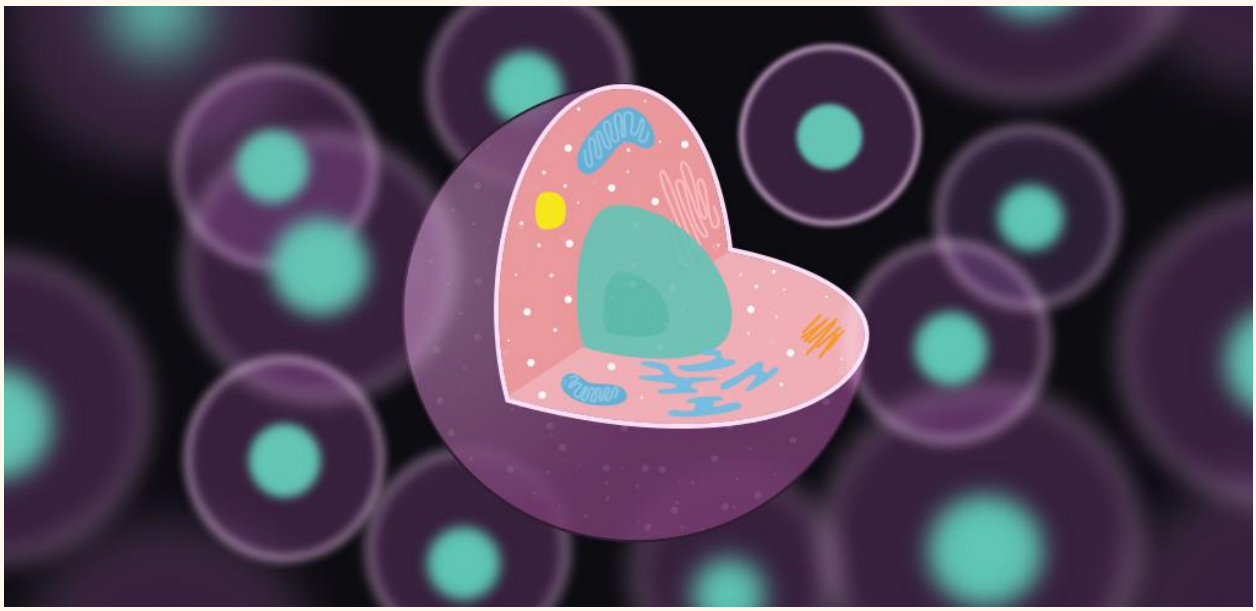
Przetwarzanie danych w chmurach obliczeniowych
18.12.2021

Projekt
Dokumentacja

—

Michał Kuś



WSTĘP

W ramach projektu należało wykonać projekt typu *"proof-of-concept"* wykorzystujący grafową bazę Neo4j AuraDB w z interfejsem dostępu stworzonym ramach dowolnej technologii internetowej. Zrealizowane zostało to za pomocą REST API zdeployowanego w chmurze GCloud oraz aplikacji wizualizującej bazę stworzonych na serwerze Django.

