Západočeská univerzita v Plzni

Vyhledávání informací

KIV/IR

Implementace vlastního systému automatické indexace a vyhledávání dokumentů

Čarnogurský Jan
A19N0025P
cagy@students.zcu.cz

2. června 2020

Obsah

1 Zadání

1.1 Minimální nutná funkčnost semestrání práce pro získání 15 bodů/zápočtu

Tokenizace, Preprocessing (stopwords remover, stemmer/lemmatizer), vytvoření in-memory invertovaného indexu, tf-idf model, cosine similarity, vyhledávání pomocí dotazu vrací top x výsledků seřazených dle relevance, vyhledávání s pomocí logických operátorů AND, OR, NOT, podrobná dokumentace (programátorská i uživatelská), podpora závorek pro vynucení priority operátorů.

Semestrální práce musí umožňovat zaindexování poskytnutých dat a dat stažených na cvičení (1. cvičení Crawler).

1.2 Nadstandardní funkčnost (lze získat až dalších 15 bodů)

Vytvořte účetní aplikaci s následujícími funkcemi:

- File-based index (1b)
- pozdější doindexování dat (1-2b) - přidání nových dat do existujícího indexu
- ošetření např. HTML tagů (1b)
- detekce jazyka dotazu a indexovaných dokumentů (1b)
- vylepšení vyhledávání (1-?b)
- vyhledávání frází (i stop slova)(1b)
- vyhledávání v okolí slova (1b)
- více scoring modelů (1b)
- indexování webového obsahu (2-3b) - zadam web, program stahne data a rovnou je zaindexuje do existujícího indexu
- další předzpracování normalizace (1b)
- GUI/webové rozhraní (2b)
- napovídání keywords (1b)

- podpora více polí pro dokument (např. datum, od do)(1b)
- CRUD indexovaných dokumentů (2b)
- zvýraznění hledaného textu v náhledu výsledků (1b)
- dokumentace psaná v TEXu (1b)
- implementace dalšího modelu (použití sémantických prostorů) (1-?b)
- atd. (xb)

2 Analýza

2.1 Fulltextové vyhledávání

Je speciální způsob vyhledávání informací v databázích nebo v textových souborech, které jsou obvykle předem připraveny, tj. indexovány, aby bylo možno nalézt libovolné slovo (řetězec znaků) v nejkratším možném čase. Při full-textovém vyhledávání vyhledávací algoritmus prozkoumává všechna slova v každém uloženém dokumentu a pokouší se je porovnat se slovy zadanými uživatelem a pokouší se nalézt nejrelevantnější výsledky k zadanému dotazu.

2.2 Invertovaný index

je datová struktura používaná pro fulltextové vyhledávání. Struktura obsahuje mapování obsahu, jako jsou slova nebo čísla. Index se typicky skládá ze slovníku, kde každé slovo obsahuje seznam dokumentů v kterých se vyskytuje. Pro dané slovo lze tedy triviálně zjistit, které dokumenty mu odpovídají.

2.3 TF-IDF

TF-IDF (termín frekvence inverzní frekvence dokumentu) je statistické měřítko, které vyhodnocuje, jak relevantní slovo je pro dokument ve sbírce dokumentů. To se provádí vynásobením dvou metrik: kolikrát se slovo objeví v dokumentu a inverzní frekvence dokumentu slova v sadě dokumentů.

Funguje tak, že se úměrně zvyšuje, kolikrát se slovo objeví v dokumentu, ale je kompenzováno počtem dokumentů, které toto slovo obsahují. Slova, která jsou běžná v každém dokumentu, jako je „tento“, „co“, „a pokud“, mají nízké hodnocení, i když se mohou objevit mnohokrát, protože pro relevanci dokumentu nic neznamenají.

Term frequency (TF) Kolikrát se dané slovo vyskytlo v dokumentu.

Inverse document frequency (IDF) Jak běžné nebo vzácné je slovo v celé sadě dokumentů. Čím blíže je hodnota k 0, tím je slovo běžnější. Tato metrika lze vypočítat tak, že se vezme celkový počet dokumentů, a vydělí se počtem dokumentů, které obsahují slovo.

Vynásobením těchto dvou čísel dostaneme TF IDF ohodnocení slova v daném dokumentu.. Vyšší ohodnocení znamená, že slovo je více relevantní pro dokument.

