



Uniwersytet Rzeszowski

Projekt inżynierski II

Aplikacja wspierająca proces wyszukiwania informacji

Opiekun projektu: dr inż. Piotr Grochowalski

Autorzy Projektu: Rafał Kieroński , Jakub Kuśnierz

Data wykonania: Semestr zimowy rok akademicki 2017/2018

1.Cel projektu

Celem projektu było wykonanie aplikacji wspierającej wyszukiwanie informacji. Informacje miały być wyszukiwane trzema metodami przy czym dwie metody miały zostać zaimplementowane przy wykorzystaniu gotowych algorytmów, a jedna metoda miała zostać zaproponowana i zaimplementowana przez autorów projektu

2.Opis aplikacji

W ramach projektu powstała aplikacja umożliwiająca dodawanie oraz wyszukiwanie ofert pracy. Aplikacja została wykonana w technologii webowej wykorzystując w głównej mierze technologie Java. Ogłoszenia są magazynowane w bazie danych MySQL. Aplikacja stworzona została w oparciu o wzorzec projektowy MVC (model-widok-kontroler). Do pracy grupowej wykorzystany został system kontroli wersji git. Główny nacisk położony został na technologie wyszukiwania.

Wyszukiwanie zostało wykorzystane przy wyszukiwaniu ogłoszeń po słowie kluczowym, zaawansowanym wyszukiwaniu ogłoszeń oraz do sprawdzania czy użytkownik przy dodawaniu nowego ogłoszenia nie robi „spamu”.

Inne funkcje zostały zaimplementowane w celu możliwości obsługi całego serwisu przez stronę internetową bez konieczności wykonywania operacji bezpośrednio w bazie danych. Są to m.in. rejestracja użytkowników, logowanie, dodawanie ogłoszeń, wyświetlanie listy ogłoszeń, podgląd własnych ogłoszeń wraz z możliwością ich usuwania oraz edycji.

3. Wyszukiwanie po słowie kluczowym

Wyszukiwanie po słowie kluczowym jest najkrótszym algorytmem w projekcie. Odbywa się ono za pomocą algorytmu wyszukiwania pełnotekstowego (ang. Full-text search) zaimplementowanego w bazie danych. Największą zaletą tego wyszukiwania jest to, że zwrócone ogłoszenia mają wartość oznaczającą trafność wiersza do zapytania (na początku wyświetlane są najlepsze trafienia dla szukanej frazy).

```
1 CREATE DEFINER='root'@'localhost' PROCEDURE `fullTextSearch`(in zapytanie varchar(200))
2 BEGIN
3 SELECT * FROM ogloszenie WHERE MATCH (tytul,lokalizacja,opis) AGAINST(zapytanie);
4 END
```

Rysunek 1 Kod wyszukiwania pełnotekstowego

Wyszukiwanie w pierwszej fazie implementowane było z klauzulą „WITH QUERY EXPANSION” która wyszukuje również ogłoszenia po słowach najczęściej występujących w ogłoszeniach pasujących do wpisanej przez nas frazy. Niestety ta opcja nie znalazła zastosowania w naszej aplikacji z powodu zbyt dużej rozbieżności zwracanych rezultatów od oczekiwanych przez wyszukiującego.

4. Wyszukiwanie zaawansowane

Wyszukiwanie zaawansowane odbywa się za pomocą algorytmu list prostych oraz jego dwóch modyfikacji podziału połówkowego i tablicy adresowej wykorzystanych w celu przyspieszenia wyszukiwania. Algorytm ten polega na przeglądnięciu wszystkich składowych zapytania(termów) i porównaniu ich z deskryptorem pytania (np. czy poszukiwana odpowiedz jest opisana deskryptorem wysoki? Jeżeli nie to przechodzę to następnego wiersza, jeżeli tak porównuje kolejny deskryptor).

W naszej aplikacji wyszukiwania zaawansowane możliwe jest przez podanie następujących informacji:

- przedziału oczekiwanych zarobków
- kategorii (np. bankowość, budownictwo, IT)
- lokalizacji
- stanowiska (np. kierownik, asystent, pracownik fizyczny)
- formy zatrudnienia (np. pełen etat, pół etatu)

Użytkownik może podać dowolną ilość informacji i zapytanie zostanie dopasowane do otrzymanych wiadomości. Z powodu możliwości podania jednorazowo aż 5 różnych deskryptorów zdecydowaliśmy się zastosować modyfikacje wykorzystywane przy podaniu najczęściej podawanych przez użytkowników informacji czyli zarobków oraz kategorii. W aplikacji pierwsza wykonywany podział połówkowy gdy są podane zarobki następnie tablica adresowa jeżeli jest określona kategoria a następnie wyszukiwania na podstawie reszty deskryptorów.