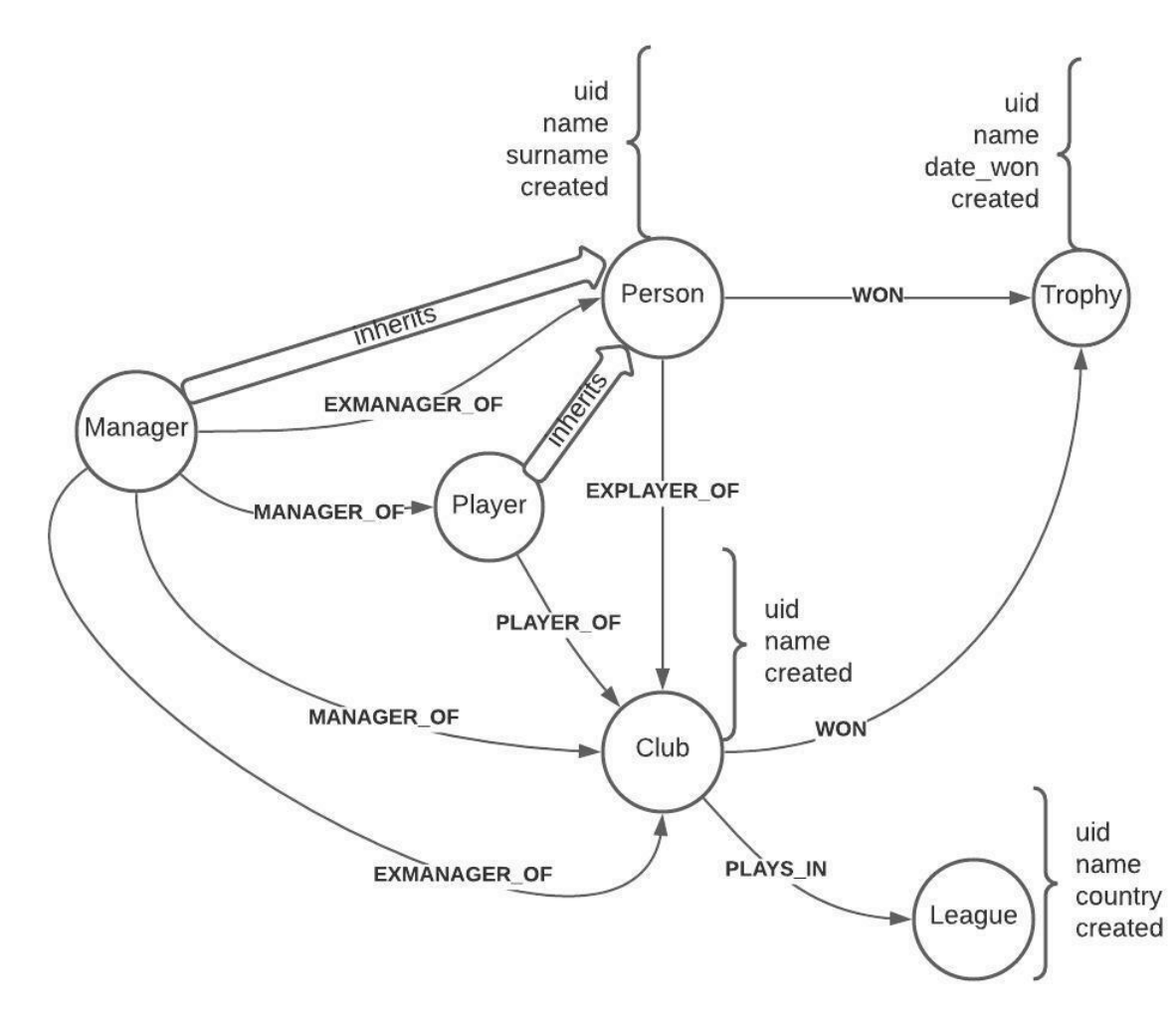
Środowisko

W realizacji projektu wykorzystane zostały:

- Django 3.2.9, w tym:
    - Django Neomodel
    - Django Rest Framework

- GCloud
- Baza Neo4j AuraDB
- Neo4j Python Driver
- Neomodel
- NeoVis.js
- VS Code

Baza Danych



Rysunek 1. Diagram utworzonej bazy danych

Baza danych wspiera 4 główne typy węzłów:

- Person
    - Player
    - Manager
- Club
- League
- Trophy

Person

- uid: string
- name: string
- surname: string
- created: datetime

Relacje:

➢ - WON -> Trophy
➢ - EXPLAYER_OF -> Club
➢ <- EXMANAGER_OF - Manager

Person:Player

Relacje:

➢ - PLAYER_OF -> Club
➢ <- MANAGER_OF - Manager

Person:Manager

Relacje:

- ➢ - EXMANAGER_OF -> Club
- ➢ - MANGER_OF -> Club
- ➢ - EXMANAGER_OF -> Person
- ➢ - MANAGER_OF -> Player

Club

- uid: string
- name: string
- created: datetime

Relacje:

- ➢ - WON -> Trophy
- ➢ - PLAYS_IN -> League
- ➢ <- EXPLAYER_OF - Person
- ➢ <- PLAYER_OF - Player
- ➢ <- EXMANAGER_OF - Manager
- ➢ <- MANAGER_OF - Manager

League

- uid: string
- name: string
- country: string
- created: datetime

Relacje:

- ➢ <- PLAYS_IN - Club

Trophy

- uid: string
- name: string
- date_won: date
- created: datetime

Relacje:

- ➢ <- WON - Person
- ➢ <- WON - Club

Modele

Do poprawnego odtworzenia tych węzłów i relacji zostały stworzone modele w Django wykorzystując moduł NeoModel, reprezentują one dane pobierane z bazy i są w stanie je serializować do JSONa za pomocą funkcji *serialize()* oraz *serialize_connections()*.

