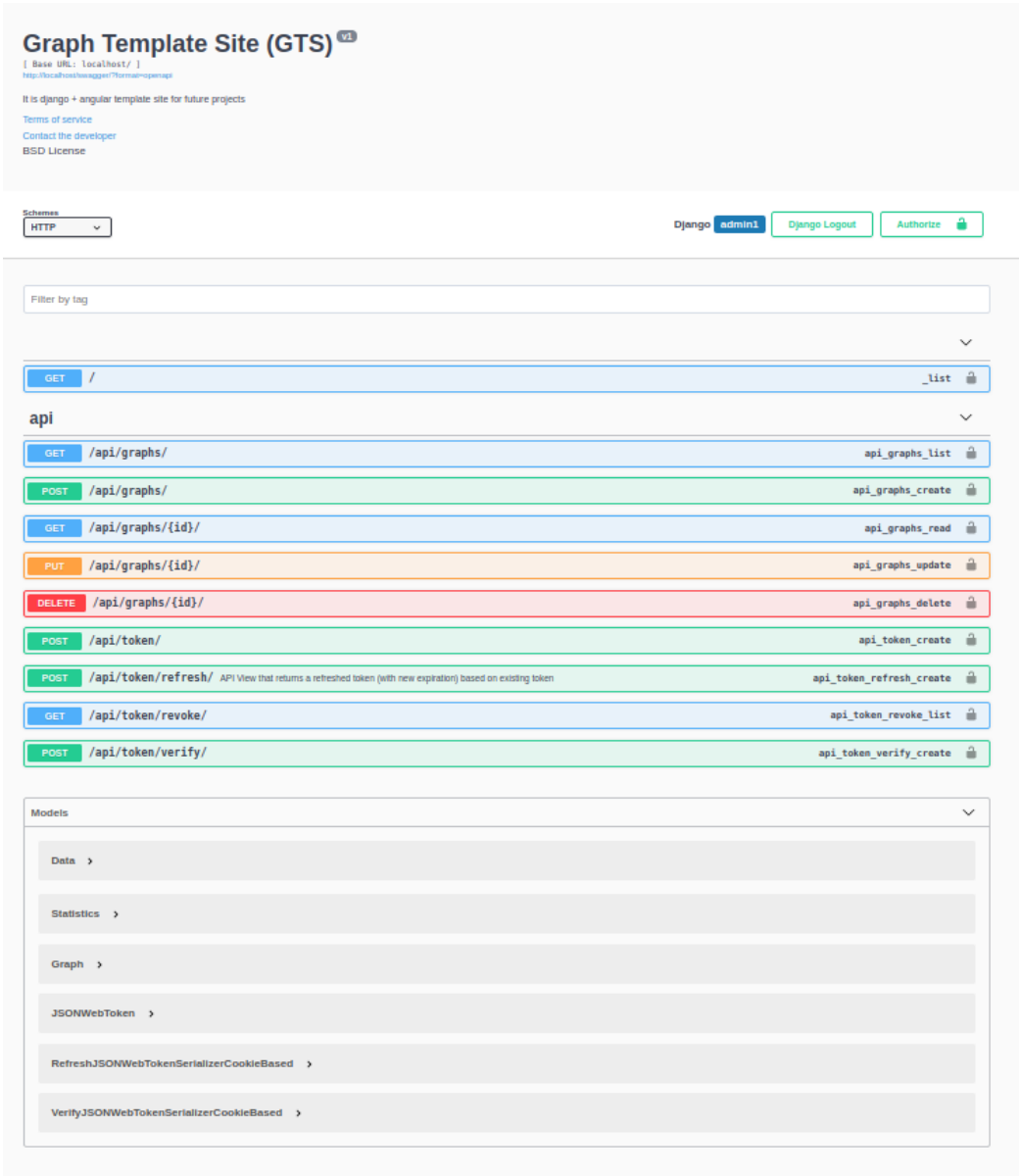
Rysunek 5: Błędne logowanie - rejestracja

Dashboard służy przeglądaniu wykresów. Jest to funkcjonalność, która pozwala zobrazować dane. Na chwilę obecną pozwala na filtrowanie wykresów po nazwie, co znacznie usprawnia przeszukiwanie. Funkcjonalność jest dostępna dla administratora jak i zwykłych użytkowników.



