

**UNIWERSYTET RZESZOWSKI**

**Kolegium Nauk Przyrodniczych**

Łukasz Domin

Nr albumu: 123631

Informatyka I stopnia

**Aplikacja internetowa do zarządzania ogrodem z elementami społecznościowymi**

Praca inżynierska

Praca wykonana pod kierunkiem

….....……………………………………..

Rzeszów, data

# Spis treści

1. [Spis treści 2](#_Toc186472617)
2. [Spis ilustracji 4](#_Toc186472618)
3. [Wstęp 6](#_Toc186472619)
4. [Cel i zakres pracy 7](#_Toc186472620)

[1 Projekt aplikacji 7](#_Toc186472621)

[1.1 Narzędzia i technologie 7](#_Toc186472622)

[1.1.1 Wymagania 7](#_Toc186472623)

[1.1.2 Część frontendowa 7](#_Toc186472624)

[1.1.3 Część backendowa 8](#_Toc186472625)

[1.1.4 Scrapper 9](#_Toc186472626)

[1.2 Diagram przypadków użycia(TODO) 9](#_Toc186472627)

[1.3 Struktura bazy danych 9](#_Toc186472628)

[1.3.1 Źródła danych 10](#_Toc186472629)

[1.3.2 Węzły, etykiety i relacje bazy danych 11](#_Toc186472630)

[2 Uruchomienie aplikacji 39](#_Toc186472631)

[2.1 Konfiguracja backendu 39](#_Toc186472632)

[2.2 Konfiguracja frontendu 42](#_Toc186472633)

[2.3 Aktualizacja danych pomiędzy backendem a frontendem 43](#_Toc186472634)

[3 Przedstawienie aplikacji(TODO) 45](#_Toc186472635)

[3.1 Struktura backendu 45](#_Toc186472636)

[3.1.1 Przykład obiektu 48](#_Toc186472637)

[3.1.2 Przykład procesu żądania i odpowiedzi 48](#_Toc186472638)

[3.2 Struktura frontendu 52](#_Toc186472639)

[3.3 Główny widok aplikacji 53](#_Toc186472640)

[3.4 Logowanie i rejestracja 53](#_Toc186472641)

[3.4.1 Logowanie 53](#_Toc186472642)

[3.4.2 Proces rejestracji 53](#_Toc186472643)

[3.5 CRUD pracownika aplikacji 53](#_Toc186472644)

[3.6 Przeglądanie ogólnodostępnych zasobów 53](#_Toc186472645)

[3.7 Zarządzanie swoimi zasobami 53](#_Toc186472646)

[3.7.1 Przykładowa walidacja dla tworzenia, edytowania oraz usuwania roślin 53](#_Toc186472647)

[3.7.2 Przykład działania walidacji 53](#_Toc186472648)

[3.7.3 Przykładowa walidacja dla dodawania, edytowania oraz usuwania rośliny na działce 53](#_Toc186472649)

[3.7.4 Przykład działania walidacji 53](#_Toc186472650)

1. [Zakończenie 55](#_Toc186472651)

# Spis ilustracji

[Obraz 1.3.1 Węzeł Roslina 12](#_Toc186472560)

[Obraz 1.3.2 Węzeł UzytkownikRoslina 12](#_Toc186472561)

[Obraz 1.3.3 Węzeł Wlasciwosc 14](#_Toc186472562)

[Obraz 1.3.4 Węzeł UzytkownikWlasciwosc 14](#_Toc186472563)

[Obraz 1.3.5 Węzeł ogrodu 15](#_Toc186472564)

[Obraz 1.3.6 Węzeł Dzialka 15](#_Toc186472565)

[Obraz 1.3.7 Węzeł Uzytkownik 18](#_Toc186472566)

[Obraz 1.3.8 Węzeł Pracownik 18](#_Toc186472567)

[Obraz 1.3.9 Węzeł Admin 19](#_Toc186472568)

[Obraz 1.3.10 Węzeł Ustawienia 20](#_Toc186472569)

[Obraz 1.3.11 Węzeł Post 21](#_Toc186472570)

[Obraz 1.3.12 Węzeł Komentarz 22](#_Toc186472571)

[Obraz 1.3.13 Węzeł RozmowaPrywatna 23](#_Toc186472572)

[Obraz 1.3.14 Węzeł Powiadomienie 24](#_Toc186472573)

[Obraz 1.3.15 Węzeł Zgloszenie 24](#_Toc186472574)

[Obraz 1.3.16 Relacja STWORZONA\_PRZEZ 25](#_Toc186472575)

[Obraz 1.3.17 Relacja ZASADZONA\_NA 26](#_Toc186472576)

[Obraz 1.3.18 Przykład rośliny z właściwościami 27](#_Toc186472577)

[Obraz 1.3.19 Relacja MA\_FORME 27](#_Toc186472578)

[Obraz 1.3.20 Relacja MA\_GLEBE 28](#_Toc186472579)

[Obraz 1.3.21 Relacja MA\_GRUPE 28](#_Toc186472580)

[Obraz 1.3.22 Relacja MA\_KOLOR\_KWIATOW 28](#_Toc186472581)

[Obraz 1.3.23 Relacja MA\_KOLOR\_LISCI 29](#_Toc186472582)

[Obraz 1.3.24 Relacja MA\_OKRES\_KWITNIENIA 29](#_Toc186472583)

[Obraz 1.3.25 Relacja MA\_OKRES\_OWOCOWANIA 29](#_Toc186472584)

[Obraz 1.3.26 Relacja MA\_KWIAT 30](#_Toc186472585)

[Obraz 1.3.27 Relacja MA\_ODCZYN 30](#_Toc186472586)

[Obraz 1.3.28 Relacja MA\_OWOC 30](#_Toc186472587)

[Obraz 1.3.29 Relacja MA\_PODGRUPE 31](#_Toc186472588)

[Obraz 1.3.30 Relacja MA\_POKROJ 31](#_Toc186472589)

[Obraz 1.3.31 Relacja SKOMENTOWAL 32](#_Toc186472590)

[Obraz 1.3.32 Relacja ODPOWIEDZIAL 32](#_Toc186472591)

[Obraz 1.3.33 Relacja JEST\_W\_POSCIE 33](#_Toc186472592)

[Obraz 1.3.34 Relacja JEST\_W\_ROZMOWIE 33](#_Toc186472593)

[Obraz 1.3.35 Relacja MA\_KOMENTARZ 34](#_Toc186472594)

[Obraz 1.3.36 Relacja MA\_POST 34](#_Toc186472595)

[Obraz 1.3.37 Relacja MA\_WIADOMOSC 35](#_Toc186472596)

[Obraz 1.3.38 Relacja OCENIL 35](#_Toc186472597)

[Obraz 1.3.39 Relacja WYSYLA\_POWIADOMIENIE 36](#_Toc186472598)

[Obraz 1.3.40 Relacja POWIADAMIA 36](#_Toc186472599)

[Obraz 1.3.41 Relacja ZGLASZA 37](#_Toc186472600)

[Obraz 1.3.42 Relacja MA\_USTAWIENIA 38](#_Toc186472601)

[Obraz 1.3.43 Relacja MA\_OGROD 38](#_Toc186472602)

[Obraz 1.3.44 Relacja MA\_DZIALKE 39](#_Toc186472603)

[Obraz 2.1.1 Parametry bazy danych 40](#_Toc186472604)

[Obraz 2.1.2 Parametry maila 40](#_Toc186472605)

[Obraz 2.1.3 Uruchomiona baza danych 40](#_Toc186472606)

[Obraz 2.1.4 Widok Swaggera 41](#_Toc186472607)

[Obraz 2.2.1 Widok strony głównej 42](#_Toc186472608)

[Obraz 2.3.1 Link do api-docs 43](#_Toc186472609)

[Obraz 2.3.2 Fragment pliku openApi.json 44](#_Toc186472610)

[Obraz 3.1.1 Struktura backendu 45](#_Toc186472611)

[Obraz 3.1.2 Katalog uploads 46](#_Toc186472612)

[Obraz 3.1.3 Katalog resources 47](#_Toc186472613)

[Obraz 3.1.4 Widok kontrolera Postu (Swagger) 48](#_Toc186472614)

[Obraz 3.1.5 Widok żądania GET /posty 49](#_Toc186472615)

[Obraz 3.3.1 Główny widok aplikacji 53](#_Toc186472616)

# Wstęp

Aplikacja internetowa pt. „Wirtualny Ogród Yukka” jest m systemem umożliwiający zarządzanie wirtualnym ogrodem, wymianę wiedzy i doświadczeń pomiędzy użytkownikami w przestrzeni społecznościowej.

Tworzona aplikacja dostarcza użytkownikom intuicyjne rozwiązania, takie jak system powiadomień o społecznościowych (np. odpowiedź na komentarz, otrzymanie zaproszenia) lub stricte ogrodowych(np. powiadomienie o okresie owocowania roślin) w ogrodzie, możliwość projektowania własnych działek z roślinami wybranymi z katalogu, które można następnie dostosować do swoich upodobań. Możliwe jest również stworzenie własnej rośliny i posadzenie jej na swojej działce. Dodatkowo, poprzez integrację części społecznościowej możliwa jest interakcja z innymi użytkownikami, udostępnianie widoku działki, dzielenie się poradami na temat roślin, a także możliwość prowadzenia rozmów prywatnych. Dzięki temu system wspiera zarówno początkujących, jak i zaawansowanych ogrodników, tworząc przyjazne środowisko do dzielenia się swoją pasją.

# Cel i zakres pracy

## Projekt aplikacji

### Narzędzia i technologie

#### Wymagania

Do testowania i działania aplikacji wymagane są następujące komponenty:

* Java 17
* npm (Node Package Manager)
* Baza danych Neo4j. W tym przypadku, używany jest program Neo4j Desktop
* Ustawiony adres email, który obsługuje żądania zmiany hasła, adresu email, czy aktywacji konta
* Zainstalowany python3 (Do uruchomienia Scrappera)

#### Część frontendowa

**Języki programowania:** TypeScript 5.5.2

TypeScript to rozszerzenie języka JavaScript dodające typowanie statyczne, klasy oraz interfejsy. Ma dużo mniejszą tolerancję na błędy, a sam jego kod jest kompilowany na końcu do JavaScript.

Link do pobrania: <https://www.typescriptlang.org/download/>

**Framework:** Angular 18.2.11

Jest to framework opartym na języku TypeScript do budowy aplikacji webowych, rozwijanego przez Google. Wspomniana wersja posiada mechanikę sygnałów do komunikacji pomiędzy komponentami, wsparcie dla SSR (rendererowania po stronie serwera), hydracji (użycie statycznej części wygenerowanej strony przez SSR), oraz lepsza obsługa ładowania asynchronicznego.

Link do pobrania: <https://v18.angular.dev/installation>

**Pozostałe narzędzia i technologie:**

**Bootstrap 5** - biblioteka CSS. Zawiera zestaw narzędzi ułatwiających tworzenie interfejsu graficznego stron oraz aplikacji internetowych. Bazuje głównie na gotowych rozwiązaniach HTML oraz CSS i może być stosowany m.in. do stylizacji poszczególnych elementów strony.

**Npm (Node Package Manager**) - Menedżer pakietów dla JavaScriptu, dostarczany wraz z Node.js. Służy do instalowania, zarządzania i udostępniania bibliotek oraz narzędzi JavaScript.

#### Część backendowa

**Języki programowania:** Java 17

Java to język obiektowy działający na różnych systemach operacyjnych bez większych trudów. Jest szeroko stosowana do tworzenia przeróżnych aplikacji czy to internetowych, czy to mobilnych.

Link do pobrania: <https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>

**Framework:** Spring Boot 3.3.2

Spring Boot to framework języka Java, który upraszcza proces konfiguracji i uruchamiania aplikacji, automatycznie konfigurując większość komponentów, skracając kod przy użyciu adnotacji, a także upraszcza komunikację z bazą danych i frontendem.

Link do pobrania: <https://docs.spring.io/spring-boot/installing.html>

#### Scrapper

Jest to scrapper napisany w języku Python używany do pobierania danych o roślinach ze strony <https://e-katalogroslin.pl> do pliku *katalog\_roslin.csv.* Wykorzystuje on sterownik Firefox, wielowątkowość oraz asynchroniczność do szybszego pobierania danych ze stron.