Modyfikacja podziału połówkowego dzieli listę przeszukiwanych ogłoszeń na połowę i jeżeli środkowy atrybut jest większy od poszukiwanego przeszukuje tylko pierwszą połowę tablicy, w przeciwnym razie przeszukuje tylko drugą połowę. Wadą tej modyfikacji jest możliwość utraty kompletności wyszukiwania. Może się to zdarzyć gdy środkowy element będzie taki sam jak nasz deskryptor. Wtedy w nieprzeszukanej połowie tablicy może wystąpić algorytm spełniający kryteria naszego wyszukiwania. Wada ta została u nas rozwiązana przez zaimplementowanie własnej poprawki która naprawia wyżej opisaną wadę i gwarantuje uzyskanie kompletnego wyszukiwania.

```
List<Advertisement> ads = new ArrayList<>();
if (minSalary == middleSalary || (middleSalary > minSalary && middleSalary < maxSalary)) {
    //sprawdzaj wartosc middleIndex-k do momentu az wartosc bedzie wieksza od maxSalary, gdzie k to ilosc iteracji
    int index = startingIndex(foundAds, middleIndex, middleSalary);
    while (index < foundAds.size() && foundAds.get(index).getZarobki() <= maxSalary) {
        ads.add(foundAds.get(index));
        index++;
    }
    return ads;
} else if (maxSalary == middleSalary) {
    //sprawdzaj wartosc od middleIndex+k, do momentu az wartosc bedzie mniejsza od minimalnej gdzie k to ilosc iteracji
    int index = startingIndexFromTheEnd(foundAds, middleIndex, middleSalary);
    while (index > -1 && foundAds.get(index).getZarobki() >= minSalary) {
        ads.add(foundAds.get(index));
        index--;
    }
    return ads;
}
```

Rysunek 2 Kawatek algorytmu modyfikacji połówkowej

Drugą zastosowaną modyfikacją jest tablica adresowa. Wykorzystana zostaje ona gdy podana zostaje kategoria. Jej zadaniem jest posortowanie ogłoszeń oraz utworzenia tablicy z adresem początkowym i końcowym ogłoszeń dla danej kategorii na liście wszystkich ogłoszeń. Dzięki niej automatycznie zostają odrzucone ogłoszenia z inną kategorią i nie muszą być przeszukiwane.

```

//tablica adresowa ze wzgledu na kategorie wyszukiwania
private List<Advertisement> tableAddress(List<Advertisement> foundAds, int idCategory) {
    List<Advertisement> listFoundsAds = new ArrayList<>();
    ArrayList<Integer> idCategories = new ArrayList<>();
    int[][] tabIndex = new int[31][2];
    Collections.sort(foundAds, new AdvertisementComparator());
    if (foundAds.size() > 0) {
        idCategories.add(foundAds.get(0).getId_kategoria());
        tabIndex[0][0] = 0;
        for (int i = 1; i < foundAds.size(); i++) {
            if (foundAds.get(i).getId_kategoria() != idCategories.get(idCategories.size() - 1)) {
                idCategories.add(foundAds.get(i).getId_kategoria());
                tabIndex[idCategories.size() - 2][1] = i - 1;
                tabIndex[idCategories.size() - 1][0] = i;
            }
        }
        tabIndex[idCategories.size() - 1][1] = foundAds.size() - 1;
    }
    int indexCategory = idCategories.indexOf(idCategory);
    if (indexCategory > -1) {
        for (int i = tabIndex[indexCategory][0]; i <= tabIndex[indexCategory][1]; i++) {
            listFoundsAds.add(foundAds.get(i));
        }
    }

    return listFoundsAds;
}

```

Rysunek 3 Algorytm tworzenia tablicy adresowej

5. Sprawdzanie dodawania spamu

Sprawdzanie spamu odbywa się za pomocą połączenia algorytmów: przeszukiwania z wartownikiem, algorytmu miary TD-IDF oraz użyciu „stoplisty” dla słów polskich która usuwa słowa nie wnoszące nic dla ważności kontekstu (np. na, i ,a , o).

