Projekt zaliczeniowy – Nowoczesne metody przetwarzania danych

Paweł Wąsowski
nr albumu: 23131

Artur Spychalla
nr albumu: 23546

Sprawozdanie:

1. **Jednoznaczny i precyzyjny temat i opis projektu**
   Projekt opiera się na prostym portalu społecznościowym, który udostępnia poniższe możliwości:
   - panel rejestracji i logowania użytkownika,
   - dodawanie postów na tablicę,
   - lajkowanie postów na tablicy,
   - obserwowanie osób oraz sprawdzenie osób, które Cię obserwują,
   - czytanie postów osób które obserwujesz,
   - wyszukanie wszystkich użytkowników i zaobserwowanie ich (możliwość wyszukania po nazwisku),
   - wyszukanie najszybszej drogi do poznania kogoś(za pomocą sieci kontaktów)
   -edycja swojego maila.

   Projekt opiera się na bazie Neo4J oraz języku programowania: JavaScript (node.js z biblioteką express).

2. **Wskazany model danych**
   Baza NoSQL grafowa – Neo4J.

3. **Przekonywujące uzasadnienie wyboru modelu danych oparte na własnościach i funkcjonalnościach projektu i modelu**
   Tworząc social media należy pamiętać o głównych aspektach takich aplikacji. Wszystkie aplikacje Social Media dynamicznie zmieniają swoją wielkość, przepływ informacji stoi cały czas na wysokim poziomie. Nasz wybór padł na Neo4J z powodu, iż posiada poniższe funkcjonalności:

   Performance - wydajność jest cały czas na wysokim poziomie, nawet mimo zwiększenia ilości danych.

   Flexibility - elastyczność bazy danych powoduje, że jest ona łatwa do aktualizacji oraz łatwo ją dostosować do wprowadzanych zmian w aplikacji. Skalowalność bazy danych.

   Agility - struktura baz danych jest łatwa do aktualizacji nowych funkcjonalności, przez co może wraz z rozwojem aplikacji, rozwijać swoją strukturę.

   Z racji tego, że social media posiadają rozbudowane relacje między użytkownikami, mogą być przedstawione na grafowych bazach danych. W związku z tym dobrym wyborem jest neo4j, ponieważ posiada wszystkie cechy, które odgrywają ważną rolę w tworzeniu aplikacji social media.

4. **Wskazane specyficzne (2-3) własności bazy danych – opis i uzasadnienie ich użycia**
   Algorithm shortest paths - znajduje najkrótsze ścieżki, które powinniśmy obrać aby zaznajomić się z innym użytkownikiem. Przykładowo: chcemy poznać prezydenta Poznania.

Aby znaleźć najkrótszą drogę musimy przeanalizować różne możliwości dróg. Kolega naszego ojca ma znajomego, który zna prezydenta Poznania i to jest właśnie najkrótsza droga, która została wyliczona.

Approximate Nearest Neighbors - znajduje osoby, które możemy znać. Zastanawiamy się jeszcze nad tym czy weryfikacja będzie odbywać się na podstawie lajków czy znajomych naszych znajomych.

5. **Opis implementacji – „schemat" bazy danych**
   Baza danych grafowa Neo4J nie ma sztywno zdefiniowanego schematu, właściwości mogą być nadawane dynamicznie.

   Ograniczenie dla bazy jest narzucane jedynie przez strukturę stworzonego przez nas interfejsu, więc możemy mówić o pewnego rodzaju "schemacie".