##### Pobranie roślin

Aby rozpocząć proces pobierania roślin, należy uruchomić plik scrap\_threads.py, który pobierze dane od początkowej strony aż do ostatniej. Należy mieć na uwadze, że czasami program może mieć trudności z dostępnością do strony, przejściem na inną stronę oraz inne czynniki zewnętrzne które mogą zakłócić bądź uniemożliwić scrappowanie danych. Pobranie wszystkich danych ze strony powinno zająć ok. 50-70 minut.

##### Pobranie obrazów roślin

Plik *katalog\_roslin.csv* zawiera informacje o roślinach takie jak nazwa rośliny, nazwa łacińska, opis, kwasowość gleby, a także link do obrazu rośliny (jeśli takowy posiada). Aby pobrać obrazy rośliny, należy uruchomić plik *download\_images.py*, który będzie iterował przez kolejne rzędy pliku *katalog\_roslin.csv* i zapisze obrazy do folderu *images.* Proces ten powinien zająć do 5 minut.

### Diagram przypadków użycia(TODO)

1. Diagram przypadków użycia (UML)
2. (Konsultacja z Kamil Matusz)

### Struktura bazy danych

**Baza danych:** Neo4j 5.12.0

Neo4j jest grafową bazą danych, która przechowuje dane w postaci węzłów, ich etykiet, relacji pomiędzy nimi oraz właściwościami węzłów bądź relacji. Została ona wybrana dlatego, że ten typ bazy danych jest polecany w budowaniu aplikacji społecznościowych ze względu na budowę relacji pomiędzy poszczególnymi elementami bazy.

#### Źródła danych

Źródłem danych jest strona internetowa *e-katalogroslin.pl* należąca do Związku Szkółkarzy Polskich, który jest stowarzyszeniem producentów drzew, krzewów i bylin, a także roślin owocowych. Wymieniona strona jest katalogiem zawierający ponad 10 000 roślin o różnych właściwościach takich jak preferowany rodzaj gleby, okres owocowania, zimozieloność liści, zakres wysokości itp.

Link do strony: <https://e-katalogroslin.pl>

#### Węzły, etykiety i relacje bazy danych

##### Węzły

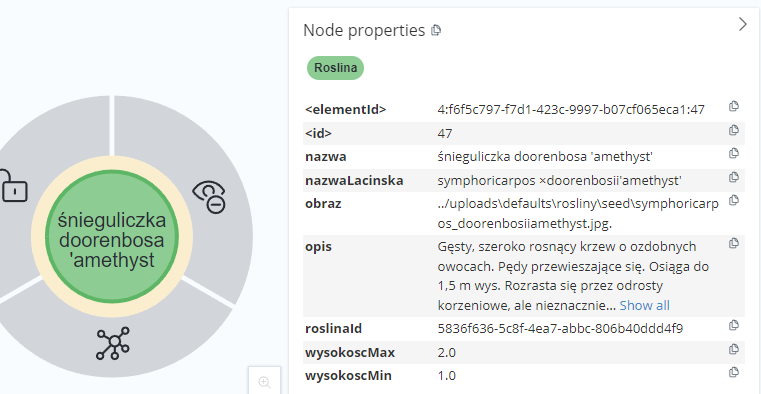
###### Roslina

Roślina jest jednym z głównym węzłów w bazie danych. Reprezentuje roślinę z katalogu oraz roślinę użytkownika i jest wykorzystywana w ogrodzie. Rośliny użytkownika posiadają etykietę Roslina oraz UzytkownikRoslina, a roślina z katalogu może mieć tylko etykietę Roslina.

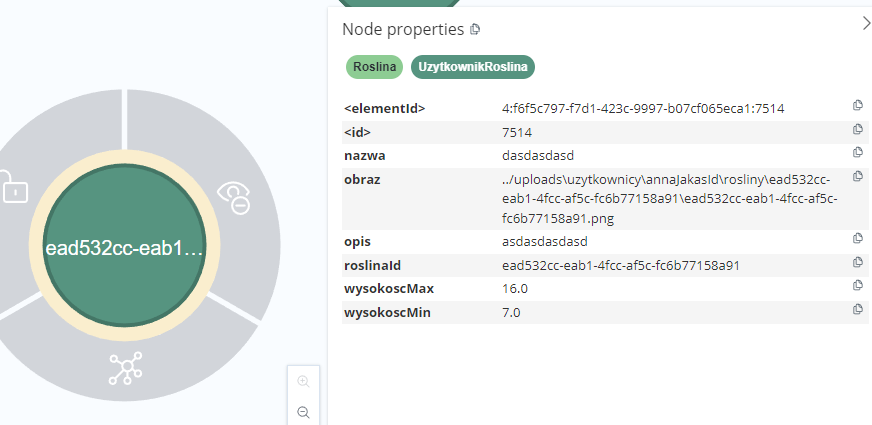
**Etykiety:** Roslina, UzytkownikRoslina

**Właściwości:**

* roslinaId - Id rośliny wygenerowane na backendzie. Jest ono używane zamiast id, gdyż w Neo4J jest ono przestarzałe i zostanie w przyszłości zastąpione.
* nazwa – nazwa rośliny
* nazwaLacinska – nazwa łacińska rośliny
* opis – opis rośliny
* wysokoscMin, wysokoscMax – wysokość minimalna i maksymalna rośliny
* obraz – ścieżka do obrazu rośliny, jeśli takowy posiada. W przeciwnym wypadku obrazem rośliny jest obraz domyślny



Obraz 1.3.1 Węzeł Roslina



Obraz 1.3.2 Węzeł UzytkownikRoslina

Właściwości rośliny

Rośliny posiadają właściwości o 16 różnych etykietach takich jak *Owoc, Grupa, Gleba,* które klasyfikują typ danej właściwości. Zwykła właściwość ma etykietę *Wlasciwosc*, za to właściwość użytkownika ma dodatkowo etykietę *UzytkownikWlasciwosc.*

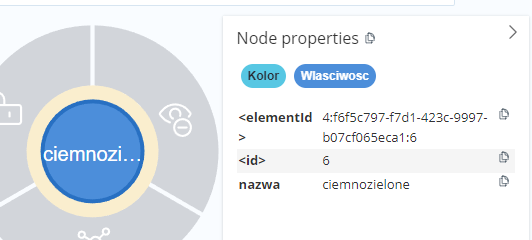
**Etykiety główne:** Wlasciwosc, UzytkownikWlasciwosc

**Etykiety dodatkowe:**

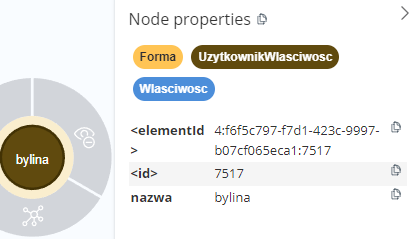
* Forma (Forma rośliny)
* Gleba (Rodzaj gleby przyjazny dla rośliny)
* Grupa (Grupa rośliny)
* Kolor (Kolor kwiatów albo liści rośliny)
* Kwiat (Rodzaj kwiatu rośliny)
* Odczyn (Odczyn ph gleby przyjazny dla rośliny)
* Okres (Okres owocowania albo kwitnienia rośliny wyrażony w miesiącach)
* Owoc (Rodzaj owocu rośliny)
* Podgrupa (Podgrupa rośliny)
* Pokroj (Pokrój/Kształt rośliny)
* SilaWzrostu (Siła wzrostu rośliny, czyli jak szybko rośnie)
* Stanowisko (Odpowiednie nasłonecznienie rośliny)
* Wilgotnosc (Optymalna wilgotność gleby rośliny)
* Zastosowanie (Do czego pasuje roślina, albo czy ma jakiś użytek poza wyglądem)
* Zimozielonosc (Jak zachowują się liście/igły rośliny podczas zimy, czy opadają, a może nie)

**Właściwości:**

nazwa – nazwa właściwości



Obraz 1.3.3 Węzeł Wlasciwosc



Obraz 1.3.4 Węzeł UzytkownikWlasciwosc

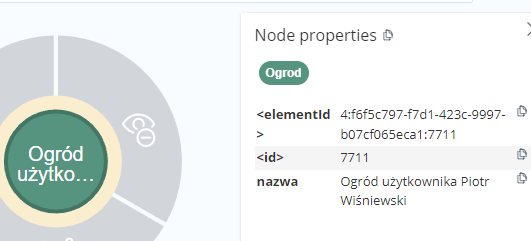
###### Ogrod

Ogród jest węzłem tworzonym dla każdego użytkownika. Reprezentuje ogród podzielony na 10 działek, które również są węzłami złączonymi z nim.

**Etykiety:** Ogrod

**Właściwości:**

* nazwa – nazwa ogrodu



Obraz 1.3.5 Węzeł ogrodu

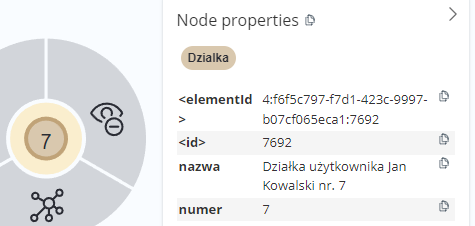
###### Dzialka

Działka jest węzłem związanym z ogrodem, tworzony wraz z nim. Łącznie dla każdego użytkownika jest tworzone po 10 działek na ogród, gdzie każda działka ma wymiary 20x20 kafelków.

**Etykiety:** Dzialka

**Właściwości:**

* nazwa – nazwa działki
* numer – numer działki od 1 do 10



Obraz 1.3.6 Węzeł Dzialka

###### Uzytkownik

Użytkownik jest węzłem pełniącym rolę użytkownika w bazie danych. Zwykli użytkownicy posiadają tylko etykietę Uzytkownik, za to pracownicy posiadają dodatkowo etykietę Pracownik, a admini posiadają wszystkie z tych etykiet. Każda etykieta oznacza inny poziom uprawnień. Poniżej znajdują się przykładowe uprawnienia dla kolejno użytkownika, pracownika i admina.

Uprawnienia użytkownika:

* Tworzenie własnych roślin i ich modyfikacja
* Dodawanie roślin do swojego ogrodu i ich modyfikacja
* Tworzenie postów i komentarzy
* Zapraszanie i uczestniczenie w rozmowach prywatnych
* Zgłaszanie i blokowanie użytkowników
* Zmiana nazwy ogrodu, działki, informacji profilowych
* Zmiana własnych ustawień, np. wyłączenie powiadomień o odpowiedziach do własnych komentarzy, czy wyłączenie powiadomień okresie owocowania roślin na działce
* Usuwanie własnych roślin, komentarzy i postów

Uprawnienia pracownika:

* Tworzenie roślin do katalogu i ich modyfikacja
* Dostęp do treści prywatnych zwykłych użytkowników
* Usuwanie treści zwykłych użytkowników
* Banowanie zwykłych użytkowników

Uprawnienia admina:

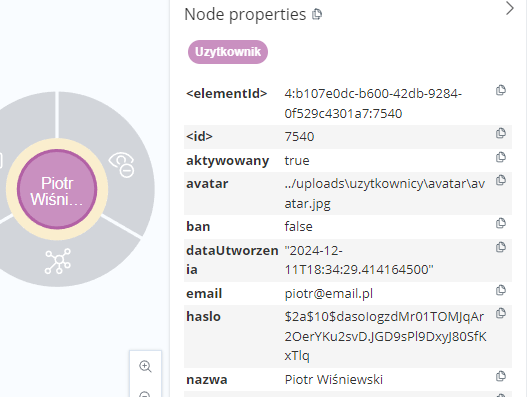
* Dostęp do treści prywatnych użytkowników i pracowników
* Usuwanie treści użytkowników i pracowników
* Banowanie pracowników

Dodatkowo, każdy użytkownik ma węzeł Ustawienia, węzeł Ogrod, oraz 10 węzłów Dzialka przypisanych do tego ogrodu. Informacje o użytkowniku są w ten sposób rozdzielone, aby zoptymalizować pobór danych z bazy danych i pobierać tylko te części danych, które są potrzebne.