Ohodnocení TF IDF pro slovo t v dokumetu d, ze skupiny dokumentů D je vypočítáno následovně:

$$tfidf(t, d, D) = log_{10}(1 + tf(t, d)) \cdot idf(t, D) \tag{1}$$

2.4 Vector space model

Je algebraický model pro reprezentaci textových dokumentů (a všech objektů obecně) jako vektor identifikátorů. Ve vyhledávání je tento model použit pro reprezentaci dokumentů, kde každý dokument je reprezentován váhami TF-IDF, kde řádky jsou definovány slovy v dokumentu. Velikost vektoru tedy odpovídá všem slovům napříč všemi dokumenty. Pokud se dané slovo v dokumentu nevyskytuje, je hodnota TF-IDF pro dané slovo v dokumentu 0.

Jako metrika pro výpočet ohodnocení mezi dvěma vektory se používá *kosínova podobnost*.

$$\cos(\vec{q}, \vec{d}) = \text{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

q_i - váha TF-IDF slova i v dotazu

d_i - váha TF-IDF slova i v dokumentu

$| \overrightarrow{q} |$ a $| \overrightarrow{d} |$ jsou normalizované délky $\overrightarrow{q}$ a $\overrightarrow{d}$

2.5 Boolean vyhledávání

Booleovský dotaz se skládá z atomů (slova, základní dotazy), které jsou spojeny booleovskými operátory. Odpověď na dotaz je pak sada dokumentů. Dokument buď vyhovuje dotazu, nebo ne, to znamená, že tato metoda neumožňuje rozhodnout, který dokument je pro uživatele relevantnější než druhý. Nejčastěji používanými booleovskými operátory jsou:

OR: term1 OR term2 Dotaz vrací sadu dokumentů, ve kterých je přítomen term1 nebo term 2

AND: term1 AND term2 Získaná sada se skládá z dokumentů, které obsahují term1 a term2.

NOT: term1 NOT term2 Dotaz NOT vrací sadu dokumentů, ve kterých je term1 přítomen, ale term2 není.

3 Implementace

3.1 Invertovaný index

Níže jsou popsány třídy, které tvoří strukturu invertovaného indexu.

3.1.1 Třída InvertIndex

Třída představuje samotný invertovaný index. Obsahuje mapu ve spojení *slovo - InvertedIndexItem*. Třída dále obsahuje informaci o tom, jaký typ dat index obsahuje a počet indexovaných dokumentů.

3.1.2 Třída InvertIndexItem

Třída slouží pro držení informací o daném slově. Obsahuje informace o tom, v kterých dokumentech se slovo vyskytuje. Objekt tedy obsahuje mapu ve spojení *dokumentID - WordStats*. Dále se v objektu drží informace o hodnotě `IDF` slova.

3.1.3 Třída WordStats

Obashuje statistiky pro slovo v dokumentu. Třída tedy obsahuje informaci o tom, kolikrát se dané slovo v dokumentu vyskytlo, hodnotu `TF-IDF` a referenci na *DocumentBag*.

3.1.4 Třída DocumentBag

Představuje obal nad zaindexovaným dokumentem. Třída obsahuje seznam všech slov v dokumentu a vypočítanou euklidovskou střední hodnotu. Tím, že třída obashuje seznam všech slov v dokumentu, značně urychlý proces indexování, protože při výpočtu euklidovské střední hodnoty, není nutné procházet celý slovník a kontrolovat, zda dané slovo obsahuje dokument.

3.1.5 Načtení dat

Pro načtení dat jsou implementovány třídy, které implementují rozhraní `ILoader`. Pro získání správné třídy pro načtení dat je implementována továrna `LoaderFactory`, která na základě předaného výčtového typu `ELoaderType` vrátí příslušnou třídu. Cesty k jednotlivým souborům, které jsou tří-

dami načítány jsou nastavené ve třídě `Config` a jsou předány konstruktorem v továrně.