Person

- uid: UniqueIdProperty
- name: StringProperty
- surname: StringProperty
- created: DateTimeProperty

Relacje:

- ➢ trophies: RelationshipTo: Trophy, WON
- ➢ career_clubs: RelationshipTo:Club, EXPLAYER_OF

Player

Relacje:

- ➢ current_club: RelationshipTo:Club, PLAYER_OF

- manager: RelationshipFrom:Manager, MANAGER_OF

Person:Manager

Relacje:

- past_clubs: RelationshipTo:Club, EXMANAGER_OF
- managed_club:RelationshipTo:Club, MANGER_OF
- past_managed_players:RelationshipTo:Person, EXMANAGER_OF
- managed_squad:RelationshipTo:Player, MANAGER_OF

Club

- uid: UniqueIdProperty
- name: StringProperty
- created: DateTimeProperty

Relacje:

- club_trophies:RelationshipTo:Trophy, WON
- league:RelationshipTo:League, PLAYS_IN
- past_players:RelationshipFrom:Person, EXPLAYER_OF
- squad:RelationshipFrom:Player, PLAYER_OF
- past_managers:RelationshipFrom:Manager, EXMANAGER_OF
- manager:RelationshipFrom:Manager, MANAGER_OF

League

- uid: UniqueIdProperty
- name: StringProperty
- country: StringProperty

- created: DateTimeProperty

Relacje:

➢ clubs_participating:RelationshipFrom:Club, PLAYS_IN

Trophy

- uid: UniqueIdProperty
- name: StringProperty
- date_won: DateProperty
- created: DateTimeProperty

Relacje:

➢ winners_list:RelationshipFrom:Person, WON
➢ club_winners:RelationshipFrom:Club, WON

Modele mają zaimplementowane funkcje *setup_relationship()* pozwalające na łączenie węzłów ze sobą.

API

API znajduje się pod adresem [*https://djangographdb.appspot.com/api/*](https://djangographdb.appspot.com/api/)

Łączenie z bazą realizowane jest przez *NeoModel*, wykorzystujący driver *Neo4j* dla języka Python, dane logowania przechowywane są na platformie *Google Cloud* w aplikacji *Secret Manager*. Dane są serializowane i odczytywane przez funkcje pomocnicze zawarte w plikach */neo4japi/forms.py*, */neo4japi/utils.py* oraz */neo4japi/serialisers.py*

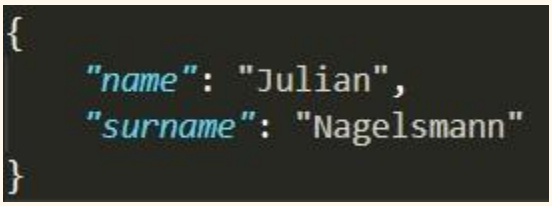
Główny plik zawierający metody API znajduje się w */neo4japi/views.py* i zawiera następujące endpointy:

```
path('people/', views.person),
path('people/<str:uid>', views.person_detail),
path('trophies/', views.trophy),
path('trophies/<str:uid>', views.trophy_detail),
path('leagues/', views.league),
path('leagues/<str:uid>', views.league_detail),
path('clubs/', views.club),
path('clubs/<str:uid>', views.club_detail),
path('players/', views.player),
path('players/<str:uid>', views.player_detail),
path('managers/', views.manager),
path('managers/<str:uid>', views.manager_detail),
path('connect/', views.connect),
```

Rysunek 2. Endpointy API w pliku /neo4japi/urls.py

/people/

- GET
  - Zwraca listę węzłów Person z limitem 50 na stronę
  - Atrybut n - filtrowanie po polu name, default = ‘’
  - Atrybut s - filtrowanie po polu surname, default = ‘’
  - Atrybut p - strona wyniku, default = 1
- POST
  - Dodaje osobę do bazy za pomocą przekazanego JSONa
  - Pola JSON:
    - ‘name’
    - ‘surname’