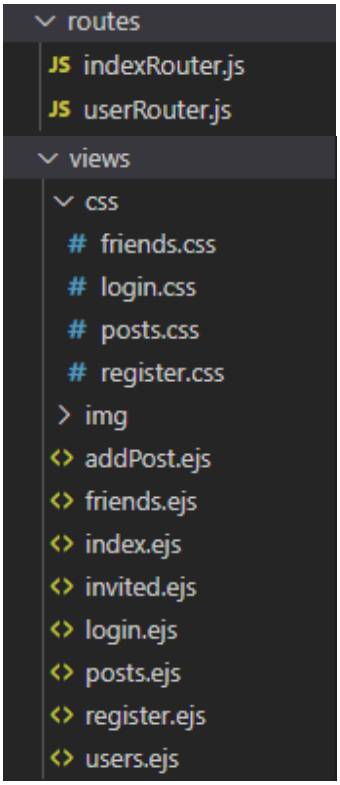
   W Neo4J występują węzły oraz relacje, w naszej implementacji prezentuje się to następująco:
   1. Mamy 2 rodzaje węzłów:

      a) Person(Osoba/Użytkownik).
      b) Post.

   2. Mamy 3 rodzaje relacji

      a) FRIEND_WITH (jest znajomym), relacja ta może występować między dwoma węzłami Person w obie strony.
      b) LIKE (polubił), relacja ta może występować od węzła Person do węzła Post.
      (Person) —-[:LIKE]——>(Post)
      c) POSTED_BY (opublikowany przez), relacja ta może występować od węzła Post do węzła Person.
      (Person)< —-[:POSTED_BY]——(Post)
6. **Opis implementacji – opis zaimplementowanych metod z zaznaczeniem wykorzystania specyficznych własności**.

   Schemat aplikacji:



Połączenie do bazy danych Neo4J:

```
const express = require('express');
const router = express.Router();
const neo4j = require('neo4j-driver');
const { ensureAuthenticated } = require('../config/auth')

const driver = neo4j.driver('bolt://localhost', neo4j.auth.basic('neo4j', '123456'));
const session = driver.session();

```

Metoda, która pobiera z bazy danych użytkowników, których nie mamy w znajomych. Następnie przekazuje informacje takie jak: imię, nazwisko i adres e-mail, który wypisujemy na ekranie aplikacji:

```
// Get users list
router.get('/users', ensureAuthenticated , (req, res) => {
    console.log(req.body.surname);
    session.run('MATCH(n: Person),(p:Person{email:$email}) WHERE NOT (n)<-[:FRIEND_WITH]-(p) RETURN n;',
    {surname: req.body.surname ? req.body.surname : "", email: req.user.records[0]._fields[0].properties.email}).then(result => {
        const usersArr = [];
        result.records.forEach(record => {
            usersArr.push({id: record._fields[0].identity.low, name: record._fields[0].properties.name, surname: record._fields[0].properties.surname, email: record._fields[0].properties.email});
        });
        console.log(usersArr);
        res.render('users', {friends: usersArr, user: req.user});
    }).catch(err => {
        console.log(err);
    });
});
```

Metoda, która wyszukuje w bazie danych użytkownika po nazwisku, które zdefiniujemy w polu input. Następnie przekazuje informacje takie jak: imię, nazwisko i adres e-mail, który wypisujemy na ekranie aplikacji:

```
// Get users list - with search by surname
router.post('/users', ensureAuthenticated , (req, res) => {
    session.run('MATCH(n: Person),(p:Person{email:$email}) WHERE n.surname CONTAINS $surname AND NOT (n)<-[:FRIEND_WITH]-(p) RETURN n;',
    {surname: req.body.surname ? req.body.surname : "", email: req.user.records[0]._fields[0].properties.email}).then(result => {
        const usersArr = [];
        result.records.forEach(record => {
            usersArr.push({id: record._fields[0].identity.low, name: record._fields[0].properties.name, surname: record._fields[0].properties.surname, email: record._fields[0].properties.email});
        });
        res.render('users', {friends: usersArr, user: req.user});
    }).catch(err => {
        console.log(err);
    });
});
```