Do sprawdzania spamu przyjęliśmy następujące założenia:

- ogłoszenia porównywane są tylko z ogłoszeniami dodanymi przez danego użytkownika
- ogłoszenia mogą być podobne w różnych lokalizacjach (np. podczas rekrutacji do oddziałów jednej firmy w różnych miastach)

-stosunek sumy słów powtarzających się do ilości słów musi być mniejszy niż 90%

-stosunek ilości słów występujących w drugim ogłoszeniu do sumy słów musi być mniejszy niż 80%

Wartości te mogą być modyfikowane po obserwacji pracy serwisu oraz zachowaniach użytkowników.

Pierwszym etapem wyszukiwania spamu jest zastosowanie algorytmu przeszukiwania z wartownikiem w celu znalezienia ogłoszeń użytkownika który próbuje dodać nowe ogłoszenie. Wyszukiwanie to cechuje się tym że na koniec listy dodawany jest tzw. „wartownik” czyli element który chcemy wyszukać. Lista jest przeszukiwana do momentu wystąpienia szukanego elementu. W momencie znalezienia sprawdza czy jest to nasz „wartownik” jeżeli tak kończy przeszukiwanie w innym wypadku dodaje go do nowej listy i szuka dalej. Algorytm ten jest oszczędniejszy obliczeniowo ponieważ operacja sprawdzenia czy jest to koniec listy odbywa się tylko po natrafieniu na szukany element a nie po każdym elemencie listy.

```
//przeszukiwanie z wartownikiem
private List<Advertisement> linearSearchWithGuard(int userID) {
    List<Advertisement> adverts = new ArrayList<>();
    List<Advertisement> foundAds = new ArrayList<>();
    adverts = advertisementDAO.findAllByID(userID);
    if (adverts.size() == 0) return adverts;
    adverts.add(new Advertisement(userID, id_kategoria: 0, id_forma_zatrudnienia: 0, id_stanowisko: 0, tytul: "guard", lokalizacja: "guard", zarobke: 0, opis: "guard"));
    int i = 0;
    while (true) {
        if (adverts.get(i).getId_uzytkownik() == userID) {
            if (i == adverts.size() - 1) {
                break;
            }
            foundAds.add(adverts.get(i));
        }
        i++;
    }
    return foundAds;
}
```

Rysunek 4 Algorytm przeszukiwania z wartownikiem

Kolejnym etapem jest przygotowanie tekstu do porównywania poprzez usunięcie zbędnych słów nie zmieniających kontekstu. Odbywa się to za pomocą przygotowanej odpowiedniej listy słów oraz usunięciu ich z tekstu.

Ostatnim etapem jest porównanie przygotowanych tekstów algorytmem miary td-idf z dostosowanymi wzorami dla naszego przypadku oraz określenie wyniku na podstawie otrzymanych rezultatów.

```
private String algorithmTF_idf(List<Advertisement> ads, Advertisement advertisement) {
    String[] description = advertisement.getOpis().split( regex: " ");
    description = removeDuplicates(description);
    int[] numberDuplicates = new int[description.length];
    for (Advertisement ad : ads) {
        int amount = 0;
        int amountUseWords = 0;
        fillInZero(numberDuplicates);
        String[] text = ad.getOpis().split( regex: " ");
        fillInDuplicates(description, numberDuplicates, text);

        for (int numberDuplicate : numberDuplicates) {
            amount += numberDuplicate;
            if (numberDuplicate > 0) amountUseWords++;
        }

        float resultSpam = ((float) amount / (float) description.length);
        if (resultSpam > 0.9) {
            float resultSpam2 = ((float) amountUseWords / (float) description.length);
            if (resultSpam2 < 0.8) {
                System.out.println("Wspolczynniki spamu: " + resultSpam + " , " + resultSpam2);
            } else {
                System.out.println("Wspolczynniki spamu: " + resultSpam + " , " + resultSpam2);
                return "spam";
            }
        }
    }
    return "noSpam";
}
```

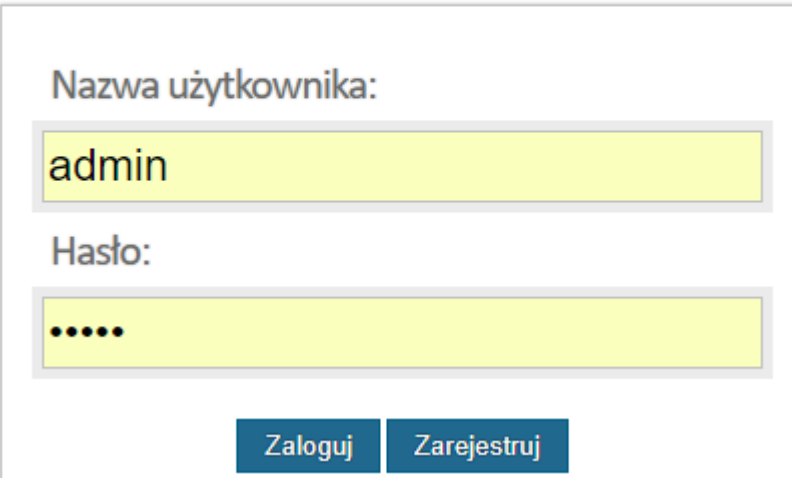
Rysunek 5 Algorytm miary TF-IDF

6. Inne funkcjonalności aplikacji

- **Możliwość logowania oraz rejestracji**

Aplikacja jest wyposażona w moduł logowania oraz rejestracji nowych użytkowników wraz z podstawową walidacją danych wprowadzanych przy próbie zarejestrowania nowego użytkownika.

Logowanie:



The screenshot shows a login form with the following elements:

- Label: Nazwa użytkownika:
- Input field containing: admin
- Label: Hasło:
- Input field containing:
- Buttons: Zaloguj and Zarejestruj

Rysunek 6 Okno logowania

- **Wyświetlania listy ogłoszeń**

Po wyborze z menu listy ogłoszeń zobaczymy stronę z listą wszystkich ogłoszeń. Na górze strony znajdują się okienko do wyszukiwania. Na stronie dodatkowo została zaimplementowana paginacja z limitem 15 ogłoszeń oraz możliwością przechodzenia między podstronami z ogłoszeniami.

Wyszukaj Ogłoszenie: [SZUKAJ](#)

Lista wszystkich ogłoszeń

1 2 3 4 5

Murarz

Autor: admin

Kategoria: Budownictwo

Forma zatrudnienia: Pełny etat

Stanowisko: Pracownik fizyczny

Lokalizacja: Rzeszów

Zarobki: 3400

Zatrudnię murarza. Praca na terenie podkarpacia. Na okres próbny umowa zlecenie, a następnie umowa o pracę.

Tynkarz

Autor: admin

Kategoria: Budownictwo

Forma zatrudnienia: Pełny etat

Stanowisko: Pracownik fizyczny

Lokalizacja: Łódź

Zarobki: 2900

Zatrudnię na stanowisku tynkarza maszynowego. Praca głównie w woj. łódzkim

Pracownik biurowy-likwidacja szkód

Autor: admin

Kategoria: Administracja biurowa

Forma zatrudnienia: Pełny etat

Stanowisko: Specjalista

Lokalizacja: Częstochowa

Zarobki: 3000

Rysunek 7 Fragment strony z ogłoszeniami

- **Własne ogłoszenia**

Strona z własnymi ogłoszeniami widoczna jest tylko dla zalogowanego użytkownika. Wyglądem przypomina listę wszystkich ogłoszeń, ale wyposażona jest jeszcze w przyciski do edytowania i usuwania dodanego wcześniej ogłoszenia

- **Dodawanie ogłoszeń**

Strona została wyposażona w moduł dodawania nowych ogłoszeń wraz z walidacją poprawności danych oraz minimalnych długości informacji wprowadzonych przez użytkownika.

Dodawanie nowego ogłoszenia

Kategoria:

Forma zatrudnienia:

Stanowisko:

Tytuł:

Lokalizacja:

Zarobki:

Opis:

Dodaj

Rysunek 8 Formularz dodawania nowego ogłoszenia

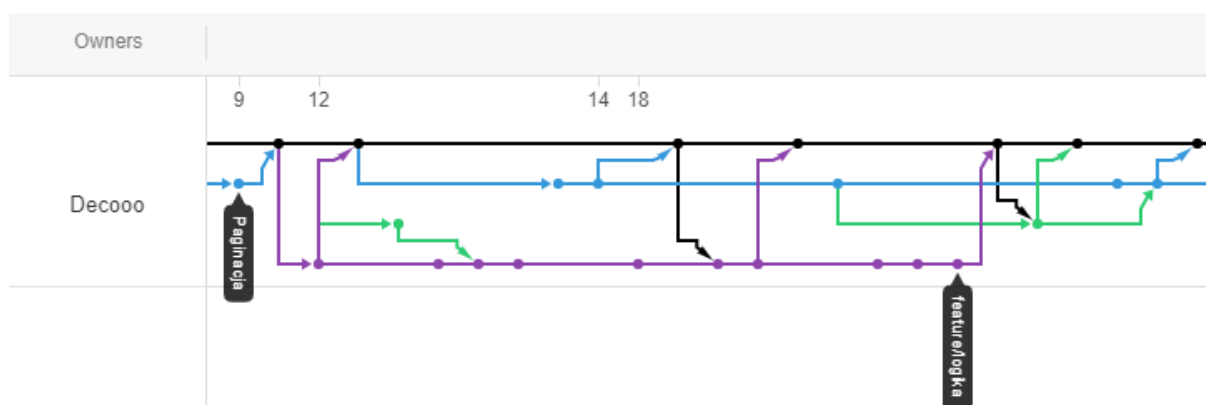
7. Budowa aplikacji i system kontroli wersji

Do tworzenia aplikacji wykorzystany został system kontroli wersji. Znacznie pomógł i przyspieszył tworzenie aplikacji w grupie oraz zapewniał bieżący dostęp do poprzednich wersji co pozwalało łatwo porównywać zmiany oraz przywracać wcześniejsze wersje.

W całym procesie tworzenia wykonane zostało 129 commitów, utworzone zostało 15 różnych branchów oraz „wydane” zostało 6 wersji projektu (releases).



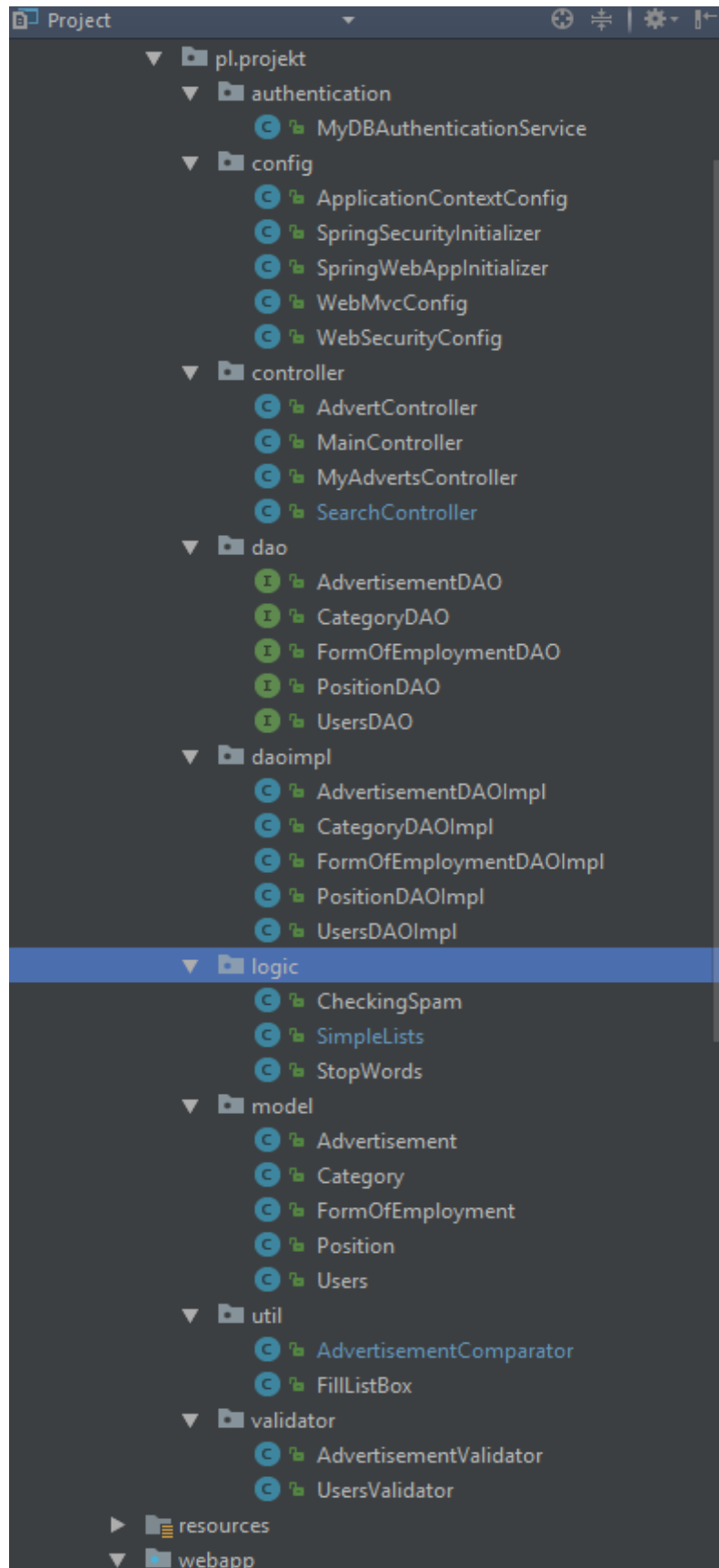
Rysunek 9 Statystyki z gitHuba



Rysunek 10 Fragment wykresu rozgałęziania projektu

Przy tworzeniu aplikacji wykorzystany został wzorzec projektowy **Model-View-Controller** który służy do organizowania struktur aplikacji posiadających graficzny interfejs użytkownika. Użyty został również komponent **Data Access Object** wykorzystywany jest do komunikacji między aplikacją a źródłem danych (np.. bazą danych). DAO stosowany w modelu MVC oddziela dostęp do danych od logiki warstwy prezentacji. Aplikacja w języku java zawiera 27 klas,

5 interfejsów, 13 stron JSP oraz kilka plików konfiguracyjnych. Budowa aplikacji pokazana została na poniższych zdjęciach.

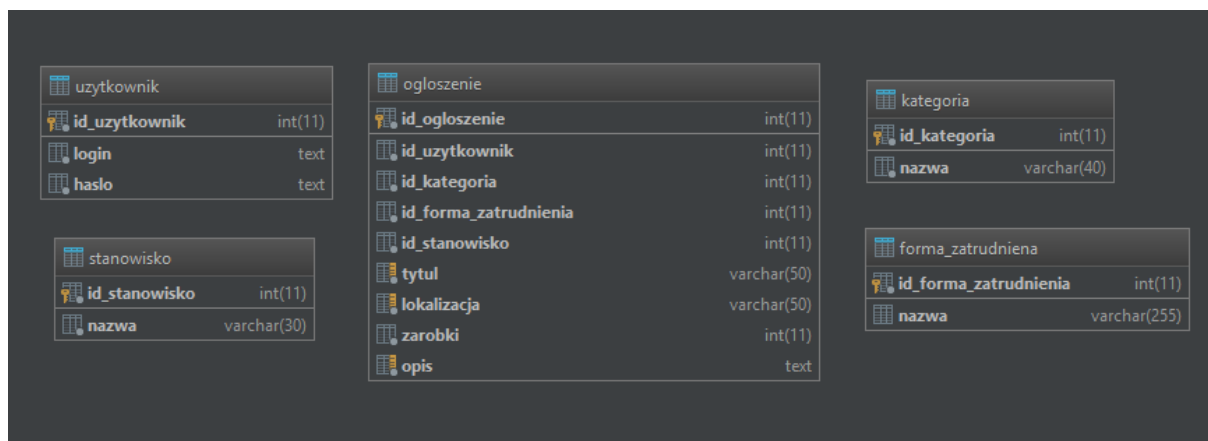


8. Baza danych

Baza danych wykonana została w MySQL. Do celów naszego projektu zostało wykonane 5 tabel. Z powodu łatwej budowy istnieje możliwość bardzo prostej rozbudowy bazy o kolejne encje. W bazie danych zostało napisanych także 5 procedur obsługujących zapytania oraz 1 procedura do wyszukiwania pełnotekstowego.

Tabele w bazie danych:

- użytkownik (dane użytkowników: login i hasło)
- kategoria (nazwa i id kategorii)
- forma zatrudnienia (id i nazwa formy zatrudnienia)
- stanowisko (id i nazwa stanowiska)
- ogłoszenia (autor, kategoria, stanowisko, forma zatrudnienia, tytuł, lokalizacja, zarobki oraz opis)



Rysunek 11 Schemat wizualizujący bazę danych

9. Wykorzystane technologie/programy

- Baza danych została wykonana w MySQL
- Kod źródłowy napisany został w technologii java 1.8
- Do pisania aplikacji wykorzystane zostały następujące technologie:
 - Spring w wersji 4.2.5
 - Spring Security w wersji 4.0.4
 - Hibernate w wersji 5.1.0
- Do zarządzania projektem używany jest Maven w wersji 4.0
- Strona www wykonana została w technologii JSP (JavaServlet Pages)
- Do uruchamiania aplikacji wykorzystany został Apache Tomcat w wersji 2.2
- Projekt napisany został w programie IntelliJ IDEA
- Wykorzystany został system kontroli wersji Git