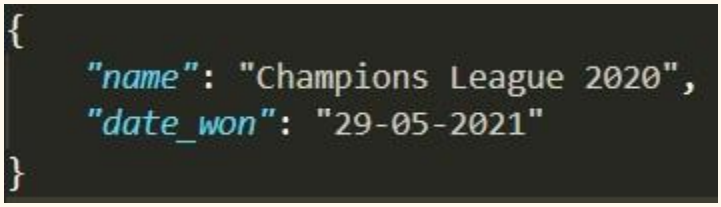
Rysunek 3. Przykładowy JSON wysłany POSTem na /people/

/people/<str:uid>

- GET
  - Pobiera z bazy atrybuty i relacje określonego węzła Person
- PUT
  - Aktualizuje w bazie atrybuty określonego węzła Person wartościami przesłanymi przez JSON(taki jak w endpoincie /people/)
- DELETE
  - Usuwa określony węzeł Person

/trophies/

- GET
  - Zwraca listę węzłów Trophy z limitem 50 na stronę
  - Atrybut n - filtrowanie po polu name, default = ‘’
  - Atrybut p - strona wyniku, default = 1
- POST
  - Dodaje trofeum do bazy za pomocą przekazanego JSONa
  - Pola JSON:
    - ‘name’
    - ‘date_won’



Rysunek 4. Przykładowy JSON wysłany POSTem na /trophies/

/trophies/<str:uid>

- GET
  - Pobiera z bazy atrybuty i relacje określonego węzła Trophy
- PUT
  - Aktualizuje w bazie atrybuty określonego węzła Trophy wartościami przesłanymi przez JSON(taki jak w endpoincie /trophies/)
- DELETE
  - Usuwa określony węzeł Trophy

/leagues/

- GET
  - Zwraca listę węzłów League z limitem 50 na stronę
  - Atrybut n - filtrowanie po polu name, default = ‘’
  - Atrybut c - filtrowanie po polu country, default = ‘’
  - Atrybut p - strona wyniku, default = 1
- POST
  - Dodaje ligę do bazy za pomocą przekazanego JSONa
  - Pola JSON:
    - ‘name’
    - ‘country’

```
{
    "name": "Serie A",
    "country": "Italy"
}
```

Rysunek 5. Przykładowy JSON wysłany POSTem na /leagues/

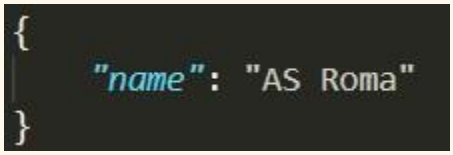
/leagues/<str:uid>

- GET
  - Pobiera z bazy atrybuty i relacje określonego węzła League
- PUT

    - Aktualizuje w bazie atrybuty określonego węzła League wartościami przesłanymi przez JSON(taki jak w endpoincie /leagues/)
- DELETE
    - Usuwa określony węzeł League

/clubs/

- GET
    - Zwraca listę węzłów Club z limitem 50 na stronę
    - Atrybut n - filtrowanie po polu name, default = ‘’
    - Atrybut p - strona wyniku, default = 1
- POST
    - Dodaje klub do bazy za pomocą przekazanego JSONa
    - Pola JSON:
        - ‘name’



Rysunek 6. Przykładowy JSON wysłany POSTem na /clubs/

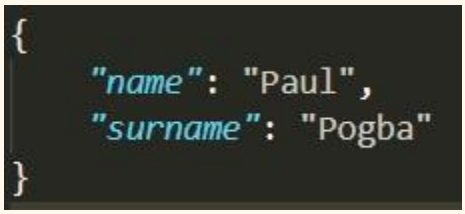
/clubs/<str:uid>

- GET
    - Pobiera z bazy atrybuty i relacje określonego węzła Club
- PUT
    - Aktualizuje w bazie atrybuty określonego węzła Club wartościami przesłanymi przez JSON(taki jak w endpoincie /clubs/)
- DELETE
    - Usuwa określony węzeł Club