Metoda, która wyszukuje w bazie danych wszystkich użytkowników, których obserwujemy. Następnie przekazuje informacje takie jak: imię, nazwisko i adres e-mail, który wypisujemy na ekranie aplikacji:

```
//Get people that you've invited
router.get('/invited/:surname?', ensureAuthenticated, (req, res) => {
    session.run('MATCH(Person{email:$emailParam})-[:FRIEND_WITH]->(m:Person) WHERE m.surname CONTAINS $surnameParam AND NOT(m.email CONTAINS $email) RETURN m;',
    {emailParam: req.user.records[0]._fields[0].properties.email, surnameParam: req.params.surname ? req.params.surname : "", email: req.user.records[0]._fields[0].properties.email}).then(result => {
        const friendsArr = [];
        result.records.forEach(record => {
            friendsArr.push({id: record._fields[0].identity.low, name: record._fields[0].properties.name, surname: record._fields[0].properties.surname});
        });
        res.render('invited', {friends: friendsArr, user: req.user});
    }).catch(err => {
        console.log(err);
    });
});
```

Metoda, która wyszukuje w bazie danych obserwowanego użytkownika po nazwisku, które zdefiniujemy w polu input. Następnie przekazuje informacje takie jak: imię, nazwisko i adres e-mail, który wypisujemy na ekranie aplikacji:

```
//Get people that you've invited - with search by surname
router.post('/invited', ensureAuthenticated, (req, res) => {
    session.run('MATCH(Person{email:$emailParam})-[:FRIEND_WITH]->(m:Person) WHERE m.surname CONTAINS $surnameParam AND NOT(m.email CONTAINS $email) RETURN m;',
    {emailParam: req.user.records[0]._fields[0].properties.email, surnameParam: req.body.surname ? req.body.surname : "", email: req.user.records[0]._fields[0].properties.email}).then(result => {
        const friendsArr = [];
        result.records.forEach(record => {
            friendsArr.push({id: record._fields[0].identity.low, name: record._fields[0].properties.name, surname: record._fields[0].properties.surname});
        });
        res.render('invited', {friends: friendsArr, user: req.user});
    }).catch(err => {
        console.log(err);
    });
});
```

Metoda, która wyszukuje w bazie danych użytkowników, którzy nas obserwują:

```
//Get all friends - with search by surname
router.post('/getFriends', ensureAuthenticated, (req, res) => {
    session.run('MATCH(Person{email:$emailParam})-[:FRIEND_WITH]->(m:Person)-[:FRIEND_WITH]->(Person{email:$emailParam}) WHERE m.surname CONTAINS $surnameParam AND NOT(m.email CONTAINS $email) RETURN m;',
    {emailParam: req.user.records[0]._fields[0].properties.email, surnameParam: req.body.surname ? req.body.surname : "", email: req.user.records[0]._fields[0].properties.email}).then(result => {
        const friendsArr = [];
        result.records.forEach(record => {
            friendsArr.push({id: record._fields[0].identity.low, name: record._fields[0].properties.name, surname: record._fields[0].properties.surname});
        });
        res.render('friends', {friends: friendsArr, user: req.user});
    }).catch(err => {
        console.log(err);
    });
});
```

Metoda, która wyszukuje w bazie danych użytkowników, których obserwujemy i przez których jesteśmy obserwowani:

```
//Get all friends
router.get('/friends/:surname?', ensureAuthenticated, (req, res) => {
    session.run('MATCH(Person{email:$emailParam})-[:FRIEND_WITH]->(m:Person)-[:FRIEND_WITH]->(Person{email:$emailParam}) WHERE m.surname CONTAINS $surnameParam AND NOT(m.email CONTAINS $email) RETURN m;',
    {emailParam: req.user.records[0]._fields[0].properties.email, surnameParam: req.params.surname ? req.params.surname : "", email: req.user.records[0]._fields[0].properties.email}).then(result => {
        const friendsArr = [];
        result.records.forEach(record => {
            friendsArr.push({id: record._fields[0].identity.low, name: record._fields[0].properties.name, surname: record._fields[0].properties.surname});
        });
        res.render('friends', {friends: friendsArr, user: req.user});
    }).catch(err => {
        console.log(err);
    });
});
```