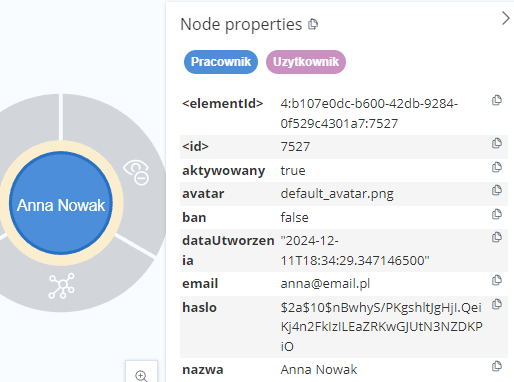
**Etykiety:** Uzytkownik, Pracownik, Admin

**Właściwości:**

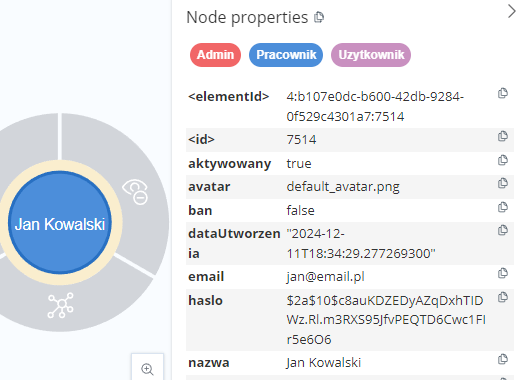
* uzytId – Id użytkownika wygenerowane na backendzie
* nazwa – nazwa użytkownika
* email – adres email użytkownika
* haslo – zaszyfrowane hasło użytkownika
* avatar – zdjęcie użytkownika. Jeśli takowego nie posiada, jest ono zastąpione domyślnym avatarem
* dataUtworzenia – data utworzenia użytkownika
* aktywowany – oznacza, czy konto zostało aktywowane po rejestracji
* ban – czy użytkownik jest zbanowany
* banDo – data, do której użytkownik jest zbanowany
* imie – imię użytkownika na jego profilu
* nazwisko – nazwisko użytkownika na jego profilu
* miasto – miasto użytkownika na jego profilu
* miejsceZamieszkania – miejsce zamieszkania użytkownika na jego profilu
* opis – opis użytkownika na jego profilu



Obraz 1.3.7 Węzeł Uzytkownik



Obraz 1.3.8 Węzeł Pracownik



Obraz 1.3.9 Węzeł Admin

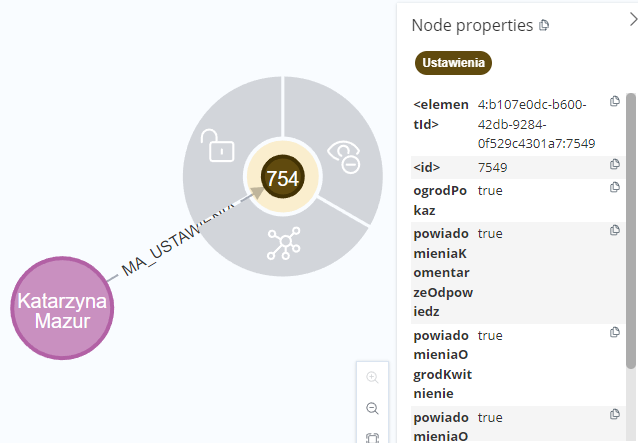
###### Ustawienia

Węzeł Ustawienia tworzony dla każdego użytkownika reprezentuje jego ustawienia. Może on decydować, jakie on chce otrzymywać powiadomienia, czy co chce on pokazywać publicznie.

**Etykiety:** Ustawienia

**Właściwości:**

* ogrodPokaz – decyduje, czy użytkownik pozwala innym użytkownikom przeglądać jego ogród. Nie tyczy się to pracowników i adminów.
* powiadomieniaKomentarzeOdpowiedz – decyduje, czy użytkownik ma otrzymywać powiadomienia o odpowiedzi na jego komentarz
* powiadomieniaOgrodKwitnienie – decyduje, czy użytkownik ma otrzymywać powiadomienia o nadejściu okresu kwitnienia roślin w swoim ogrodzie
* powiadomieniaOgrodOwocowanie – decyduje, czy użytkownik ma otrzymywać powiadomienia o nadejściu okresu owocowania w swoim ogrodzi
* statystykiProfilu – decyduje, czy na profilu użytkownika mają się wyświetlać jego statystyki.



Obraz 1.3.10 Węzeł Ustawienia

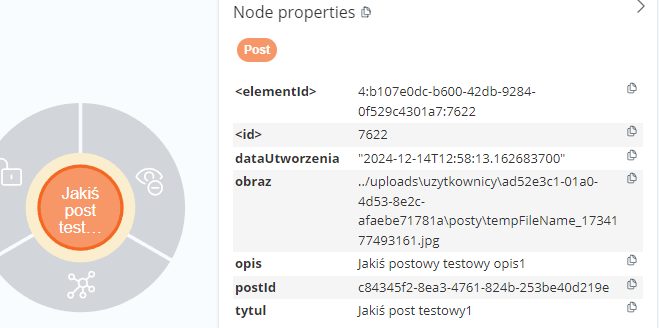
###### Post

Węzeł Post pełni rolę postu w części społecznościowej. Oprócz nazwy i opisu może również zawierać jeden obraz. Do tego postu mogą być przypisywane komentarze użytkowników, a także może być on oceniany przez nich.

**Etykiety:** Post

**Właściwości:**

* postId – Id postu wygenerowany na backendzie
* tytul – tytuł postu
* opis – opcjonalny opis postu
* obraz – ścieżka do obrazu postu
* dataUtworzenia – data utworzenia posta



Obraz 1.3.11 Węzeł Post

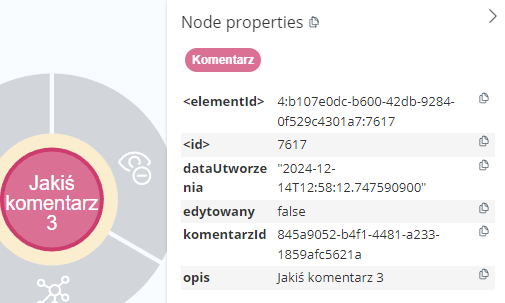
###### Komentarz

Węzeł Komentarz pełni rolę komentarza w części społecznościowej. Są one używane do komentowania postów, odpowiadania na komentarze, a także służą jako wiadomości w rozmowach prywatnych.

**Etykiety:** Komentarz

**Właściwości:**

* komentarzId – Id komentarza wygenerowany na backendzie
* opis – treść komentarza
* obraz – ścieżka do obrazu komentarza
* edytowany – oświadcza, czy komentarz był edytowany
* dataUtworzenia – data utworzenia komentarza



Obraz 1.3.12 Węzeł Komentarz

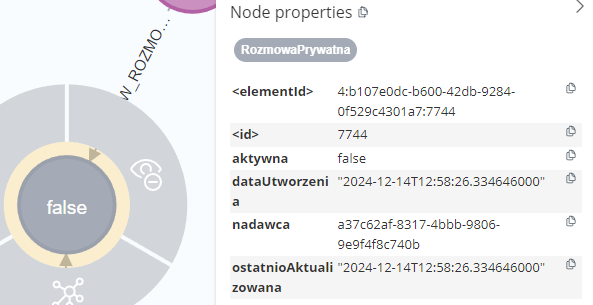
###### RozmowaPrywatna

Węzeł RozmowaPrywatna pełni rolę rozmowy prywatnej pomiędzy dwoma użytkownikami w części społecznościowej. Aby rozmowa prywatna miała miejsce, użytkownik A musi zaprosić użytkownika B do rozmowy, a odbiorca musi ją zaakceptować. Wtedy możliwe jest pisanie wiadomości pomiędzy nimi.

**Etykiety:** RozmowaPrywatna

**Właściwości:**

* aktywna – oznacza, czy rozmowa prywatna jest aktywna (odbiorca zaakceptował zaproszenie)
* nadawca – id nadawcy
* dataUtworzenia – data utworzenia rozmowy (wysłania zaproszenia)
* ostatnioAktualizowana – data od ostatniej wysłanej wiadomości, lub od kiedy rozmowa została zaakceptowana



Obraz 1.3.13 Węzeł RozmowaPrywatna

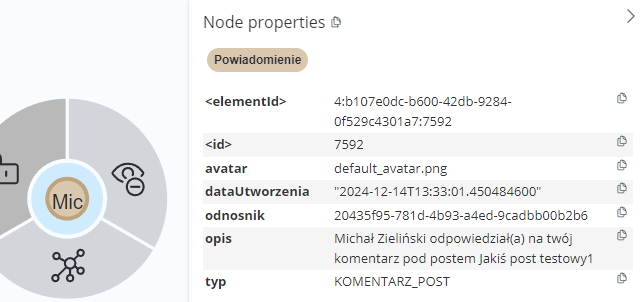
###### Powiadomienie

Węzeł Powiadomienie powiadania użytkownika o różnych zdarzeniach w systemie, takie jak odpowiedź na komentarz, rozpoczęcie okresu owocowania roślin na działce, czy zaproszeniu przez użytkownika do rozmowy prywatnej. Jeśli posiada ono dodatkowo etykietę Zgloszenie, to pełni ono rolę zgłoszenia przez użytkownika, które jest przypisywane do pracownika o najmniejszej aktualnej liczbie zgłoszeń. Użytkownik może ustawić powiadomienie jako przeczytane, albo je usunąć pod warunkiem, że nie jest to zgłoszenie.

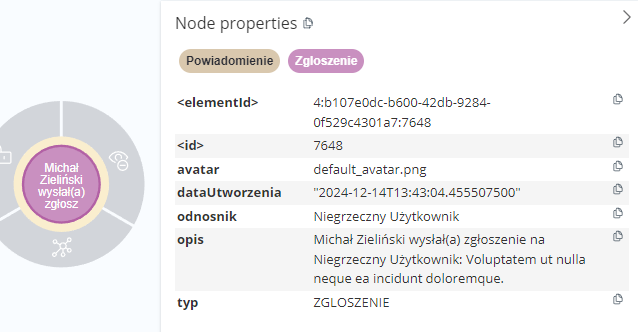
**Etykiety:** Powiadomienie, Zgloszenie

**Właściwości:**

* typ – typ powiadomienia (komentarz\_post, zaproszenie, owocowanie\_roslin)
* opis – wygenerowany opis powiadomienia
* odnośnik – odnośnik do wspomnianego elementu (post, rozmowa prywatna, ogród)
* dataUtworzenia – data utworzenia powiadomienia
* avatar – ścieżka do avataru powiadomienia. W przypadku powiadomień związanych z użytkownikiem, używa się jego avatara. W przeciwnym wypadku zastępowany jest on domyślnym avatarem.



Obraz 1.3.14 Węzeł Powiadomienie



Obraz 1.3.15 Węzeł Zgloszenie

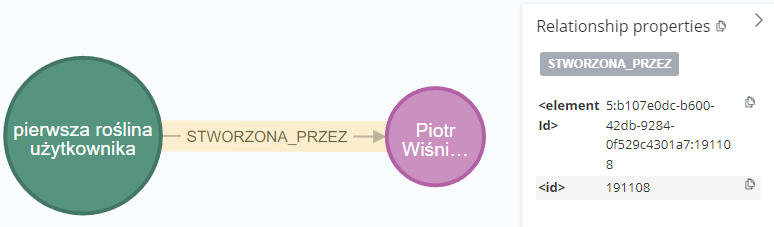
##### Relacje

###### Relacje roślin

Roślina posiada jakieś relacje.

STWORZONA\_PRZEZ

Połączenie kierunku Uzytkownik -> UzytkownikRoslina. Wskazuje roślinę nienależącą do katalogu, która została stworzona przez użytkownika.



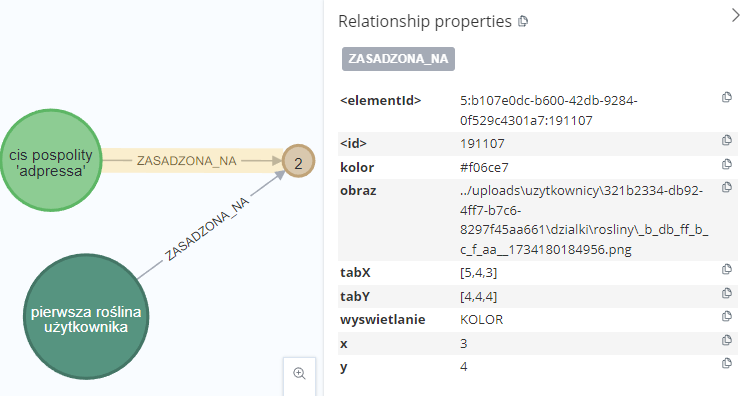
Obraz 1.3.16 Relacja STWORZONA\_PRZEZ

ZASADZONA\_NA

Połączenie kierunku Roslina -> Dzialka albo UzytkownikRoslina -> Dzialka. Wskazuje działkę, na której została zasadzona roślina wraz ze szczególnymi informacjami takie jak pozycja, obszar rośliny, kolor kafelków, czy własny obraz rośliny.

**Właściwości:**

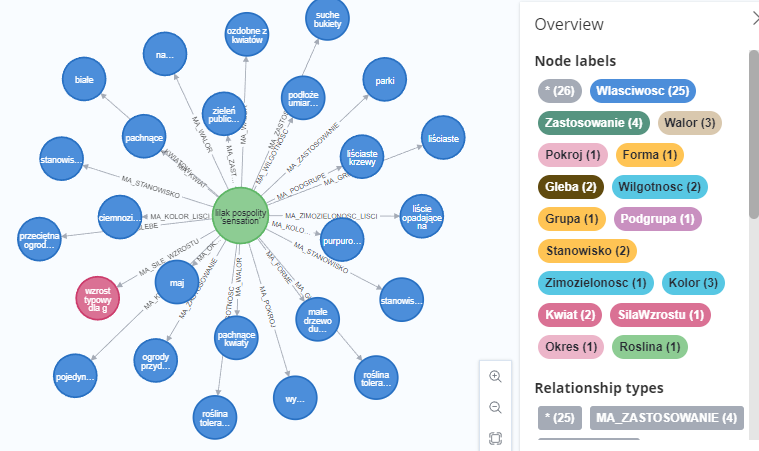
* x, y – pozycja (x, y) rośliny
* tabX, tabY – poszczególne kafelki (x, y) rośliny rozdzielone na dwie tablice jednowymiarowe. Jest to tak skonstruowane, gdyż Neo4j nie zezwala na wstawiania tablic wielowymiarowych do bazy danych.
* kolor – kolor kafelków rośliny podawany w HEX
* obraz – ścieżka do własnego obrazu rośliny
* tekstura – ścieżka do tekstury kafelków rośliny
* wyswietlanie – tryb wyświetlania rośliny na działce
  + KOLOR – wyświetla obszar rośliny jako pokolorowane kafelki
  + TEKSTURA – wyświetla obszar rośliny jako kafelki z nałożoną teksturą. Ten tryb jest możliwy tylko, jeśli zasadzona roślina posiada teksturę.
  + TEKSTURA\_KOLOR – wyświetla obszar rośliny jako pokolorowane kafelki z nałożoną teksturą



Obraz 1.3.17 Relacja ZASADZONA\_NA

###### Relacje właściwości

Właściwości dla danej rośliny są przypisywane w kierunku Roslina/UzytkownikRoslina - > Wlasciwosc/UzytkownikWlasciwosc. Poniżej znajduje się przykład pojedynczej rośliny wraz z jej właściwościami.

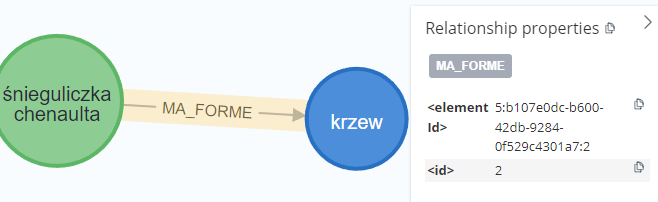


Obraz 1.3.18 Przykład rośliny z właściwościami

W dalszej części znajduje się część z relacji właściwości dla roślin.

MA\_FORME

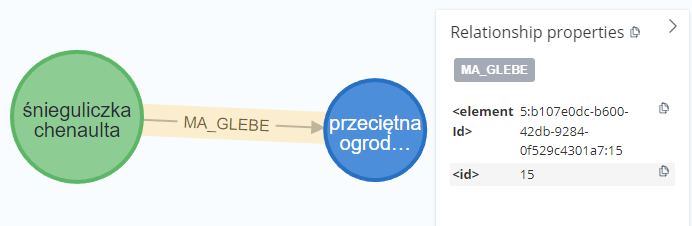
Połączenie kierunku Roslina -> Forma. Wskazuje formę rośliny.



Obraz 1.3.19 Relacja MA\_FORME

MA\_GLEBE

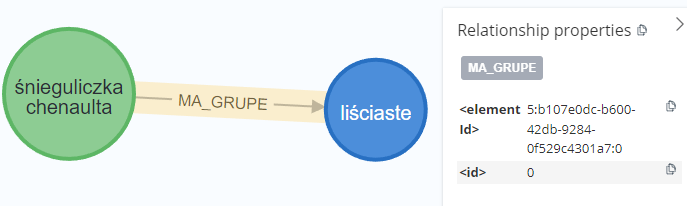
Połączenie kierunku Roslina -> Gleba. Wskazuje glebę, która jest stosowna dla danej rośliny.



Obraz 1.3.20 Relacja MA\_GLEBE

MA\_GRUPE

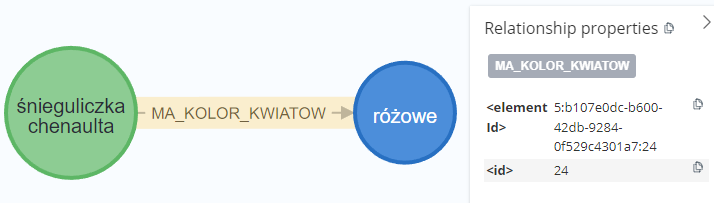
Połączenie kierunku Roslina -> Grupa. Wskazuje grupę, do której roślina należy.



Obraz 1.3.21 Relacja MA\_GRUPE

MA\_KOLOR\_KWIATOW

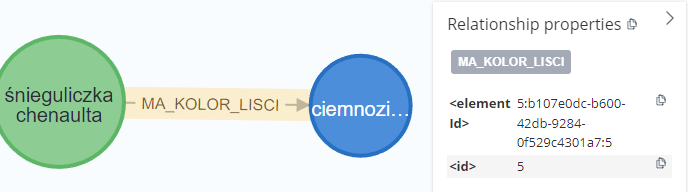
Połączenie kierunku Roslina -> Kolor. Wskazuje kolor kwiatów rośliny.



Obraz 1.3.22 Relacja MA\_KOLOR\_KWIATOW

MA\_KOLOR\_LISCI

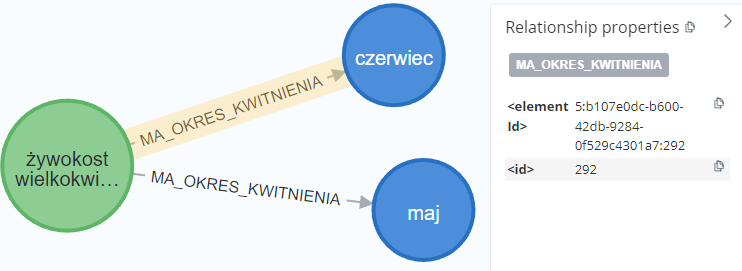
Połączenie kierunku Roslina -> Kolor. Wskazuje kolor liści rośliny.



Obraz 1.3.23 Relacja MA\_KOLOR\_LISCI

MA\_OKRES\_KWITNIENIA

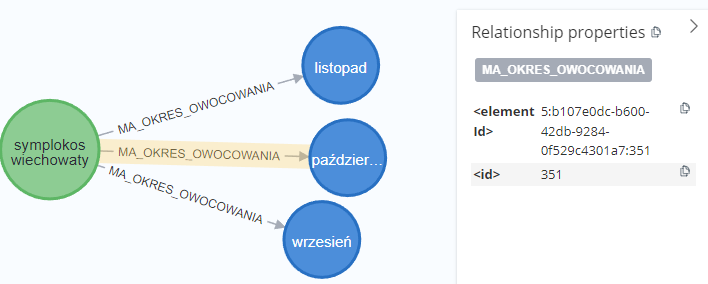
Połączenie kierunku Roslina -> Okres. Wskazuje miesiące, w której roślina kwitnie.



Obraz 1.3.24 Relacja MA\_OKRES\_KWITNIENIA

MA\_OKRES\_OWOCOWANIA

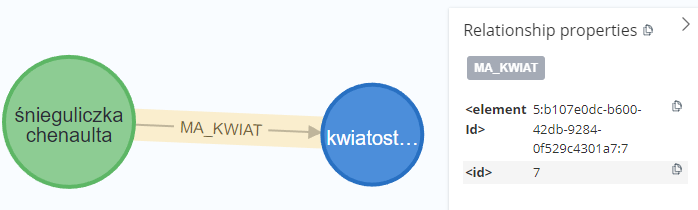
Połączenie kierunku Roslina -> Okres. Wskazuje miesiące, w której roślina owocuje.



Obraz 1.3.25 Relacja MA\_OKRES\_OWOCOWANIA

MA\_KWIAT

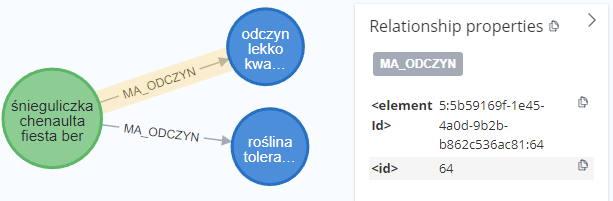
Połączenie kierunku Roslina -> Kwiat. Wskazuje rodzaj kwiatów, jakie ma roślina.



Obraz 1.3.26 Relacja MA\_KWIAT

MA\_ODCZYN

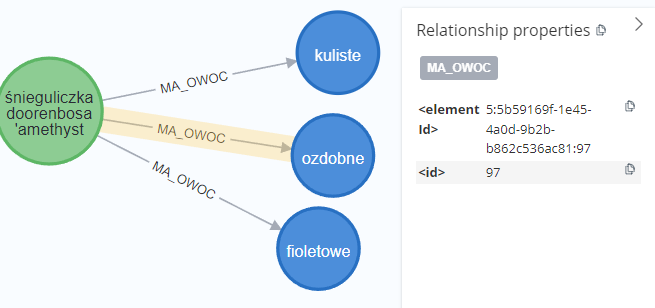
Połączenie kierunku Roslina -> Odczyn. Wskazuje odczyn gleby, w jakich roślina powinna być sadzona.



Obraz 1.3.27 Relacja MA\_ODCZYN

MA\_OWOC

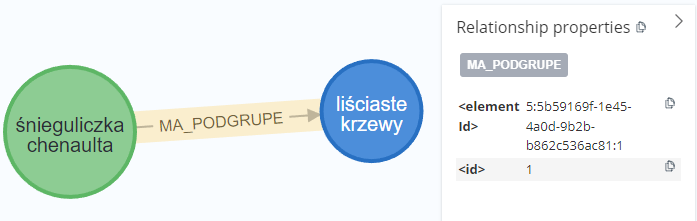
Połączenie kierunku Roslina -> Owoc. Wskazuje rodzaj, kształt albo kolor owocu, jaki ma dana roślina.



Obraz 1.3.28 Relacja MA\_OWOC

MA\_PODGRUPE

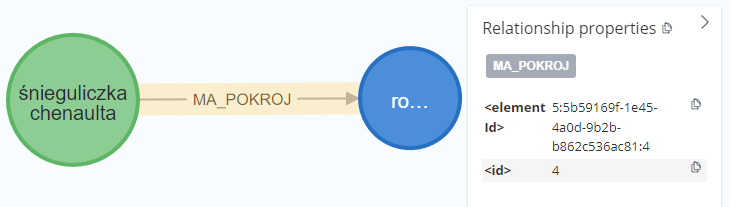
Połączenie kierunku Roslina -> Podgrupa. Wskazuje podgrupę, do której należy roślina.



Obraz 1.3.29 Relacja MA\_PODGRUPE

MA\_POKROJ

Połączenie kierunku Roslina -> Pokroj. Wskazuje pokrój rośliny.

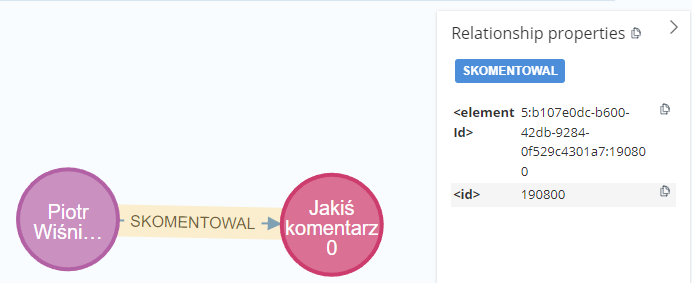


Obraz 1.3.30 Relacja MA\_POKROJ

###### Relacje społecznościowe

SKOMENTOWAL

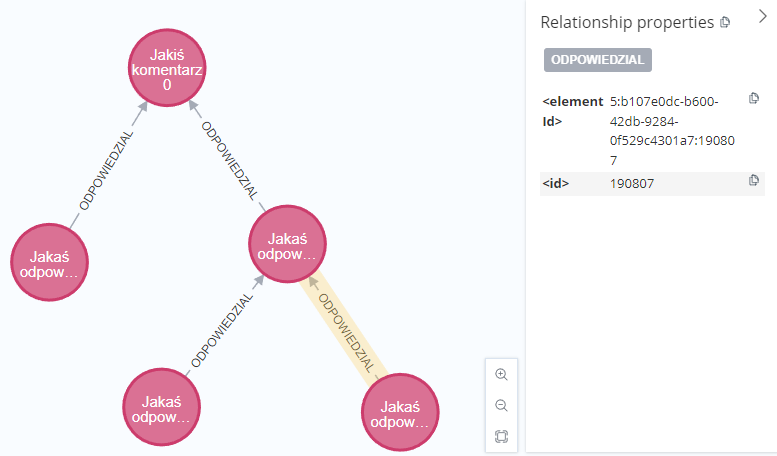
Połączenie kierunku Uzytkownik -> Komentarz. Wskazuje komentarz, który utworzył użytkownik.



Obraz 1.3.31 Relacja SKOMENTOWAL

ODPOWIEDZIAL

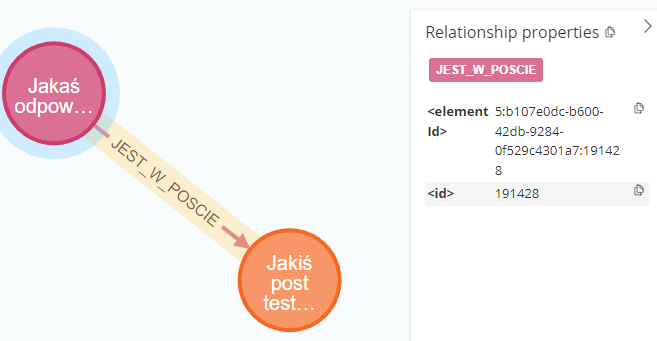
Połączenie kierunku Komentarz -> Komentarz. Wskazuje komentarz, na który odpowiedział użytkownik.



Obraz 1.3.32 Relacja ODPOWIEDZIAL

JEST\_W\_POSCIE

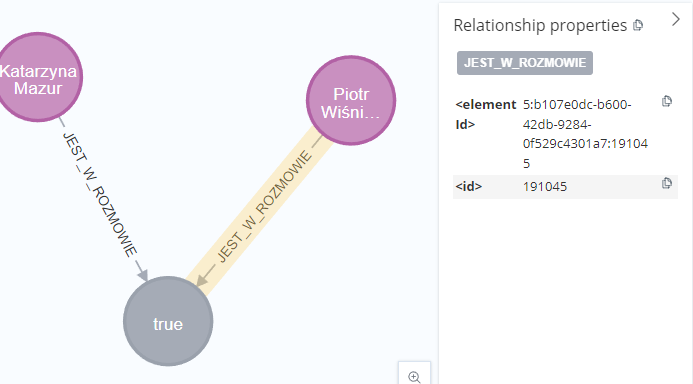
Połączenie kierunku Komentarz -> Post. Wskazuje post, który zawiera wybrany komentarz. Używana głównie do wyliczania ilości komentarzy w poście.



Obraz 1.3.33 Relacja JEST\_W\_POSCIE

JEST\_W\_ROZMOWIE

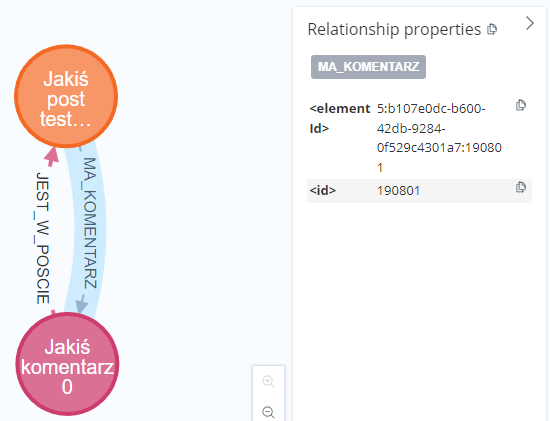
Połączenie kierunku Uzytkownik -> RozmowaPrywatna. Wskazuje rozmowę, w której uczestniczy użytkownik.



Obraz 1.3.34 Relacja JEST\_W\_ROZMOWIE

MA\_KOMENTARZ

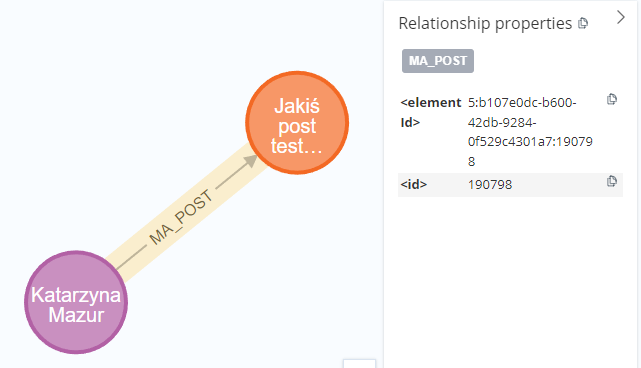
Połączenie kierunku Post -> Komentarz. Wskazuje komentarz, który komentuje dany post.



Obraz 1.3.35 Relacja MA\_KOMENTARZ

MA\_POST

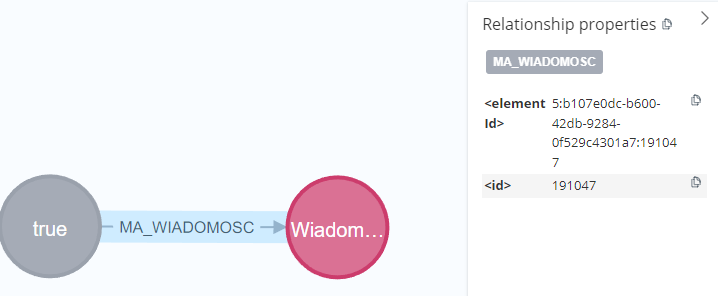
Połączenie kierunku Uzytkownik -> Post. Wskazuje post, który został stworzony przez użytkownika.



Obraz 1.3.36 Relacja MA\_POST

MA\_WIADOMOSC

Połączenie kierunku RozmowaPrywatna -> Komentarz. Wskazuje komentarz, który jest w rozmowie prywatnej.



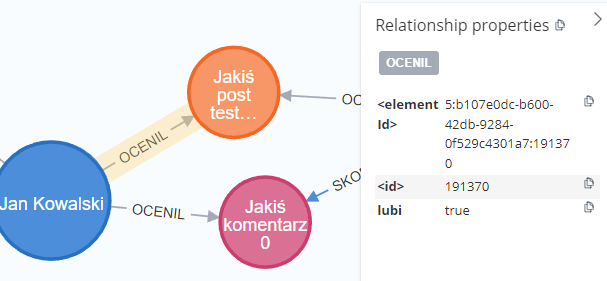
Obraz 1.3.37 Relacja MA\_WIADOMOSC

OCENIL

Połączenie kierunku Uzytkownik -> Post albo Uzytkownik -> Komentarz. Wskazuje węzeł, który został oceniony przez użytkownika. Po dodaniu oceny, może ona być zmieniana z pozytywnej na negatywną i vice versa.

**Właściwości:**

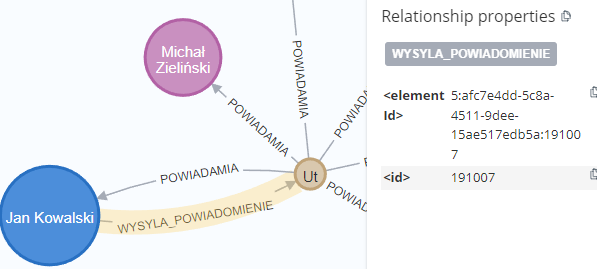
* lubi – ocena pozytywna albo negatywna



Obraz 1.3.38 Relacja OCENIL

WYSYLA\_POWIADOMIENIE

Połączenie kierunku Uzytkownik -> Powiadomienie. Wskazuje powiadomienie, które jest wysyłane przez użytkownika. Używane przy tworzeniu powiadomień globalnych (np. powiadomienie pracowników o nowych wytycznych, czy powiadomienie wszystkich użytkowników o czymś ważnym).



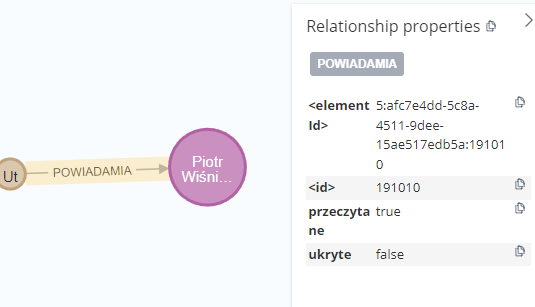
Obraz 1.3.39 Relacja WYSYLA\_POWIADOMIENIE

POWIADAMIA

Połączenie kierunku Powiadomienie -> Uzytkownik. Wskazuje użytkownika, który jest powiadamiany.

**Właściwości:**

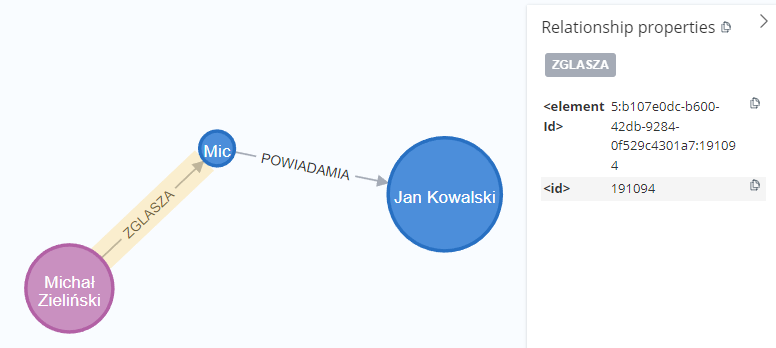
* przeczytane – oznacza, czy powiadomienie zostało przeczytane przez danego użytkownika
* ukryte – oznacza, czy powiadomienie zostało ukryte przez danego użytkownika. Stosowane wobec zgłoszeń i specjalnych powiadomień



Obraz 1.3.40 Relacja POWIADAMIA

ZGLASZA

Połączenie kierunku Uzytkownik -> Powiadomienie. Wskazuje zgłoszenie, które wysłał użytkownik do pracownika.



Obraz 1.3.41 Relacja ZGLASZA

###### Relacje użytkownika

MA\_USTAWIENIA

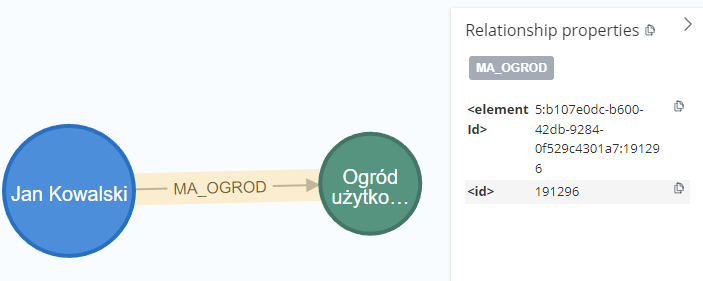
Połączenie kierunku Uzytkownik -> Ustawienia. Wskazuje ustawienia użytkownika.



Obraz 1.3.42 Relacja MA\_USTAWIENIA

MA\_OGROD

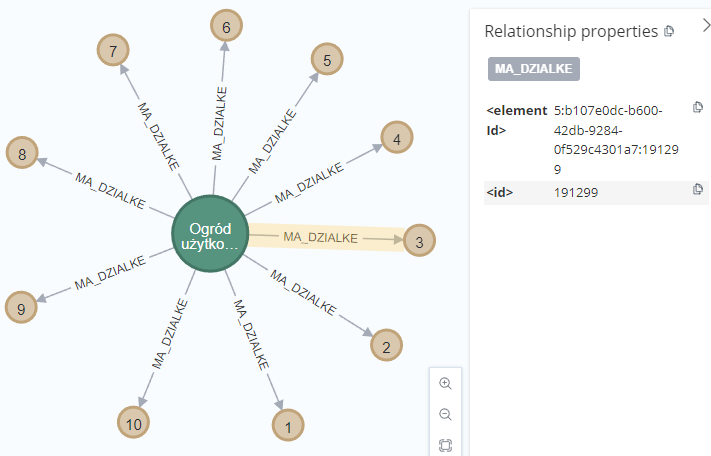
Połączenie kierunku Uzytkownik -> Ogrod. Wskazuje ogród użytkownika.



Obraz 1.3.43 Relacja MA\_OGROD

MA\_DZIALKE

Połączenie kierunku Ogrod -> Dzialka. Wskazuje działkę ogrodu użytkownika.



Obraz 1.3.44 Relacja MA\_DZIALKE

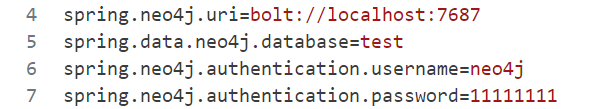
## Uruchomienie aplikacji

Przed uruchomieniem aplikacji należy wykonać parę kroków:

### Konfiguracja backendu

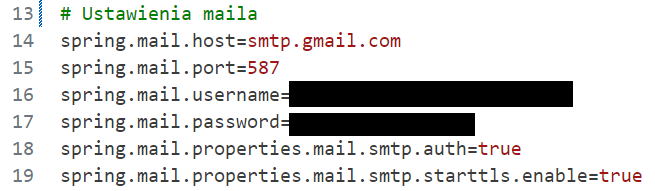
Na początku, należy w pliku *application.properties* podać kolejno parametry:

* Adres bazy
* Nazwę bazy
* Nazwę użytkownika
* Hasło



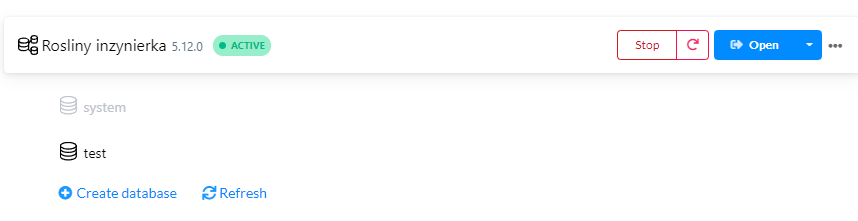
Obraz 2.1.1 Parametry bazy danych

Potrzebna jest również konfiguracja maila. Poniżej wystarczy podać odpowiednie parametry do wybranego konta, które będzie wysyłało maile dotyczące zmiany hasła, adresu email, albo aktywacji konta.



Obraz 2.1.2 Parametry maila

Po skonfigurowaniu bazy i maila, należy przejść do bazy danych. W programie Neo4j desktop albo w przeglądarce należy uruchomić wybraną bazę:



Obraz 2.1.3 Uruchomiona baza danych

Backend Spring Boot musi mieć zainstalowane odpowiednie zależności. Można to zrobić poprzez przejście do folderu yukka i wpisanie komendy:

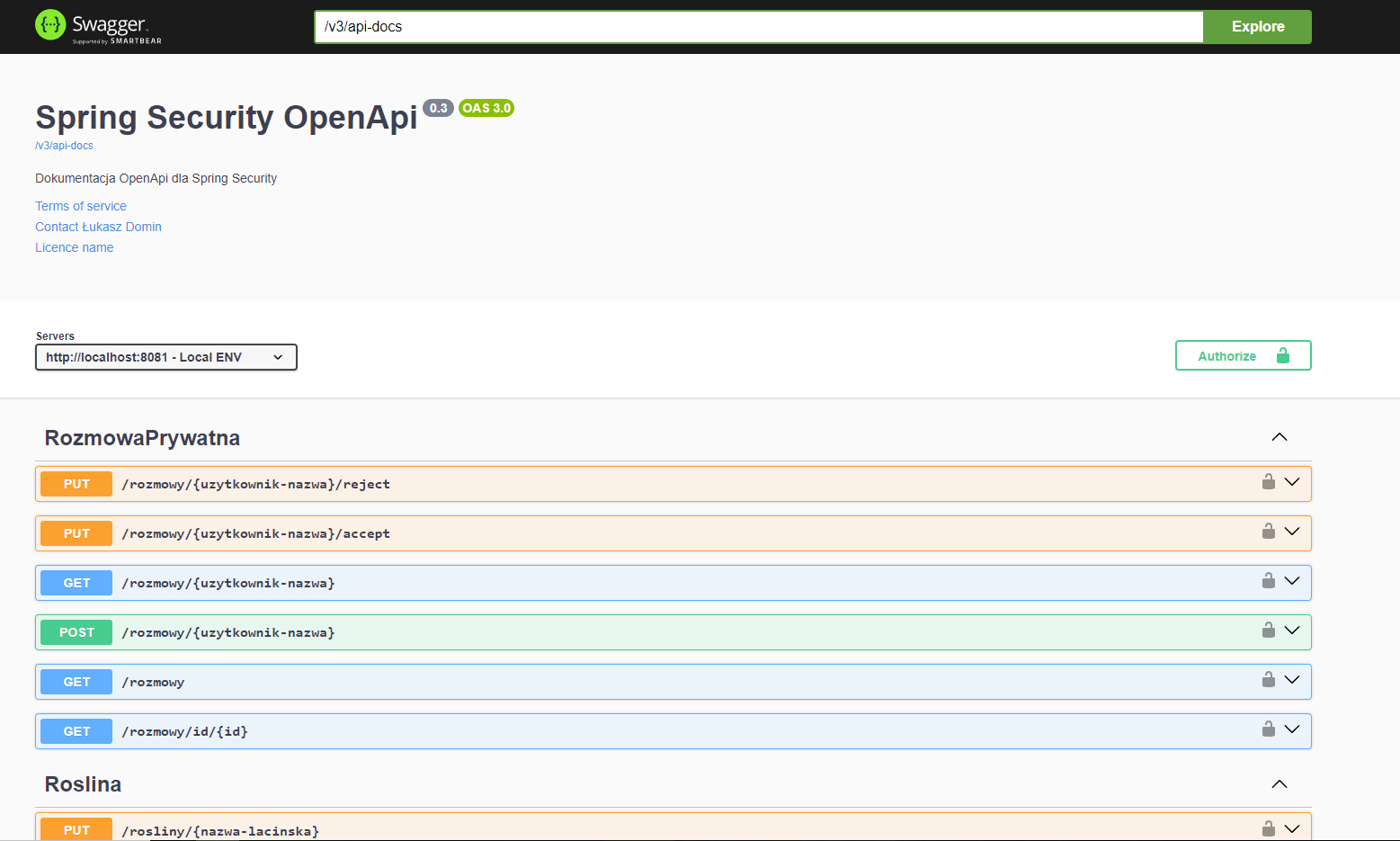
mvn clean install -D seed.database=true

Jeśli operacja wykona się poprawnie, zależności aplikacji zostaną zainstalowane, a sama baza danych zostanie zaseedowana. Jeśli ostatnia część nie jest potrzebna, można wykonać tą komendę bez argumentu lub zamieniając jego wartość na *false.*

Po instalacji, należy wykonać następującą komendę, aby uruchomić aplikację:

mvn spring-boot:run

Po uruchomieniu, można zobaczyć dokumentację OpenApi przechodząc do adresu <http://localhost:8081/swagger-ui/index.html>



Obraz 2.1.4 Widok Swaggera

W tym adresie jest pokazana dokumentacja OpenApi dla Spring Security. Zawiera ona wszystkie endpointy, ich parametry, requesty i przykładowe odpowiedzi. Jest ona szczególnie ważna do komunikacji z frontendem.

### Konfiguracja frontendu

Przejdź do folderu *yukka-ui* i zainstaluj wymagane pakiety do frontendu, wykonując komendę:

npm i

Po zainstalowaniu pakietów, możesz uruchomić serwer na frontendzie używając jednej z tych komend:

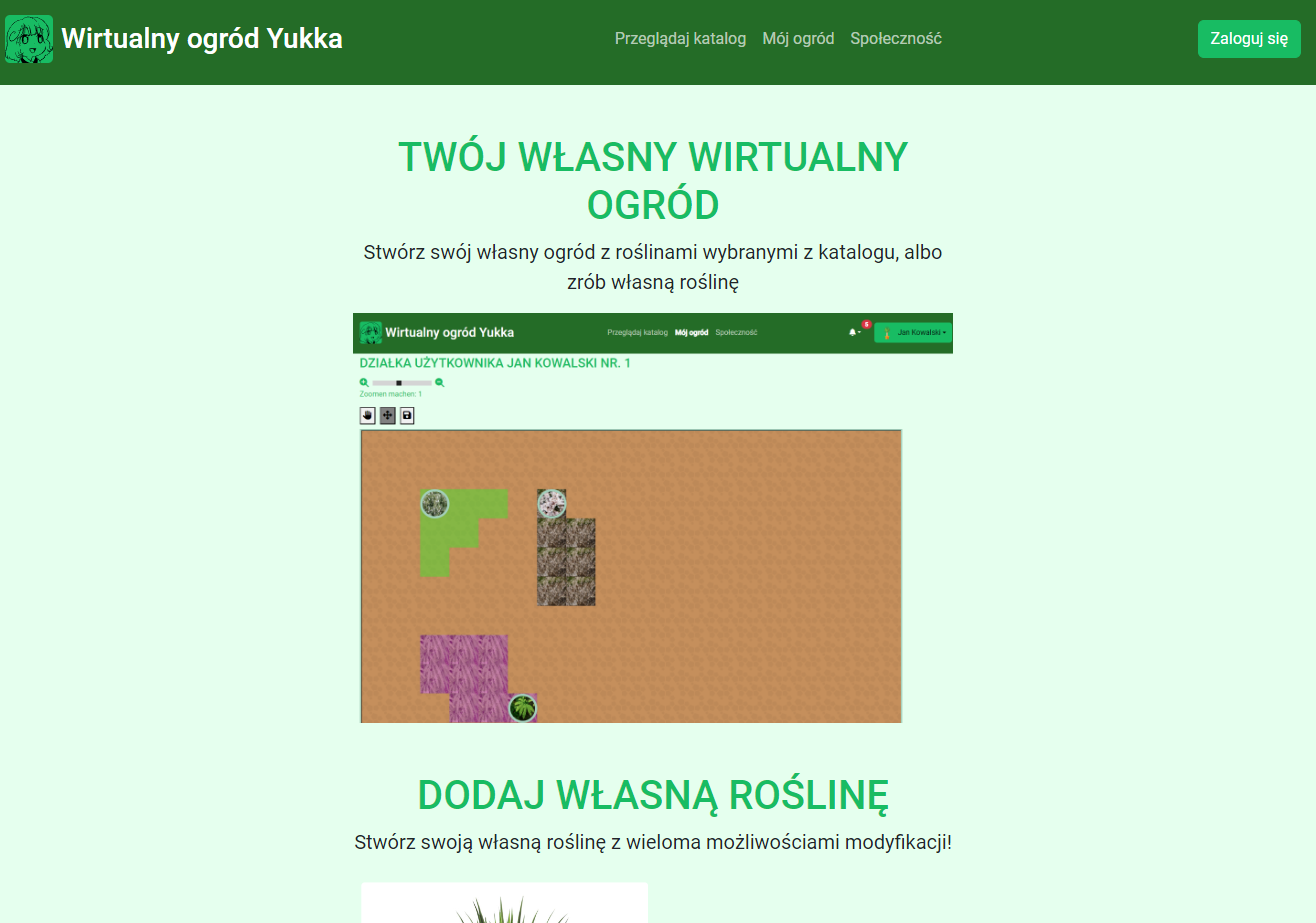
ng serve

Albo

npm run start

Po uruchomieniu serwera, należy przejść pod ten adres:

<http://localhost:4200>

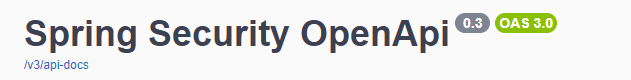


Obraz 2.2.1 Widok strony głównej

### Aktualizacja danych pomiędzy backendem a frontendem

Jeśli na backendzie były wprowadzane jakieś zmiany (Modyfikacja obiektu, dodanie nowego frontentu itp.), należy go zresetować i przejść do dokumentacji OpenApi: <http://localhost:8081/swagger-ui/index.html>,

Następnie przejść pod ten adres [/v3/api-docs](http://localhost:8081/v3/api-docs):

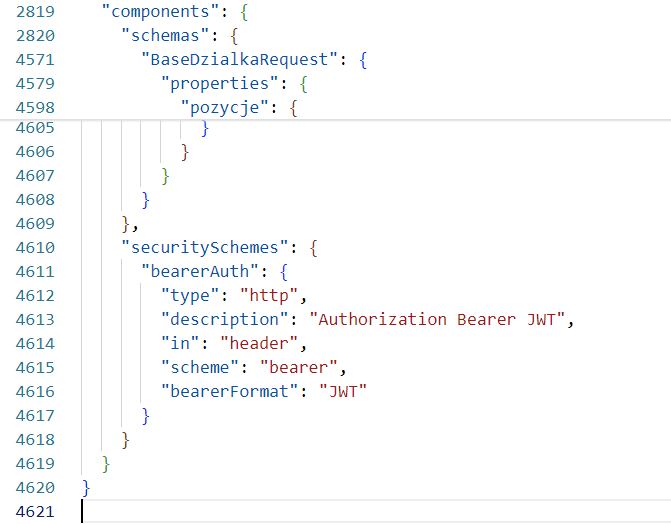


Obraz 2.3.1 Link do api-docs

Ten adres zwraca JSON zawierający informacje o dokumentacji OpenApi. Należy zastosować formatowanie stylistyczne i skopiować go CTRL + A:



Następnie należy przejść do pliku na frontendzie pod ścieżką *yukka-ui/src/app/openApi/openApi.json* i wkleić zawartość:



Obraz 2.3.2 Fragment pliku openApi.json

Następnie będąc w folderze yukka-ui wykonać komendę:

npm run api-gen

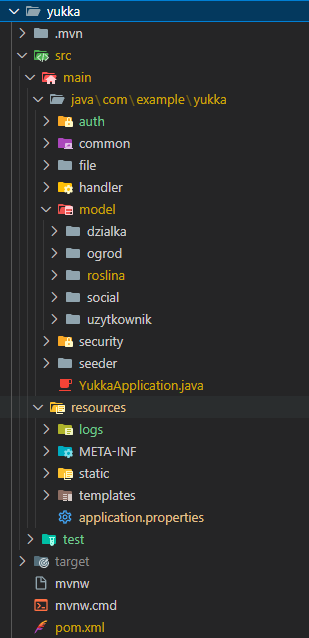
Ta komenda używając pakietu *ng-openapi-gen* wygeneruje modele odwzorowane na podstawie wklejonej dokumentacji i przerabia je tak, aby można było używać ich na frontendzie. Przykładowo, zamiast ręcznie pisać obiekt rośliny, request i funkcję obsługującą żądanie wyszukiwania rośliny, generator robi to za programistę, oszczędzając czas.

## Przedstawienie aplikacji(TODO)

### Struktura backendu

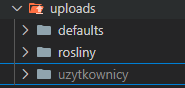
Backend jest projektem Spring Boot mający za zadanie odbieranie żądań z frontendu wysyłanych przez REST, walidację danych, a następnie wysyłanie zapytań do bazy danych i dostarczenie odpowiedzi w formacie JSON z powrotem do frontendu.

Podzielony jest on na część programową i część testową. W części testowej znajdują się testy jednostkowe służące do sprawdzania działania poszczególnych elementów programu, a w części programowej znajduje się faktyczna logika omawianego projektu.



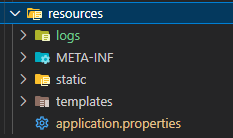
Obraz 3.1.1 Struktura backendu

* Auth – Część programu zajmująca się autoryzacją użytkownika i wykonywaniu związanych z nią operacji, takich jak logowanie, rejestracja, wysyłanie wiadomości email do potwierdzenia, czy też usuwania konta.
* Common – Folder z obiektami często używanymi w projekcie, np. odpowiedź zawierająca listę obiektów z dołączoną paginacją.
* File – Część programu zajmująca się odczytywaniem, aktualizacją i zapisywaniem plików oraz folderów, które znajdują się w zdefiniowanym katalogu uploads zawierający pliki domyślne, rośliny z katalogu, oraz foldery użytkowników wraz z należącymi do nich plikami.



Obraz 3.1.2 Katalog uploads

* Handler – Część programu zajmująca się przetwarzaniem przeróżnych błędów, jakie wystąpiły podczas działania programu i przekazanie odpowiedzi do użytkownika na frontendzie.
* Model – Część programu, gdzie znajdują się obiekty reprezentujące poszczególne elementy z bazy danych, a także odpowiadające im kontrolery, serwisy, repozytoria, żądania, obiekty mapujące i przede wszystkim odpowiedzi.
  + Dzialka – Część programu odpowiadająca za obsługę działek.
  + Ogrod – Część programu odpowiadająca za obsługę ogrodu.
  + Roslina – Część programu odpowiadająca za obsługę roślin.
  + Social – Część programu odpowiadająca za obsługę części społecznościowej, gdzie znajdują się posty, komentarze, oceny, rozmowy prywatne i powiadomienia.
  + Uzytkownik – Część programu odpowiadająca za obsługę użytkownika i usług z nim związanych.
* Security – Część programu, gdzie ustawiane są zabezpieczenia dotyczące wykonywania żądań. Przede wszystkim ustawia się uprawnienia do wykonywania żądań, skąd te żądania mogą być nadawane, jakie elementy żądania powinny być zezwolone, jakie filtry należy zastosować i ile razy można maksymalnie wykonywać żądania w danym odstępie czasu.
* Seeder – Część programu odpowiadająca za seedowanie (zaludnienie) bazy danych. Operacja ta jest wykowywana, jeśli użytkownik przy uruchamianiu programu ustawił seed-database=true.
* YukkaApplication – Główna klasa aplikacji.
* Resources – Folder, w którym znajdują się logi, pliki .html wysyłane jako wiadomość email, a także application.properties, w którym znajdują się zmienne z ustawieniami aplikacji, takie jak port serwera, ustawienia maila, konfiguracja tokena jwt, czy ścieżki plików.



Obraz 3.1.3 Katalog resources

* Pom.xml – Plik, w którym znajdują się zależności programu

#### Przykład obiektu

Poniżej znajduje się obiekt Post reprezentujący węzeł Post z bazy danych.

/\*\*

 \* Reprezentuje post.

 \*

 \* <ul>

 \*   <li><strong>id</strong>: Unikalny identyfikator posta.</li>

 \*   <li><strong>postId</strong>: Identyfikator posta jako uuid.</li>

 \*   <li><strong>tytul</strong>: Tytuł posta.</li>

 \*   <li><strong>opis</strong>: Opis posta.</li>

 \*   <li><strong>obraz</strong>: URL obrazu powiązanego z postem.</li>

 \*   <li><strong>dataUtworzenia</strong>: Data utworzenia posta.</li>

 \*   <li><strong>autor</strong>: Użytkownik, który stworzył post.</li>

 \*   <li><strong>komentarze</strong>: Lista komentarzy powiązanych z postem.</li>

 \*   <li><strong>komentarzeWPoscie</strong>: Lista komentarzy zawartych w poście.</li>

 \* </ul>

 \*/

@Node

@Getter

@Setter

@ToString

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class Post extends Oceniany {

    @Id @GeneratedValue

    private Long id;

    @Property(name = "postId")

    private String postId;

    @Property(name = "tytul")

    private String tytul;

    @Property(name = "opis")

    private String opis;

    @Property(name = "obraz")

    private String obraz;

    @CreatedDate

    @Property(name = "dataUtworzenia")

    private LocalDateTime dataUtworzenia;

    @Relationship(type = "MA\_POST", direction = Relationship.Direction.INCOMING)

    private Uzytkownik autor;

    @Relationship(type = "MA\_KOMENTARZ", direction = Relationship.Direction.OUTGOING)

    private List<Komentarz> komentarze;

    @Relationship(type = "JEST\_W\_POSCIE", direction = Relationship.Direction.INCOMING)

      private List<Komentarz> komentarzeWPoscie;

}

Obiekt posiada adnotacje, które nadają mu różne właściwości przy równoczesnym skróceniu kodu:

* Node – Oznacza, że jest to Węzeł z bazy danych o etykiecie takiej samej, jak nazwa obiektu.
* @Getter – Dodaje Gettery do każdej właściwości posta. Gettery to funkcje zwracające daną właściwość.
* @Setter – Dodaje Settery do każdej właściwości posta. Gettery to funkcje modyfikujące daną właściwość.
* @ToString – Dodaje przeciążenie funkcji toString(), zwracająca informacje o obiekcie w formie stringa.
* @NoArgsConstructor – Dodaje konstruktor bezargumentowy obiektu.
* @AllArgsConstructor – Dodaje konstruktor ze wszystkimi argumentami obiektu.
* @Builder – Dodaje builder, który pozwala na „budowanie” obiektu z wybranymi parametrami bez konieczności tworzenia wszystkich kombinacji konstruktorów.
* @Id, @GeneratedValue – Oznajmia, że jest to pole id węzła, które jest generowane dla każdego obiektu mapowanego z bazy danych.
* @Property – Wskazuje nazwę właściwości obiektu z bazy danych, pozwalając na mapowanie do zmiennej.
* @CreatedDate – Data utworzenia.
* @Relationship – Wskazuję relację danego typu o kierunku od/do.

Dodatkowo, obiekt ten jest dzieckiem obiektu Ocenialny, który jest stosowany do węzłów mogących zostać ocenionymi (Post, Komentarz). Poprzez bycie jego dzieckiem, odziedzicza on jego właściwości. Poniżej znajduje się omawiany rodzic:

/\*\*

 \* Klasa <strong>Oceniany</strong> reprezentuje obiekt oceniany. Dziedziczą je obiekty Post i Komentarz.

 \*

 \* <ul>

 \* <li><strong>ocenil</strong> - lista obiektów typu <strong>OcenilReverse</strong> reprezentujących użytkowników którzy to ocenili.</li>

 \* </ul>

 \*

 \* Metody:

 \* <ul>

 \* <li><strong>getOcenyLubi</strong> - zwraca liczbę ocen pozytywnych.</li>

 \* <li><strong>getOcenyNieLubi</strong> - zwraca liczbę ocen negatywnych.</li>

 \* </ul>

 \*/

@Getter

@Setter

@ToString

@NoArgsConstructor

@AllArgsConstructor

public class Oceniany {

    @Relationship(type = "OCENIL", direction = Relationship.Direction.INCOMING)

    private List<OcenilReverse> ocenil;

    public int getOcenyLubi() {

        return (int) ocenil.stream().filter(OcenilReverse::isLubi).count();

    }

    public int getOcenyNieLubi() {

        return (int) ocenil.stream().filter(ocenil -> !ocenil.isLubi()).count();

    }

}

Obiekt ten posiada funkcje zwracające ilość ocen pozytywnych i negatywnych, a także ma relację OcenilReverse, która wskazuje na obiekt pośredni będący relacją z właściwościami zawierająca wskazywany węzeł:

/\*\*

 \* Klasa reprezentująca relację ocenienia w odwrotnym kierunku.

 \*

 \* <ul>

 \*   <li><strong>id</strong>: Unikalny identyfikator relacji.</li>

 \*   <li><strong>lubi</strong>: Wartość logiczna określająca, czy użytkownik lubi dany element.</li>

 \*   <li><strong>uzytkownik</strong>: Obiekt reprezentujący użytkownika, który ocenił.</li>

 \* </ul>

 \*/

@RelationshipProperties

@Getter

@Builder

@AllArgsConstructor

@NoArgsConstructor

public class OcenilReverse {

    @Id @GeneratedValue

    private Long id;

    @Property("lubi")

    private boolean lubi;

    @TargetNode

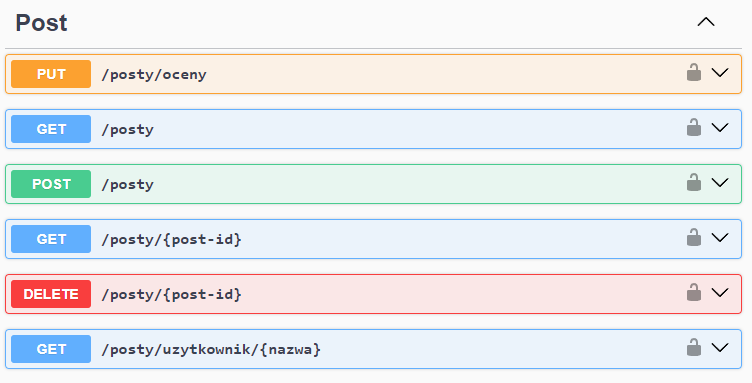
    private Uzytkownik uzytkownik;

}

Warto zauważyć, iż ten obiekt posiada adnotację @RelationshipProperties, która oznacza, że jest to relacja ze zdefiniowanymi właściwościami. W tym wypadku tą właściwością oprócz id jest właściwość lubi, a sama relacja posiada wewnątrz siebie węzeł Uzytkownik, na którego wskazuje.

#### Przykład procesu żądania i odpowiedzi

Aby wykonał jakąś operacje, musi on wysłać żądanie do serwera na backendzie. Poniżej znajduje się widok możliwych żądań dla Kontrolera o swaggerowym tagu „Post”:

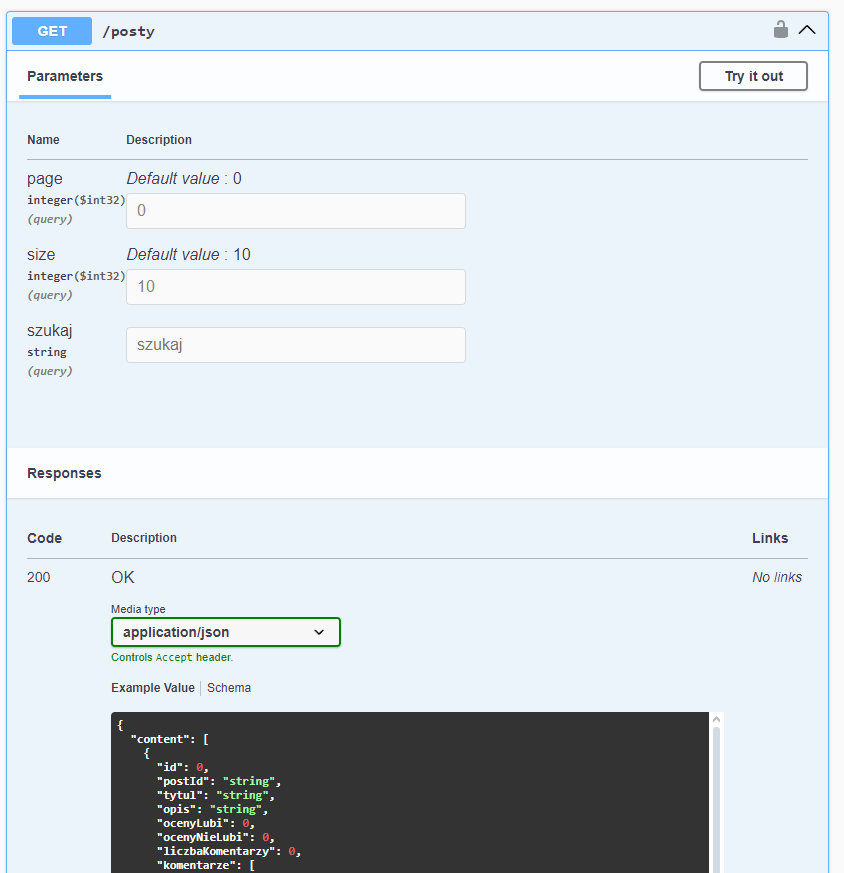


Obraz 3.1.4 Widok kontrolera Postu (Swagger)

Kontroler posiada następujące ścieżki:

* /posty/oceny PUT – Żądanie dodające lub aktualizujące ocenę postu.
* /posty GET – Żądanie zwracające listę postów przy użyciu paginacji.
* /posty POST – Żądanie dodające post.
* /posty/{post-id} GET – Żądanie zwracające post o podanym id.
* /posty/{post-id} DELETE – Żądanie usuwające post o podanym id.
* /posty/uzytkownik/{nazwa} – Żądanie zwracające posty użytkownika o danej nazwie przy użyciu paginacji.

Załóżmy, że użytkownik chce pobrać sobie wszystkie posty. Do tego zadania należy użyć metody GET /posty. Poniżej znajduje się widok ze Swaggera:



Obraz 3.1.5 Widok żądania GET /posty

Do żądania można opcjonalnie podać następujące parametry:

* page(strona) o domyślnej wartości 0
* size(ilość zwracanych elementów na stronę) o domyślnej wartości 10
* szukaj(nazwa postu).

Przykładowa odpowiedź dla tego żądania zawiera listę postów wraz z takimi informacjami jak aktualna strona, całkowitą ilość stron, całkowitą ilość elementów, oraz czy jest to pierwsza/ostatnia strona:

**{**

**"content": [**

**{**

**"id": 0,**

**"postId": "string",**

**"tytul": "string",**

**"opis": "string",**

**"ocenyLubi": 0,**

**"ocenyNieLubi": 0,**

**"liczbaKomentarzy": 0,**

**"komentarze": [**

**{**

**"id": 0,**

**"komentarzId": "string",**

**"opis": "string",**

**"edytowany": true,**

**"ocenyLubi": 0,**

**"ocenyNieLubi": 0,**

**"dataUtworzenia": "string",**

**"post": "string",**

**"odpowiadaKomentarzowi": "string",**

**"odpowiedzi": [**

**"string"**

**],**

**"uzytkownikNazwa": "string",**

**"obraz": "string",**

**"avatar": "string"**

**}**

**],**

**"uzytkownik": "string",**

**"obraz": "string",**

**"avatar": "string",**

**"dataUtworzenia": "string"**

**}**

**],**

**"number": 0,**

**"size": 0,**

**"totalElements": 0,**

**"totalPages": 0,**

**"first": true,**

**"last": true**

**}**

Samo żądanie w kodzie wygląda tak:

    /\*\*

     \* Metoda obsługująca żądanie GET do wyszukania wszystkich postów.

     \*

     \* @param page numer strony wyników, domyślnie 0

     \* @param size rozmiar strony wyników, domyślnie 10

     \* @param szukaj nazwa lub opis postu

     \* @return ResponseEntity zawierające stronę wyników z postami

     \*/

    @GetMapping(produces="application/json")

    public ResponseEntity<PageResponse<PostResponse>> findAllPosty(

            @RequestParam(name = "page", defaultValue = "0", required = false) int page,

            @RequestParam(name = "size", defaultValue = "10", required = false) int size,

            @RequestParam(name = "szukaj", required = false) String szukaj) {

        return ResponseEntity.ok(postService.findAllPosts(page, size, szukaj));

    }

Kontroler wykonuje funkcję z serwisu, którą opakowuje w ciało odpowiedzi o kodzie 200 (OK). Przechodząc do serwisu można znaleźć następujący kod:

    /\*\*

     \* Znajduje wszystkie posty z możliwością paginacji i filtrowania.

     \*

     \* @param page <ul><li><strong>int</strong>: Numer strony.</li></ul>

     \* @param size <ul><li><strong>int</strong>: Rozmiar strony.</li></ul>

     \* @param szukaj <ul><li><strong>String</strong>: Wartość, po której ma być filtrowana lista postów.</li></ul>

     \* @return <ul><li><strong>PageResponse(PostResponse)</strong>: Odpowiedź zawierająca listę postów oraz dane paginacji.</li></ul>

     \*/

    @Transactional(readOnly = true)

    public PageResponse<PostResponse> findAllPosts(int page, int size, String szukaj) {

        Pageable pageable = PageRequest.of(page, size, Sort.by("post.dataUtworzenia").descending());

        Page<Post> posts = postRepository.findAllPosts(szukaj, pageable);

        return postMapper.postResponsetoPageResponse(posts);

    }

Serwis używa metody z repozytorium, aby zwróciło odpowiedź zawierającą daną stronę z paginacji postów, a następnie mapuje je odpowiednio do odpowiedzi, którą zwraca. Przechodząc do repozytorium, można znaleźć następujące zapytanie:

    @Query(value = """

        MATCH path = (post:Post)<-[:MA\_POST]-(:Uzytkownik)

        OPTIONAL MATCH path2 = (post)-[:OCENIL]-(:Uzytkownik)

        OPTIONAL MATCH path3 = (post)<-[:JEST\_W\_POSCIE]-(kom:Komentarz)

        WHERE $szukaj IS NULL OR post.tytul CONTAINS $szukaj OR post.opis CONTAINS $szukaj

        RETURN post, collect(nodes(path)), collect(relationships(path)),

                collect(nodes(path2)), collect(relationships(path2)),

                collect(nodes(path3)), collect(relationships(path3))

        :#{orderBy(#pageable)} SKIP $skip LIMIT $limit

        """,

       countQuery = """

        MATCH (post:Post)<-[:MA\_POST]-(:Uzytkownik)

        WHERE $szukaj IS NULL OR post.tytul CONTAINS $szukaj OR post.opis CONTAINS $szukaj

        RETURN count(post)

        """)

    Page<Post> findAllPosts(@Param("szukaj") String szukaj, Pageable pageable);

Zapytanie sformułowane w języku CYPHER przyjmuje parametr wyszukujący i obiekt mapujący wynik na stronę w paginacje, a następnie wysyła go do bazy danych, zwracając odpowiedź lub błąd. Samo zapytanie jest podzielone na dwa zapytania, gdyż dla obiektu Page wymagane jest, aby podano w osobnym zapytaniu całkowitą liczbę elementów z zapytania.

Po wykonaniu tego wszystkiego, otrzymuje się odpowiedź. Poniżej znajduje się przykładowa odpowiedź z parametrem size=2 (Zawartość obrazów pominięto ze względu na to, że są one tablicami bajtów o dużej długości):

**{**

**"content": [**

**{**

**"id": 7615,**

**"postId": "760b84e6-2d16-4abf-9d11-e0951d6e4c0f",**

**"tytul": "Ut molestiae id quisquam.",**

**"opis": "Est deserunt dolor aspernatur. Molestias in nobis iste maiores dicta. Debitis porro laudantium dolores quibusdam autem reprehenderit pariatur.",**

**"ocenyLubi": 4,**

**"ocenyNieLubi": 1,**

**"liczbaKomentarzy": 4,**

**"komentarze": [],**

**"uzytkownik": "Anna Nowak",**

**"avatar": "…"**

**"dataUtworzenia": "3 dni temu"**

**},**

**{**

**"id": 7603,**

**"postId": "2870b5c0-0962-4a37-ad4b-fd9634c35765",**

**"tytul": "Optio voluptate ut.",**

**"opis": "Est eaque eligendi quo in iusto nihil. Impedit dolorum assumenda voluptatem reprehenderit repellendus architecto. Occaecati quo quia sint optio dolor unde atque. Aut qui architecto. Odio quia molestiae ipsa ea.",**

**"ocenyLubi": 0,**

**"ocenyNieLubi": 2,**

**"liczbaKomentarzy": 6,**

**"komentarze": [],**

**"uzytkownik": "Jan Kowalski",**

**"obraz": "…"**

**"avatar": "…"**

**"dataUtworzenia": "3 dni temu"**

**}**

**],**

**"number": 0,**

**"size": 2,**

**"totalElements": 5,**

**"totalPages": 3,**

**"first": true,**

**"last": false**

**}**

### Struktura frontendu

Pokazanie jak przykładowy kontroler, serwis, odpowiedź wygląda (backend, frontend)

### Główny widok aplikacji



Obraz 3.3.1 Główny widok aplikacji

### Logowanie i rejestracja

#### Logowanie

#### Proces rejestracji

### CRUD pracownika aplikacji

### Przeglądanie ogólnodostępnych zasobów

### Zarządzanie swoimi zasobami

#### Przykładowa walidacja dla tworzenia, edytowania oraz usuwania roślin

#### Przykład działania walidacji

#### Przykładowa walidacja dla dodawania, edytowania oraz usuwania rośliny na działce

TODO

#### Przykład działania walidacji

TODO

# Zakończenie

streszczenie pracy w języku polskim, tytuł pracy dyplomowej w języku polskim oraz w języku angielskim,

oświadczenie studenta o samodzielnym przygotowaniu pracy dyplomowej (załącznik do Zarządzenia w sprawie funkcjonowania procedury antyplagiatowej w Uniwersytecie Rzeszowskim).