/players/

- GET
    - Zwraca listę węzłów Player z limitem 50 na stronę

    - Atrybut n - filtrowanie po polu name, default = ‘’
    - Atrybut s - filtrowanie po polu surname, default = ‘’
    - Atrybut p - strona wyniku, default = 1
- POST
    - Dodaje zawodnika do bazy za pomocą przekazanego JSONa
    - Pola JSON:
        - ‘name’
        - ‘surname’



Rysunek 7. Przykładowy JSON wysłany POSTem na /players/

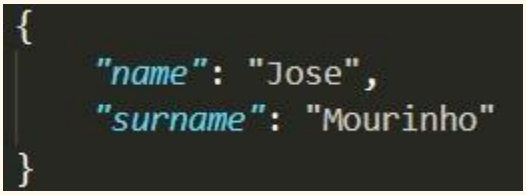
/players/<str:uid>

- GET
    - Pobiera z bazy atrybuty i relacje określonego węzła Player
- PUT
    - Aktualizuje w bazie atrybuty określonego węzła Player wartościami przesłanymi przez JSON(taki jak w endpoincie /players/)
- DELETE
    - Usuwa określony węzeł Player

/managers/

- GET
    - Zwraca listę węzłów Manager z limitem 50 na stronę
    - Atrybut n - filtrowanie po polu name, default = ‘’
    - Atrybut s - filtrowanie po polu surname, default = ‘’
    - Atrybut p - strona wyniku, default = 1
- POST
    - Dodaje trenera do bazy za pomocą przekazanego JSONa
    - Pola JSON:

- ‘name’
- ‘surname’



Rysunek 8. Przykładowy JSON wysłany POSTem na /managers/

/managers/<str:uid>

- GET
  - Pobiera z bazy atrybuty i relacje określonego węzła Manager
- PUT
  - Aktualizuje w bazie atrybuty określonego węzła Manager wartościami przesłanymi przez JSON(taki jak w endpoincie /managers/)
- DELETE
  - Usuwa określony węzeł Manager

/connect/

- POST
  - Łączy ze sobą dwa węzły określonym typem relacji za pomocą JSONa
  - Pola JSON:
    - “start_node_type”
    - “start_node_uid”
    - “end_node_type”
    - “end_node_uid”
    - “rel_type”

```
{
    "start_node_type": "Person",
    "start_node_uid": "f15894cf1ace44e0b212d50eac6f3208",
    "end_node_type": "Club",
    "end_node_uid": "0bf8ffba09b34d6abad7840aa85cd9fe",
    "rel_type": "career_clubs"
}
```