Metoda, która wyszukuje w bazie danych użytkowników (po nazwisku), których obserwujemy i przez których jesteśmy obserwowani. Następnie przekazuje informacje takie jak: imię, nazwisko i adres e-mail, który wypisujemy na ekranie aplikacji:

```
// Get posts of your friends
router.get('/posts', ensureAuthenticated, (req, res) => {
    session.run('MATCH(n: Post)-[:POSTED_BY]->(p:Person)<-[:FRIEND_WITH]-(m:Person{email: $emailValue}) OPTIONAL MATCH ()-[r:LIKE]->(n)-[:POSTED_BY]->(p)<-[:FRIEND_WITH]-(m) RETURN n, p, COUNT(r) ORDER BY n.date DESC;
    {emailValue: req.user.records[0]._fields[0].properties.email}).then(result => {
        const postsArr = [];
        result.records.forEach(record => {
            postsArr.push({id: record._fields[0].identity.low, value: record._fields[0].properties.value, date: record._fields[0].properties.date, surname: record._fields[1].properties.surname,
                name: record._fields[1].properties.name, likes: record._fields[2].low});
        });
        res.render('posts', {post: postsArr, user: req.user});
    }).catch(err => {
        console.log(err);
    });
});
```

Metoda, dzięki której możemy za pomocą przycisku na stronie, dodać do obserwowanych danego użytkownika. Zaobserwowanie użytkownika skutkuje utworzeniem relacji „FRIENDS_WITH”:

```
// Add to friends
router.post('/friends', ensureAuthenticated, (req, res) => {
    session.run('MATCH(n:Person{email:$emailParam}),(m:Person{email:$emailFriend}) CREATE (n)-[:FRIEND_WITH]->(m)',
    {emailParam: req.user.records[0]._fields[0].properties.email, emailFriend: req.body.emailFriend}).then(result => {
        res.redirect('/user/invited');
    }).catch(err => {
        console.log(err);
    });
});
```

Metoda, dzięki której pobieramy wszystkie posty użytkowników, których obserwujemy. Jeśli nie obserwujemy użytkownika, jego post nie pojawi się nam na tablicy:

```
//Get people that've invited you - with search by surname
router.get('/invitedBy/:surname?', ensureAuthenticated, (req, res) => {
    session.run('MATCH(Person{email:$emailParam})<-[:FRIEND_WITH]-(m:Person) WHERE m.surname CONTAINS $surnameParam AND NOT(m.email CONTAINS $email) RETURN m;',
    {emailParam: req.user.records[0]._fields[0].properties.email, surnameParam: req.params.surname ? req.params.surname : "", email: req.user.records[0]._fields[0].properties.email}).then(result => {
        const friendsArr = [];
        result.records.forEach(record => {
            friendsArr.push({id: record._fields[0].identity.low, name: record._fields[0].properties.name, surname: record._fields[0].properties.surname});
        });
        res.render('friends', {friends: friendsArr, user: req.user});
    }).catch(err => {
        console.log(err);
    });
});
```

Metoda, dzięki której pobieramy formularz do dodawania postów:

```
// Get form to add posts
router.get('/post', ensureAuthenticated, (req, res) => {
    res.render('addPost', { user: req.user });
});
```

Metoda, dzięki której dodajemy nowy post na tablicy. Dodatkowo pobieramy takie informacje jak data napisania posta (chwila obecna) oraz imię i nazwisko autora. Samo dodanie posta tworzy relację „POSTED_BY" między zalogowanym użytkownikiem a danym postem.

```
// Add post handle
router.post('/post', ensureAuthenticated, (req, res) => {
    const value = req.body.value;
    const emailValue = req.user.records[0]._fields[0].properties.email;
    const today = new Date();
    let day = today.getDate(),
        month = today.getMonth(),
        year = today.getFullYear(),
        time = today.getHours() + ":" + today.getMinutes();

    if (day < 10) { day = '0' + day; }
    if (month < 10) { month = '0' + month; }

    session.run('Match(p:Person{email: $emailValue}) CREATE(p)<-[:POSTED_BY]-(b:Post{value:$valueParam, date:$dateValue}) RETURN p,b ', {
        emailValue: emailValue,
        valueParam: value,
        dateValue: day + '-' + month + '-' + year + " " + time
    })
    .then(() => {
        res.redirect('posts');
    })

    .catch(err => {
        console.log(err);
    });
});
```