3.1.6 Preprocessing

Pro preprocessing byla zvolena následující konfigurace:

- ignorování stop slov
- pro stematizaci byl zvolen `CzechStemmerLight`
- pro tokenizaci byl zvolen `AdvancedTokenizer`
- odstranění akcentu po stematizaci
- převod na lowercase
- ignorování slov menších než 2

3.1.7 Uložené informace

Do indexu jsou spočítány následující informace, které se sním ukládají:

- základní struktura (slovo-dokument)
- euklidovská střední hodnota dokumentů, to znamená, že i její všechny mezivýpočty (`IDF`, `TF-IDF`, ...)

3.1.8 Princip vytvoření indexu

Na základě požadovaných dat se vybere třída pro načítání dat. Data se načtou, a předají se do indexu pro indexování. V průběhu indexace se procházejí načtená data, kde se průběžně obsah dokumentů vkládá do preprocessingu, který vrátí list upravených slov dokumentu pro indexaci. Tyto slova jsou postupně vkládána do invertovaného seznamu a také do `DocumentBagu` pro daný dokument (vždy se indexují slova jen pro jeden dokument, takže během zpracování jednoho dokumentu se vytvoří právě jeden `DocumentBag`). Po uložení všech slov dokumentů do invertovaného indexu se uloži jeho `DocumentBag` do listu pro pozdější vypočítání euklidovské střední hodnoty. Po zaindexování všech dokumentů do invertovaného indexu, jsou nastaveny ke slovům váhy `IDF` a `TD-IDF`, následně jsou vypočítány euklidovské střední hodnoty dokumentů (zde se prochází list uložených `DocumentBagů`, aby nebylo nutné hledat všechny unikátní dokumenty). Indexu se ještě nastaví informace o tom, jaké data jsou zaindexována. Po dokončení indexace je index uložen do souboru.

3.1.9 Načítání uloženého indexu

Jelikož je implementováno GUI, provádí se indexace ve dvou třídách. První je třída `TestTrecEval`, která slouží především pro kontrolu výsledků. Zde je iplementována podmínka, kde je nutné ručně změnit, jestli je požadadováno zaindexovat data, nebo ručně načíst data ze souboru. Pokud jsou data indexována, celý průběh i s uložením trvá okolo 100 sekund. Defaultně se data vždy indexují.

Implementace načítání dat v GUI je popsána v kapitole 3.3.1

3.2 Vyhledávání

3.2.1 Výběr vyhledávače

Pro vyhledávání jsou implementovány třídy, které implementují rozhraní `Searcher`. Pro získání správné třídy pro vyhledávání je implementována továrna `SearcherFactory`, která na základě předaného výčtového typu `ESearchType` vrátí příslušnou třídu.

3.2.2 VSM

Pro tento způsob vyhledávání je implementována třída `VectorSpaceModel-Searcher`. Třída nejdříve zaindexuje dotaz (*indexQuery()*) do vlastního slovníku a spočte `TF-IDF` hodnoty pro slova v dotazu a celkovou euklidovskou střední hodnotu.

Následně jsou získány všechny `DocumentBagy` všech dokumentů, ve kterých se vyskytuje některé slovo z dotazu. Potom je zavolána metoda *getResults()*, která spočte kosínovu podobnost pro `DocumentBag` dotazu s `DocumentBagy` dokumentů a seřadí je od nejrelevantnějších.

3.2.3 Boolean

Pro tento způsob vyhledávání je implementována třída `BooleanSearcher`. Pro parsování boolean dotazu byla použita knihovna *Lucene*, přesněji její třída `PrecedenceQueryParser`, která se postará o prioritu operátorů. Následně se v cyklu prochází listy jednotlivé subdotazy, které volají metodu *processQuery()*, která se postará o další operace (OR, AND, NOT). Pokud subdotaz obsahuje nějaký další subdotaz, metoda se volá rekurzivně, dokud nedojde k hodnotě (slovu).

3.3 GUI

Pro implementaci GUI byla zvolena knihovna *JavaFX*. GUI obsahuje dvě okna, jedno pro vyhledávání (`AppController`) a druhé pro editaci záznamů (`EditController`).

3.3.1 Spuštění

Aplikace zkontroluje jestli existuje uložený invertovaný index. Pokud ano, pokusí se soubor načíst. Pokud je soubor úspěšně načten, označí typ načtených dat a je možné vyhledávat. Pokud čtení souboru skončí chybou nebo soubor neexistuje, aplikace zaindexuje znovu data. Defaultní hodnota dat, které se indexují je možné změnit v třídě `Config`.

3.3.2 Vyhledávání dat

Při vyhledávání se načtou zvolené hodnoty radiobuttonů (typ vyhledávání, typ dat ve kterých vyhledávat). Nejdříve se zkontroluje, jestli zvolený typ dat je načten v paměti, pokud není, provede se indexování požadovaných dat.

Pokud je index v pořádku, nebo po zaindexování nových dat, se získá příslušný vyhledávač, podle typu vyhledávání a provede se vyhledávání. V seznamu pod vyhledáváním se zobrazí 10 nejrelevantnějších výsledků.

3.3.3 CRUD dat a doindexování

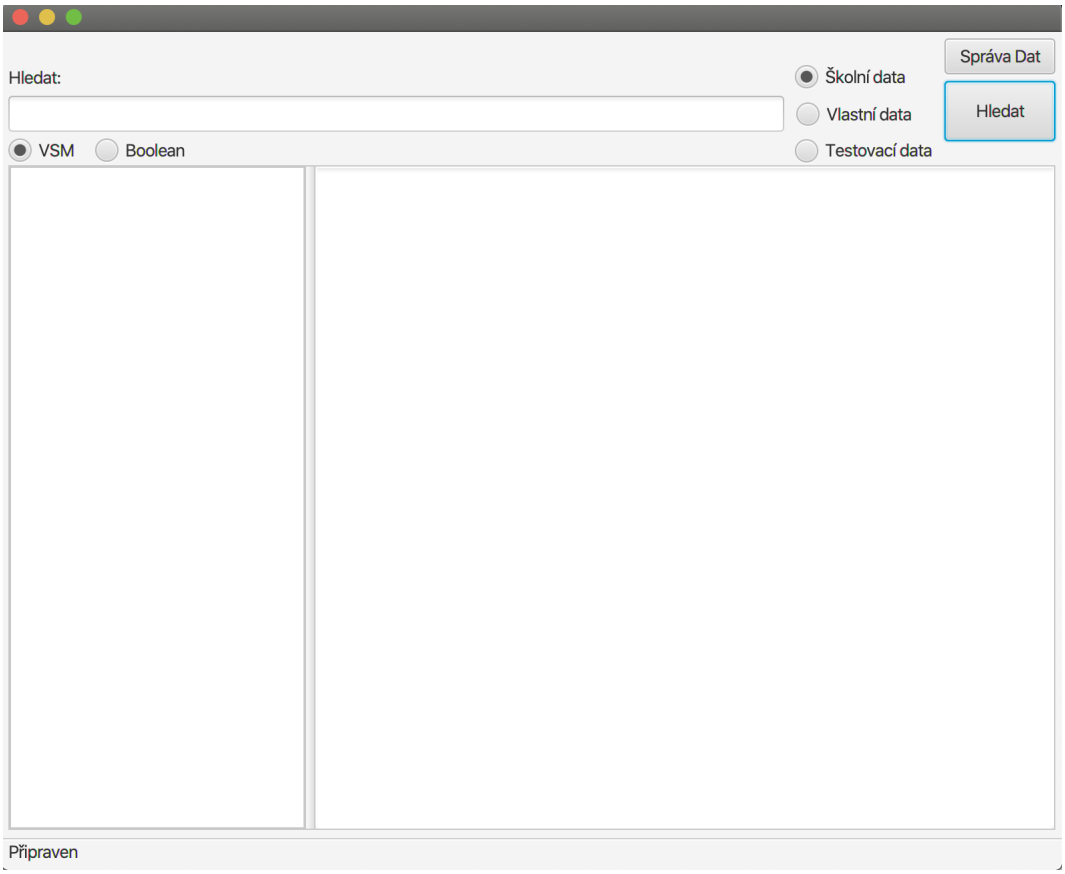
CRUD dat je možný ve správě dat. Po provedení změn je nutné data uložit a zaindexovat, pokud se změny neuloží nebudou promítnuty a budou ignorovány. Omezení, které tato implementace má je ta, že data které se načítají pro zobrazení informací (nastavení nového titulku) se drží jenom v paměti. Takže pokud se aplikace ukončí a znovu načte, index se načte se změnou, to znamená, že pokud budu hledat dokument podle nového titulku tak ho naleznu, ale v detailu se mi zobrazí starý titulek.