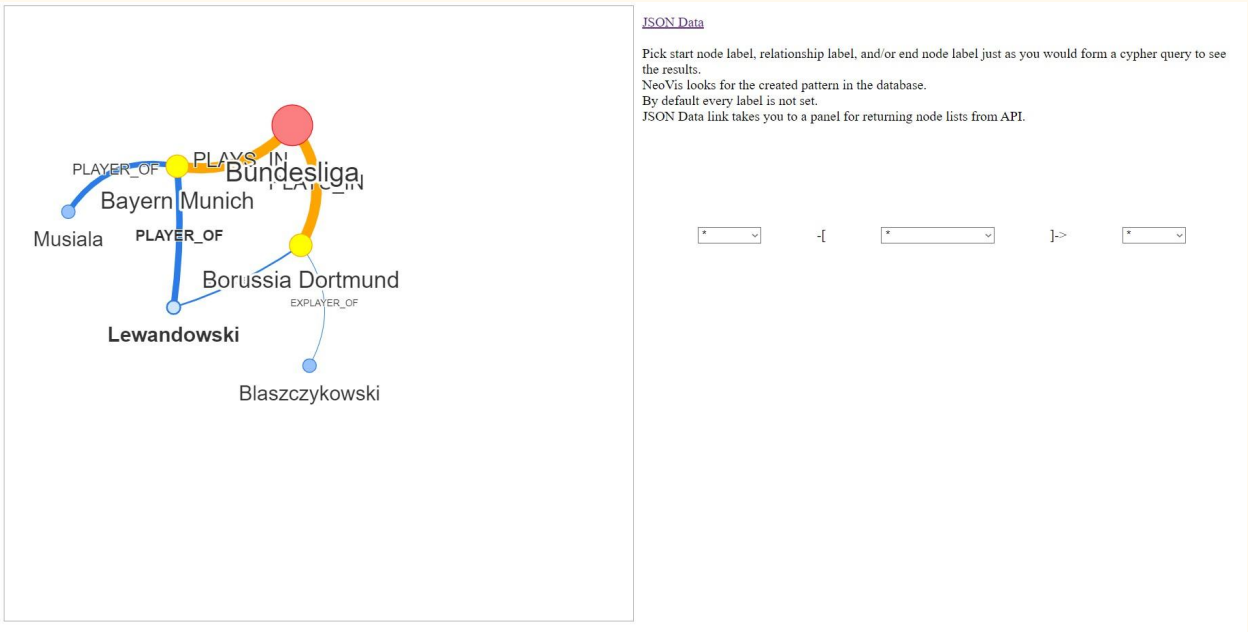

Rysunek 9. Przykładowy JSON wysłany POSTem na /connect/

Korzystanie z API jest dokładnie opisane na końcu dokumentacji.

Front-end

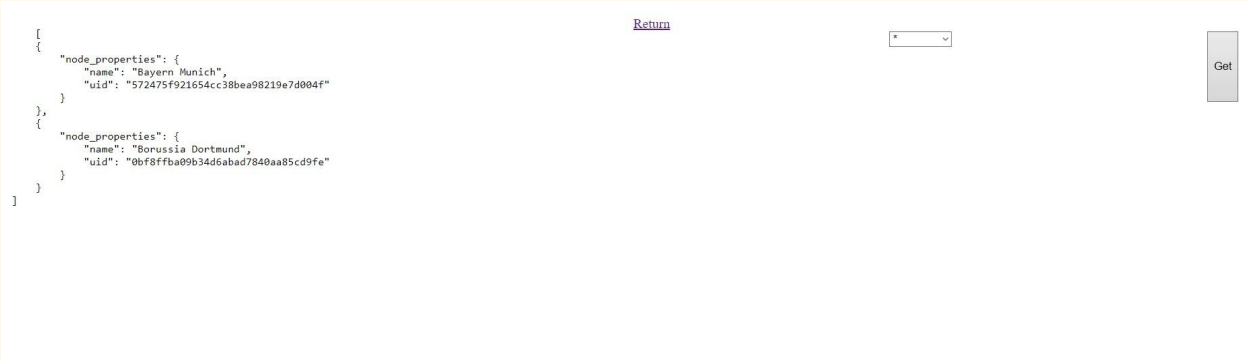
Aplikacja znajduje się pod adresem: https://djangographdb.appspot.com/

Interfejs aplikacji składa się z dwóch części. Pierwsza zawiera panel wyświetlający wizualizację danych w bazie za pomocą biblioteki NeoVis.js, oraz elementy *select* pozwalające na filtrowanie widocznych danych.



Rysunek 10. Strona główna

Druga zamiast wizualizacji pokazuje dane zwrócone ze stworzonego API w postaci JSONa. Domyślnie pokazywana jest lista osób w bazie. Po wybraniu odpowiedniego typu węzłów i wciśnięciu *Get* ładowana jest nowa lista.



Rysunek 11. Strona do odczytywania list węzłów

Podsumowanie

Stworzona aplikacja spełnia wymogi projektu implementując:

- bazę danych Neo4j
- hostowanie w chmurze GCloud
- API zwracające dane z bazy, pozwalające na modyfikację oraz dodawanie elementów
- Front-end pozwalający na wyświetlenie danych z bazy

Wnioski

Grafowa baza danych (w tym przypadku Neo4j), pozwala na przedstawienie dowolnej treści danych, oraz przeszukiwanie ich według określonych wzorów.

Aplikacja mogłaby zostać rozszerzona o implementację dodawania rzeczy do bazy z frontendu za pomocą np. Django Forms, wyszukiwanie bardziej skomplikowanych wzorów, czy połączeń między węzłami, a także powinna wspierać dużo większe zestawy atrybutów dla poszczególnych węzłów, oraz potencjalnie więcej etykiet jak na przykład etykiety związane z pozycją na boisku.

Z powodu wykorzystania publicznej instancji bazy Neo4j AuraDB, a nie płatnej wersji Neo4j Server, aplikacja stwarza kłopoty w łączeniu się z frontendu (jest to potrzebne bibliotece NeoVis.js), jest to problem z różnicą w certyfikatach SSL co powoduje problemy w komunikacji przez WebSocket. **Objawia się to nie działaniem wizualizacji na wszystkich przeglądarkach.** Działanie zostało sprawdzone w przeglądarkach Opera GX i Google Chrome, a także na niektórych urządzeniach mobilnych.

Wykorzystane źródła

https://neomodel.readthedocs.io/en/latest/

https://neo4j-examples.github.io/paradise-papers-django/

https://cloud.google.com/python/django/appengine

https://console.neo4j.io/

Obsługa API

Przedstawiony zostanie proces czytania, dodawania i modyfikacji danych w bazie za pomocą stworzonego API oraz danych/plików JSON. Pliki dostępne są w folderze *json_examples*.

Do obsługi API można wykorzystać dowolne narzędzie pozwalające na wysyłanie zapytań HTTP, w tej instrukcji posłużę się narzędziem HTTPIe: https://httpie.io . Może też to być narzędzie online np. https://reqbin.com .

1. Dodawanie węzłów

Chcemy dodać do bazy węzeł z etykietą Person, np. byłego zawodnika. Niech naszą osobą będzie legenda Bayernu Monachium bramkarz Oliver Kahn, tworzymy więc plik JSON.

```
{
    "name": "Oliver",
    "surname": "Kahn"
}
```

I wysyłamy metodą POST na odpowiedni endpoint w tym przypadku *https://djangographdb.appspot.com/api/people/* **! Musi się kończyć na slashu !.**

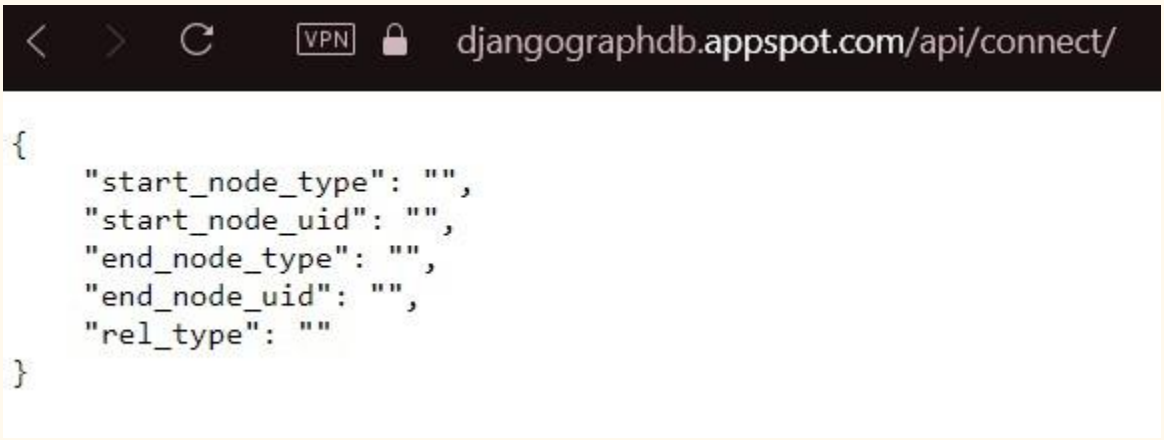
```
(env) E:\DjangoGraphDB\python-docs-samples\appengine\standard_python3\django>http POST https://djangographdb.appspot.com/api/people/ < json_examples
/kahn.json
HTTP/1.1 201 Created
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-Q050=":443"; ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000,quic=":443"
; ma=2592000; v="46,43"
Content-Length: 47
Content-Type: application/json
Date: Mon, 20 Dec 2021 17:38:28 GMT
Referrer-Policy: same-origin
Server: Google Frontend
X-Cloud-Trace-Context: 6da5bba444fea21043ec02c46fb8db4c;o=1
X-Content-Type-Options: nosniff
X-Frame-Options: DENY

{
    "name": "Oliver",
    "surname": "Kahn"
}
```

Możemy się upewnić że węzeł został dodany wysyłając zapytanie GET na ten sam endpoint, albo z panelu na stronie internetowej. Sprawdzam tym drugim sposobem.


```
[
    {
        "node_properties": {
            "created": "2021-12-13T17:53:50.916Z",
            "name": "Jakub",
            "surname": "Blaszczykowski",
            "uid": "f15894cf1ace44e0b212d50eac6f3208"
        }
    },
    {
        "node_properties": {
            "created": "2021-12-13T13:58:45.353Z",
            "name": "Jamal",
            "surname": "Musiala",
            "uid": "f11d071589d94f3b9482db12ec267af2"
        }
    },
    {
        "node_properties": {
            "created": "2021-12-20T17:38:28.095Z",
            "name": "Oliver",
            "surname": "Kahn",
            "uid": "9dc9f2abf6b94bc8b9c233d2096075fa"
        }
    },
    {
        "node_properties": {
            "created": "2021-12-20T17:40:35.314",
            "name": "Robert",
            "surname": "Lewandowski",
            "uid": "0adac929072946eea2a64ddd7ed41015"
        }
    }
]
```


Sprawdźmy czego potrzebujemy żeby połączyć węzły, wysyłając zapytanie GET na endpoint */connect/* np. z przeglądarki.



Potrzebujemy więc uid obu węzłów, ich typu oraz typu relacji jaki je łączy. W przeszłości był to zawodnik Bayernu Monachium, ten klub istnieje już w bazie, więc go wyszukajmy.

Kopiujemy więc uid Bayernu Monachium jako end_node_uid, oraz wpisujemy “Club” jako end_node_type. Potrzebujemy uid Olivera Kahna, już wcześniej je skopiowałem (9dc9f2abf6b94bc8b9c233d2096075fa), wyślijmy dokładne zapytanie o jego dane na endpoint /api/people/9dc9f2abf6b94bc8b9c233d2096075fa.

```
(env) E:\DjangoGraphDB\python-docs-samples\appengine\standard_python3\django>http https://djangographdb.appspot.com/api/people/9dc9f2abf6b94bc8b9c233d2096075fa
HTTP/1.1 200 OK
Alt-Svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-Q050=":443"; ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000,quic=":443"; ma=2592000; v="46,43"
Cache-Control: private
Content-Encoding: gzip
Content-Type: application/json
Date: Mon, 20 Dec 2021 18:08:27 GMT
Referrer-Policy: same-origin
Server: Google Frontend
Transfer-Encoding: chunked
Vary: Accept-Encoding
X-Cloud-Trace-Context: 58cc69ca2814959dd723f81a45df28b4;o=1
X-Content-Type-Options: nosniff
X-Frame-Options: DENY

{
    "node_connections": [
        {
            "direction": "TO",
            "nodes_related": [],
            "nodes_type": "Club",
            "relationship_type": "EXPLAYER_OF",
            "relationship_variable": "career_clubs"
        },
        {
            "direction": "TO",
            "nodes_related": [],
            "nodes_type": "Trophy",
            "relationship_type": "WON",
        }
    ],
    "node_properties": {
        "created": "2021-12-20T17:38:28.095Z",
        "name": "Oliver",
        "surname": "Kahn",
        "uid": "9dc9f2abf6b94bc8b9c233d2096075fa"
    }
}
```

Widzimy że chcemy dodać relację typu EXPLAYER_OF prowadzącą w kierunku węzła Klub do zmiennej *career_clubs*. A więc mamy:

```
{
   "start_node_type": "Person",
   "start_node_uid": "9dc9f2abf6b94bc8b9c233d2096075fa",
   "end_node_type": "Club",
   "end_node_uid": "572475f921654cc38bea98219e7d004f",
   "rel_type":  "career_clubs"
}
```

Jest to zawartość pliku connect_kahn.json który wysyłam POSTem na endpoint https://djangographdb.appspot.com/api/connect/ . Warto pamiętać że klucz “rel_type” oznacza nazwę zmiennej w modelu węzła początkowego zgodnie z logiką kierunku relacji.

W przypadku powodzenia zwróci nam odpowiedź succes: true. Sprawdźmy więc czy węzeł się dodał na stronie głównej.

W folderze json_examples/ są dostępne wzory JSONów odpowiednich dla różnych endpointów.