Metoda, dzięki której możemy polajkować oraz odlajkować posty obserwującym. Jeśli post nie jest polajkowany przez nas, możemy to wykonać za pomocą przycisku „Like it!" – strona odświeży się automatycznie i naniesie informację o ilości lajków. Jeśli post jest już polajkowany, ponowne naciśnięcie przycisku „Like it!" sprawi, że odlajkujemy dany post – strona odświeży się automatycznie i naniesie informację o ilości lajków.

```
// Like post
router.post('/like', ensureAuthenticated, (req, res) => {
    session.run('MATCH(p:Post{value: $postValue})<-[r:LIKE]-(m:Person{email: $emailValue}) RETURN COUNT(r)',
    {emailValue: req.user.records[0]._fields[0].properties.email, postValue: req.body.value}).then(result =>{
        if(result.records[0]._fields[0].low == 0){
            session.run('MATCH(p:Post{value: $postValue}), (m:Person{email: $emailValue}) CREATE (p)<-[:LIKE]-(m) RETURN p,m',
            {emailValue: req.user.records[0]._fields[0].properties.email, postValue: req.body.value}).then(result => {
                res.redirect('posts');
            }).catch(err => {
                console.log(err);
            });
        }else{
            session.run('MATCH(p:Post{value: $postValue})<-[r:LIKE]-(m:Person{email: $emailValue}) DELETE r',
            {emailValue: req.user.records[0]._fields[0].properties.email, postValue: req.body.value}).then(result => {
                res.redirect('posts');
            }).catch(err => {
                console.log(err);
            });
        }
        res.redirect('posts');
    }).catch(err => {
        console.log(err);
    });
});
```

Metoda, która renderuje nam nową stronę, w której możemy zmienić maila:

```
//Render email site
router.get('/email', ensureAuthenticated, (req, res) => {
    res.render('email', {user: req.user});
})
```

Metoda, dzięki której możemy zmienić adres mailowy. Dodatkowo zaimplementowana jest metoda wykrywająca czy dany mail nie jest już zajęty. Po zakończeniu operacji z rezultatem: sukces – przenosi nas do strony logowania. W przeciwnym wypadku nie zmienia maila i pozostaje na podstronie, w której jesteśmy:

```
// Change email
router.post('/email', ensureAuthenticated, (req, res) => {
    session.run('MATCH (p:Person{email:$emailValue}) RETURN COUNT(p)', {emailValue: req.body.newEmail}).then(result => {
        console.log(result.records[0]._fields[0].low);
        if (result.records[0]._fields[0].low == 0) {
            session.run('MATCH(n:Person{email:$emailValue}) SET n.email = $newEmail', {
                emailValue: req.user.records[0]._fields[0].properties.email,
                newEmail: req.body.newEmail
            }).then(result => {
                res.redirect('logout');
            }).catch(err => {
                console.log(err);
            });
        }else{ res.redirect('/user/email'); }
    });
});
```

Metoda, dzięki której możemy wylogować się z konta użytkownika. Po wylogowaniu przekierowuje nas do ekranu logowania:

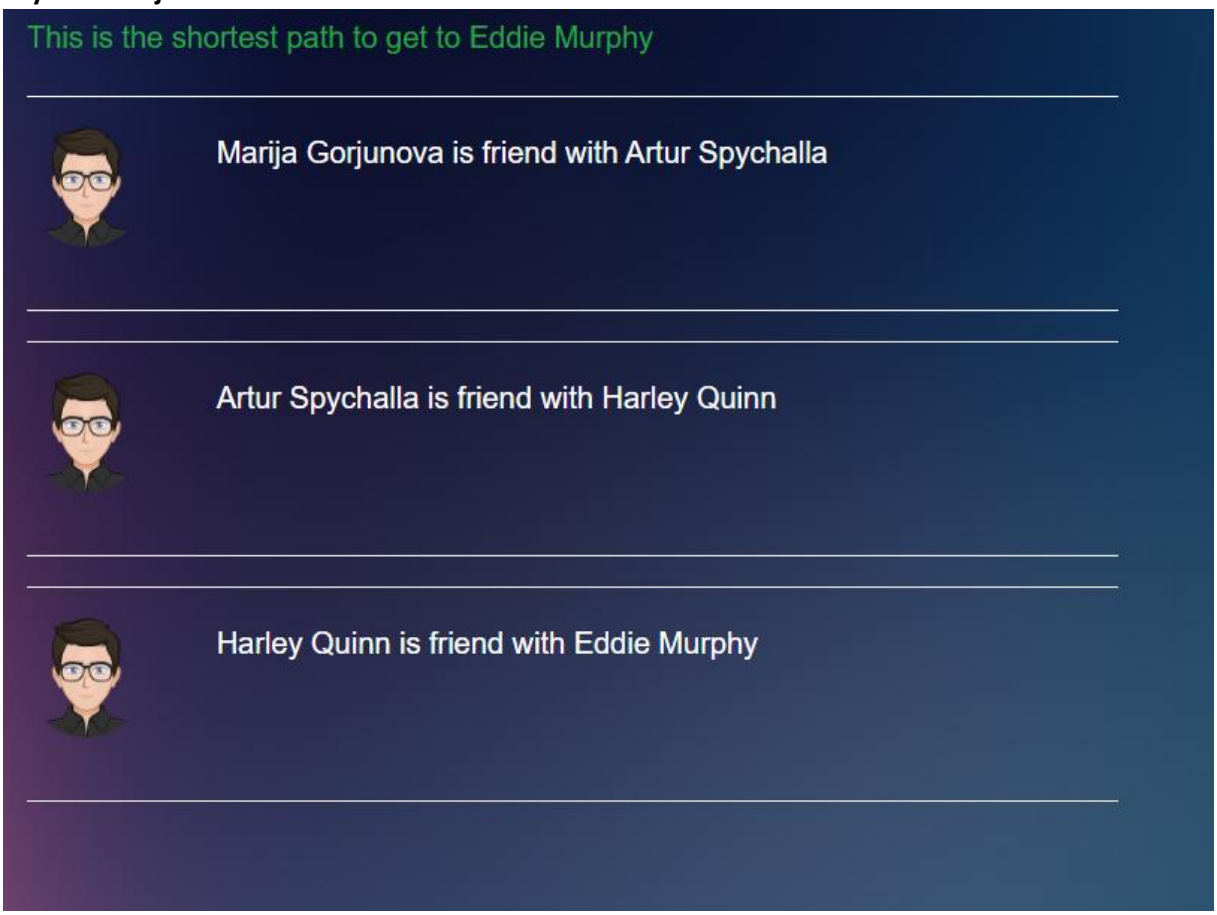
```
// Logout handle
router.get('/logout', (req, res) => {
    req.logout();
    res.redirect('/login');
});

module.exports = router;
```