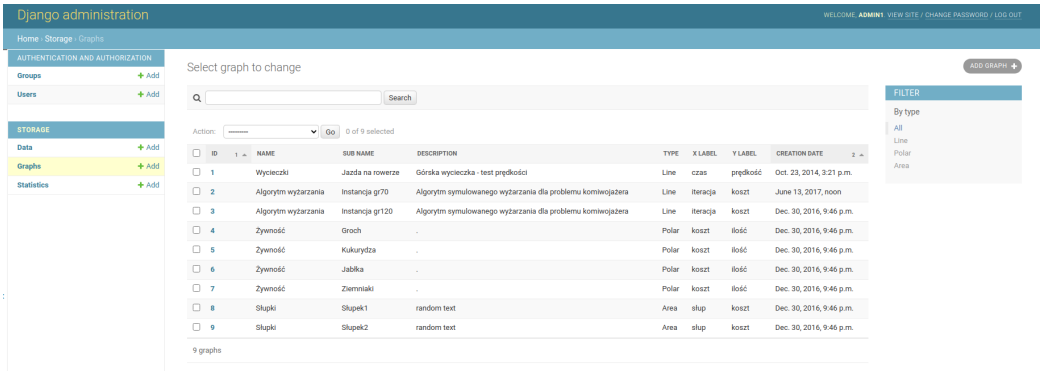
Rysunek 6: Dashboard

Jako zalogowany użytkownik mamy również dostęp do swaggera, dzięki któremu można przetestować aplikację.



Rysunek 7: Swagger

Ostatnim elementem jest panel administratora. W nim udostępniona jest możliwość przeszukiwania tabel związanych z wykresami oraz dostęp do statystyk (dostęp również poprzez API).



Rysunek 8: Panel administratora

4.2 Backend

Najważniejszym elementem backendu są modele oraz algorytmy wyliczania statystyk. Model zawiera również strukturę wykresów.

Listing 1: Modele

```
import math
from django.db import models
from django.utils import timezone

# Create your models here.
from typing import List


class Type(models.TextChoices):
    LINE = 'Line'
    POLAR = 'Polar'
    AREA = 'Area'



class Graph(models.Model):
    name = models.CharField(max_length=32)
    sub_name = models.CharField(max_length=32)
    description = models.CharField(max_length=64)
    type = models.CharField(
        max_length=16,
        choices=Type.choices,
        default=Type.LINE
    )
    x_label = models.CharField(max_length=16)
    y_label = models.CharField(max_length=16)
    creation_date = models.DateTimeField(default=timezone.now)

    objects = models.Manager()


class Data(models.Model):
```

```
    key = models.FloatField()
    value = models.FloatField()
    graph = models.ForeignKey(
        Graph,
        related_name='data',
        on_delete=models.CASCADE
    )

    objects = models.Manager()

    class Meta:
        verbose_name_plural = 'Data'


class Statistics(models.Model):
    a_mean = models.FloatField(null=True)
    w_mean = models.FloatField(null=True)
    median = models.FloatField(null=True)
    dominant = models.FloatField(null=True)
    std_deviation = models.FloatField(null=True)
    highest = models.FloatField(null=True)
    lowest = models.FloatField(null=True)
    graph = models.OneToOneField(
        Graph,
        related_name='statistics',
        on_delete=models.CASCADE,
        null=True
    )

    objects = models.Manager()

    @classmethod
    def create(cls, graph, data):
        stats = Statistics.generate_stats(data)
        return cls(
            a_mean=stats[0],
            w_mean=stats[1],
            median=stats[2],
```

```
            dominant=stats[3],
            std_deviation=stats[4],
            highest=stats[5],
            lowest=stats[6],
            graph=graph
        )


    class Meta:
        verbose_name_plural = 'Statistics'
```

Algorytmy ze względu na specyfikę aplikacji także znajdują się w ciele modelu.

Listing 2: Algorytmy

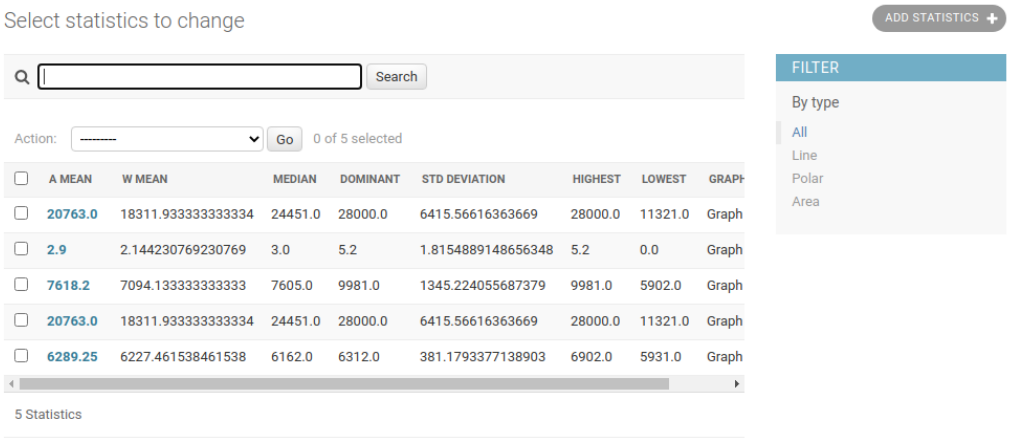
```
    @staticmethod
    def generate_stats(data):
        if len(data) > 0:
            data.sort(key=lambda x: x['value'])
            a_mean_divide,
            w_mean_divide =
            len(data),
            sum(x['key'] for x in data)
            if a_mean_divide != 0:
                a_mean =
                sum(x['value'] for x in data) / a_mean_divide
                std_deviation =
                math.sqrt(
                    sum(pow(x['value'] - a_mean, 2)
                        for x in data) / len(data))
            else:
                a_mean, std_deviation = None, None
            if w_mean_divide != 0:
                w_mean =
                sum(x['key'] * x['value']
                    for x in data) / w_mean_divide
            else:
```

```
                w_mean = None
            median = Statistics.__calculate_median(data)
            dominant = Statistics.__calculate_dominant(data)
            highest = data[-1]['value']
            lowest = data[0]['value']
            return a_mean, w_mean, median, dominant[0],
                std_deviation, highest, lowest
        return None, None, None, None, None, None, None

    @staticmethod
    def __calculate_median(data):
        if len(data) % 2 != 0:
            median =
                data[int(len(data) / 2)]['value']
        else:
            median =
                (data[int(len(data) / 2) - 1]['value'] +
                data[int(len(data) / 2)]['value']) / 2
        return median

    @staticmethod
    def __calculate_dominant(data):
        x = None
        count = 0
        for obj in data:
            if count == 0:
                x = obj['value']
                count += 1
            elif obj['value'] == x:
                count += 1
            else:
                count -= 1
        return x, count
```

Algorytmem do wyliczania dominanty (mody) jest algorytm Moora.

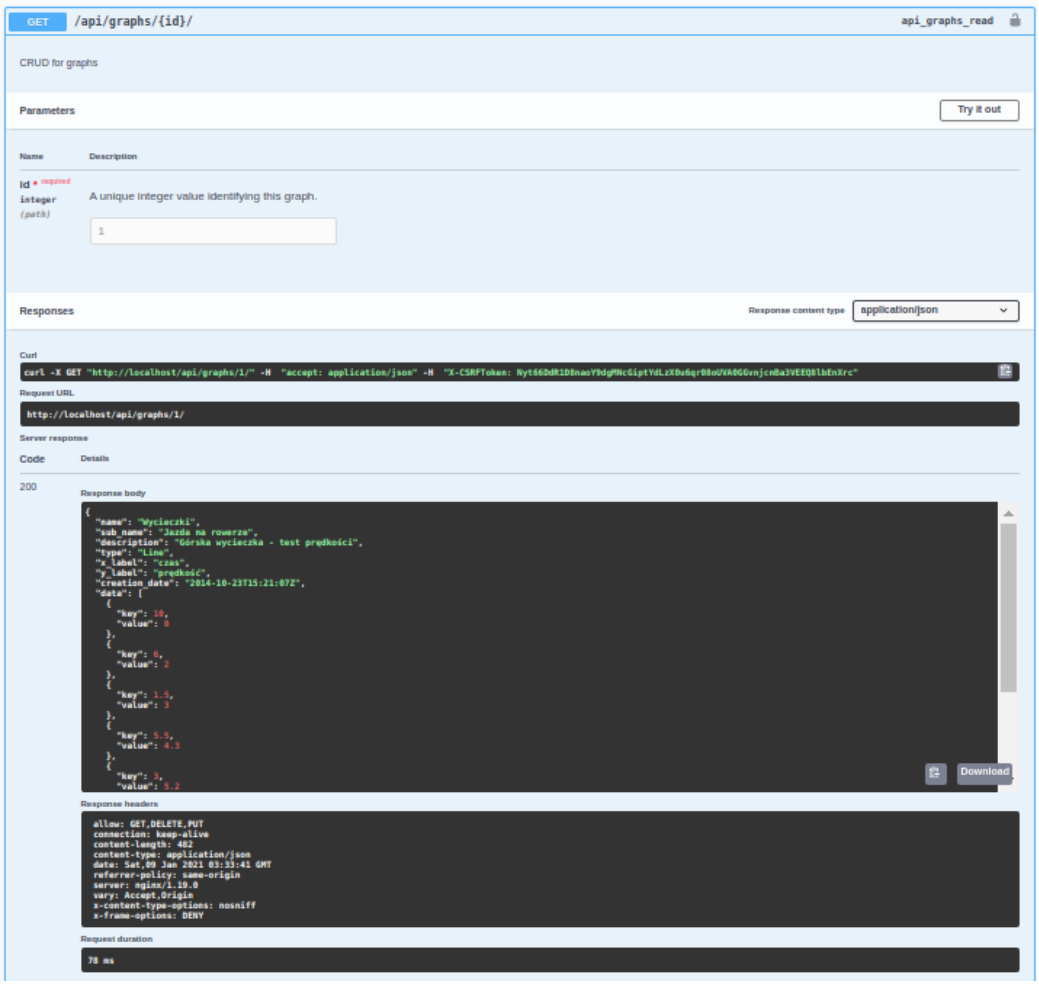
Zestaw statystyk możemy obejrzeć w dwóch miejscach:

- Panel administratora
- Swagger



Rysunek 9: Statystyki

Jednak w swaggerze jest on znacznie lepiej widoczny.



Rysunek 10: Statystyki - swagger

4.3 Docker

Cała aplikacja jest budowana i uruchamiana w dockerze. W tym celu stworzony został skrypt uruchamiający init.sh.

Listing 3: Init.sh

```
#!/bin/bash
docker-compose up -d --build
docker-compose exec web python
    /app/server/manage.py makemigrations
docker-compose exec web python
    /app/server/manage.py migrate
    # if error, try 'docker-compose␣down␣-v' and rebuild
docker-compose exec web python
    /app/server/manage.py collectstatic --noinput
docker-compose exec web python
    /app/server/manage.py createsuperuser --noinput
docker-compose exec db psql -f /app/data.sql
```

5 Wykorzystane technologie

5.1 Django

Aplikacja w frameworku Django będzie hostowana za pomocą serwera aplikacyjnego Gunicorn (WSGI).

5.2 Angular

Część frontendowa będzie najprawdopodobniej hostowana za pomocą NGINX.

5.3 Docker

Konteneryzacja wraz z docker-compose pozwoli na sprawny deployment aplikacji. Zostaną stworzone kontenery, które oddzielą poszczególne elementy w celu zachowania modularności.

Literatura

[1] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

[2] https://docs.djangoproject.com/en/3.1/ *Dokumentacja Django*.

[3] https://angular.io/docs *Dokumentacja Angular*.