Je ošetřeno pokud uživatel bude chtít přidat další dokument s duplicitním ID, nebo editovat ID na existující ID. Kontrola, jaký dokument se má upravit/smazat je řešena přes krytý label, do kterého se ukládá původní hodnota ID. Výsledky operací jsou zobrazeny ve spodní části okna.

4 Uživatelská příručka

4.0.1 Start aplikace

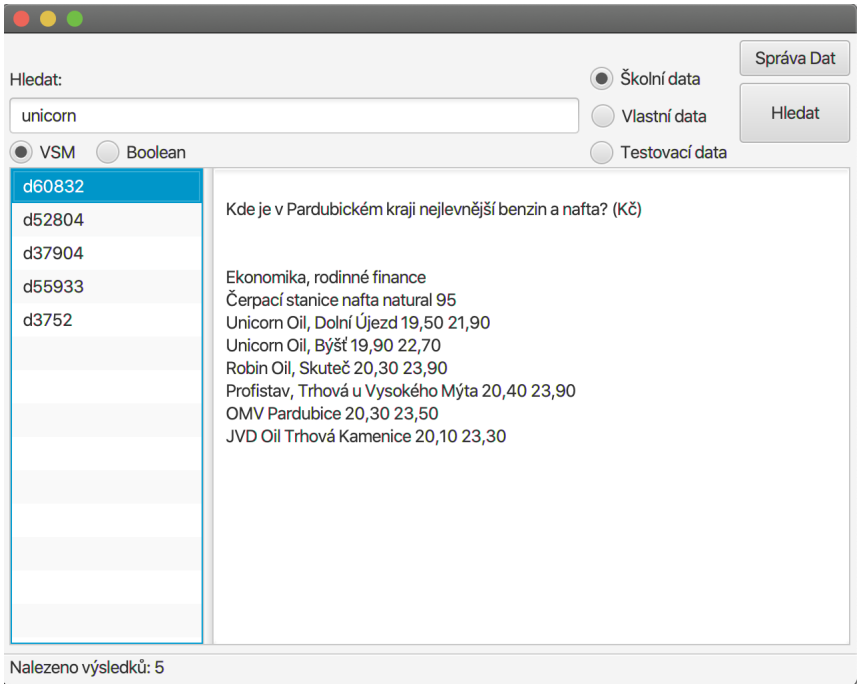
Pro start aplikace není nutné zadávat žádný parametr, pokud bude existovat uložený invertovaný index, aplikace se ho pokusí sama načíst. Po načtení by se mělo zobrazit následující okno aplikace. Hodnota typu dat se přepne podle aktuálně načteného indexu.



Obrázek 1: Apliakce po spuštění

4.1 Vyhledávání

Vyhledávání je možné zadáním dotazu do vstupního pole. Je možné si vybrat, jestli se má vyhledávat pomocí VSM nebo Boolean dotazu. Dále je možné vybrat typ dat. Pokud dojde ke změně typu dat, aplikace si na pozadí zaindexuje požadovaná data, proto se může stát, že aplikace na chvíli zamrzne. Zobrazuje se 10 nejrelevantnějších výsledků. Celkový počet výsledků je zobrazen ve spodní části okna. Kliknutím na výsledek se zobrazí jeho detail.

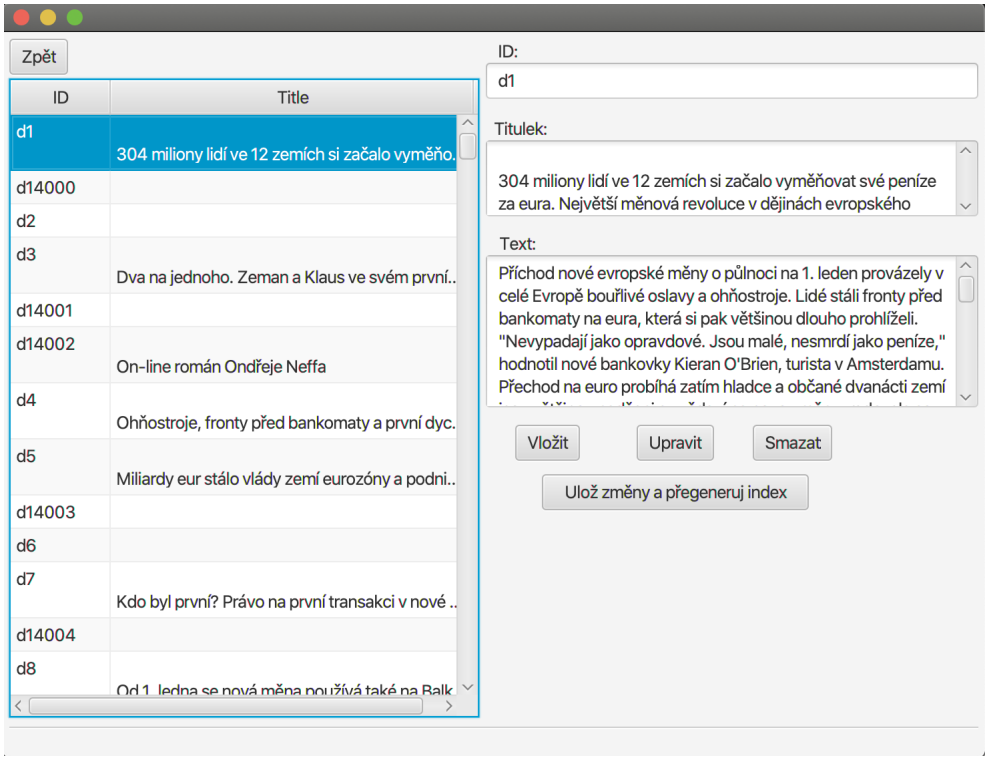


Obrázek 2: Ukázka vyhledávání

4.1.1 Editace

Editace dokumentů je možná stisknutím tlačítka „Správa dat“ vpravé horní části okna. Je možné provádět všechny CRUD operace. Po provedení změn je nutné změny uložit a zaindexovat, jinak budou ignorovány. Po uložení změn je možné se vrátit zpět do hlavního okna tlačítkem „Zpět“.

Pro vložení nového záznamu je nutné vyplnit alespoň unikátní ID dokumentu. Pro upravení/smazání je nutné nejprve vybrat dokument.



Obrázek 3: Editace

5 Závěr

Aplikace splňuje minimální požadavky semestrální práce. Nad rámec zadání byly implementovány funkčnosti:

- File-based index
- Pozdější doindexování dat
- GUI/webové rozhraní
- CRUD indexovaných dokumentů
- Dokumentace psaná v TEXu

Podle výsledků hodnota map odpovídá hodnotě 0,1701. Měření bylo omezeno pro 1000 výsledků na dotaz a hledalo se podle nadpisu a popisu.

```
num_q             all     50
num_ret           all     50000
num_rel           all     762
num_rel_ret       all     590
map               all     0.1701
gm_ap             all     0.0557
R-prec            all     0.1711
bpref             all     0.1868
recip_rank        all     0.3048
ircl_prn.0.00     all     0.3521
ircl_prn.0.10     all     0.2959
ircl_prn.0.20     all     0.2549
ircl_prn.0.30     all     0.2375
ircl_prn.0.40     all     0.2181
ircl_prn.0.50     all     0.1939
ircl_prn.0.60     all     0.1611
ircl_prn.0.70     all     0.1307
ircl_prn.0.80     all     0.1149
ircl_prn.0.90     all     0.0649
ircl_prn.1.00     all     0.0519
P5                all     0.1840
P10               all     0.1680
P15               all     0.1613
P20               all     0.1440
P30               all     0.1213
P100              all     0.0656
P200              all     0.0411
P500              all     0.0212
P1000             all     0.0118
```

Největší problém s kterým jsem se při implementaci setkal bylo „správné“ vyhodnocení boolean dotazu. Implementaci jsem převážně testoval na testovacích datech. Ukázky dotazů které jsem úspěšně otestoval jsou ukázány v třídě `BooleanSearcher`.

Snažil jsem se projekt strukturovat tak, aby byl přehledný, a proto jsem trochu pozměnil zadané rozhraní. Jelikož se ale jednalo jen o možnou šablonu, tak doufám, že to nebude problém.

Projekt je vedený na GITu.