Metoda „shortestPath”, dzięki której możemy znaleźć drogę do osoby, przez zdefiniowany adres e-mail w inpucie. Algorytm wyszukuje najszybszą i najprostszą drogę do danej osoby, wyświetlając po kolei każdego znajomego, kończąc na szukanej osobie. Na końcu przekazuje całą drogę do aplikacji, która wyświetla wszystkich użytkowników.

```
// Find connections
router.post('/findConnection', ensureAuthenticated, (req, res) => {
    const conArr = [];
    let target = '';
    session.run('MATCH (a:Person{email:$emailParam}),(b:Person{email:$emailFriend}), p=shortestPath((a)-[*]-(b)) RETURN p',
    {emailParam: req.user.records[0]._fields[0].properties.email, emailFriend: req.body.email}).then(result => {
        //console.log(JSON.stringify(result.records[0]._fields[0].segments));
        target = result.records[0]._fields[0].end.properties.name+' '+result.records[0]._fields[0].end.properties.surname;
        result.records[0]._fields[0].segments.forEach(record => {
            conArr.push({startName: record.start.properties.name + " " + record.start.properties.surname,
            relation: record.relationship.type=="FRIEND_WITH" ? "is friend with" : record.relationship.type, endName: record.end.properties.name + " " + record.end.properties.surname});
        })
        res.render('contact', {contact: conArr, user: req.user, target});
    }).catch(err => {
        console.log(err);
    });
});
```

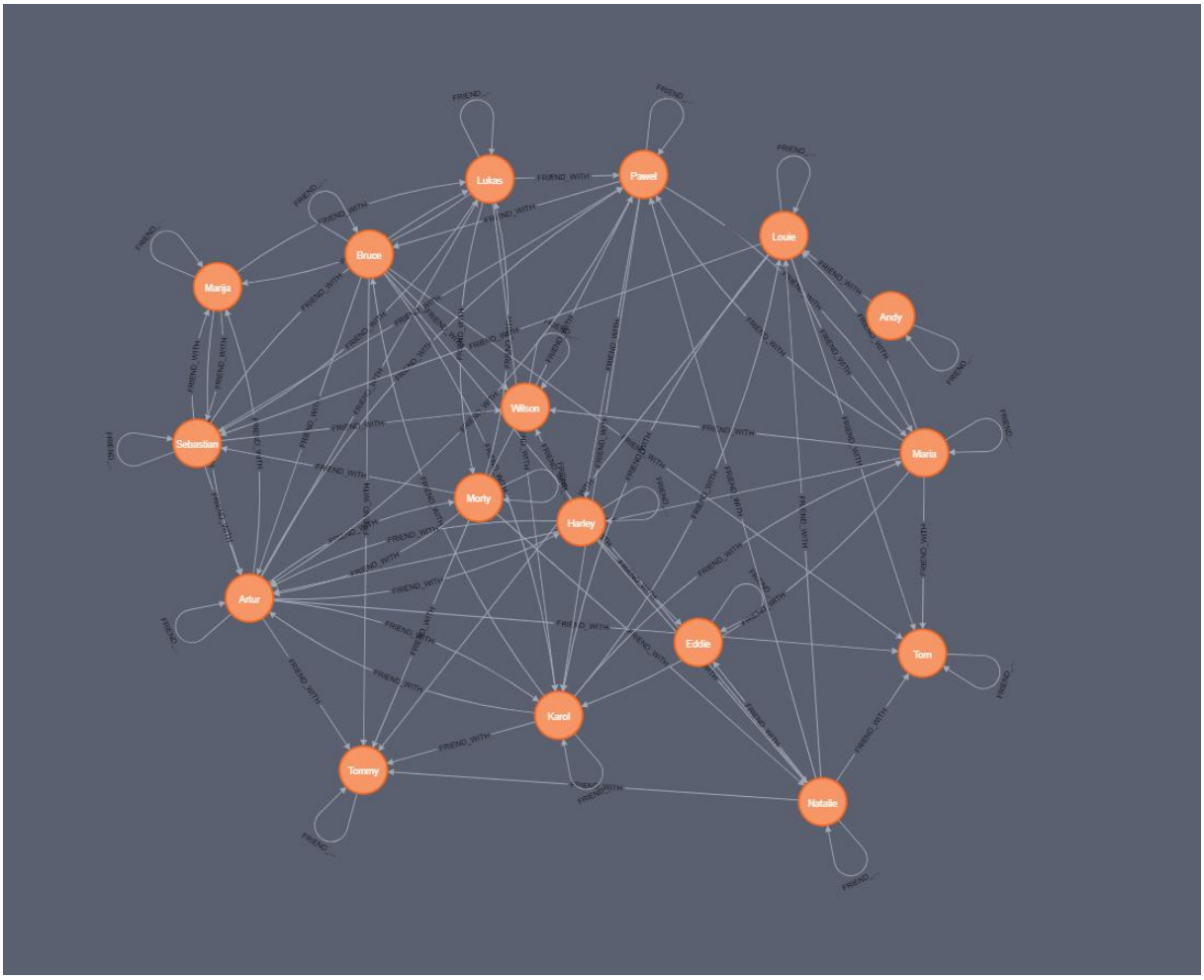
Zrzuty ekranu poniżej.

Wynik funkcji shortestPath:

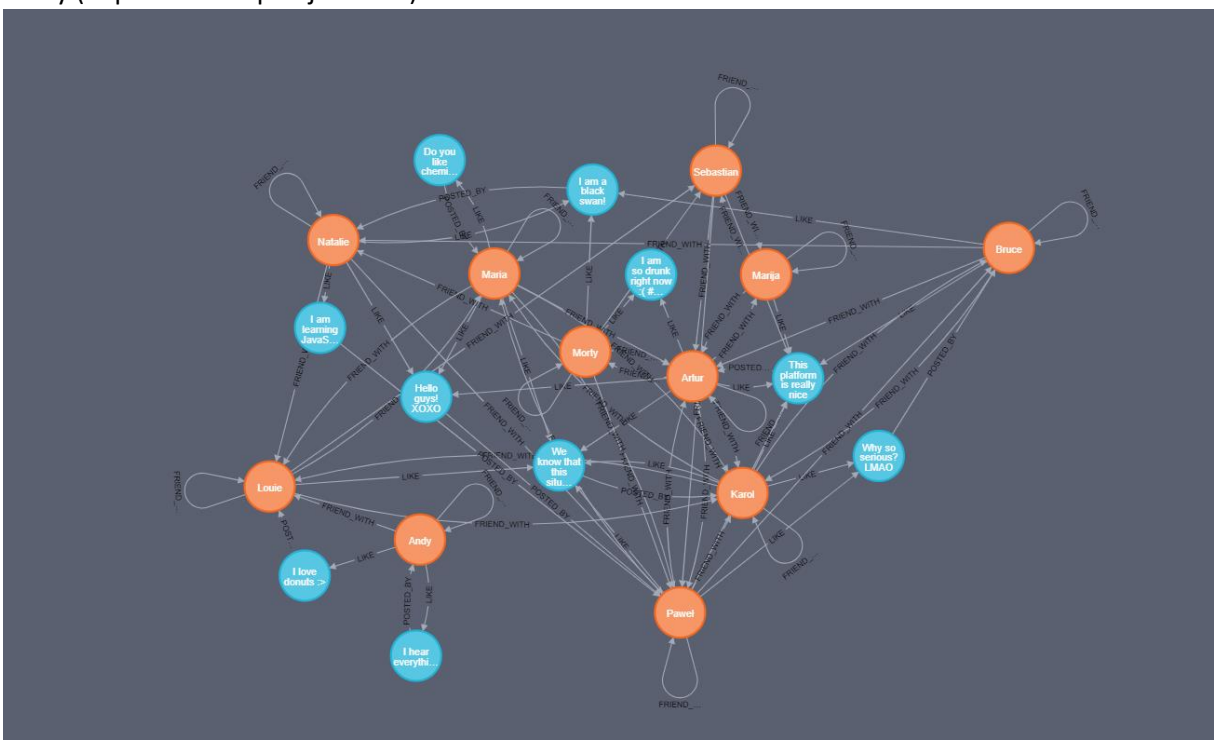


Baza danych Neo4J:

Użytkownicy:



Posty (napisane oraz polajkowane):



Cała baza danych: