FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Semestrální práce KIV/IR

Implementace vlastního systému automatické indexace a vyhledávání dokumentů

Jakub Hlaváč
A20N0050P
hlavja@students.zcu.cz

14. 04. 2020

Obsah

1 Zadání

1.1 Popis zadání

Cílem semestrální práce je naučit se implementovat komplexní systém s využitím hotových knihoven pro preprocessing. Vedlejším produktem bude hlubší porozumění indexerům, vyhledávacím systémům a přednáškám. Systém po předchozím předzpracování zaindexuje zadané dokumenty a poté umožní vyhledávání nad vytvořeným indexem. Vyhledávání je možné zadáním dotazu s logickými operátory AND, OR, NOT a s použitím závorek. Výsledek dotazu by měl vrátit top x (např. 10) relevantních dokumentů seřazených dle relevance. Semestrální práce musí umožňovat zaindexování dat stažených na cvičení (1. cvičení Crawler) a dat z portálu (složka TREC). Obě sady dat je možné zaindexovat nezávisle na sobě. Semestrální práce musí umožňovat zadávat dotazy z GUI nebo CLI (command line interface), při zadávání dotazů je možné vybrat index a model vyhledávání (vector space model vs boolean model). Výsledky vyhledávání obsahují i celkový počet dokumentů, které odpovídají zadanému dotazu.

1.2 Minimální funkčnost

- tokenizace
- preprocessing (stopwords remover, stemmer/lemmatizer)
- vytvoření in-memory invertovaného indexu s oběma sadami dat
- tf-idf model
- cosine similarity
- vyhledávání pomocí dotazu vrací top x výsledků seřazených dle relevance (tj. vektorový model - vector space model)
- vyhledávání s pomocí logických operátorů AND, OR, NOT (booleovský model)
- podrobná dokumentace (programátorská i uživatelská)
- podpora závorek pro vynucení priority operátorů

1.3 Nadstandardní funkčnost

- file-based index
- pozdější doindexování dat - přidání nových dat do existujícího indexu
- ošetření např. HTML tagů
- detekce jazyka dotazu a indexovaných dokumentů
- vylepšení vyhledávání
- vyhledávání frází (i stop slova)
- vyhledávání v okolí slova
- více scoring modelů
- indexování webového obsahu - zadám web, program stáhne data a rovnou je zaindexuje do existujícího indexu, další předzpracování normalizace
- další předzpracování normalizace
- GUI/webové rozhraní
- napovídání keywords
- podpora více polí pro dokument (např. datum, od do)
- CRUD indexovaných dokumentů
- zvýraznění hledaného textu v náhledu výsledků
- dokumentace psaná v TEXu
- vlastní implementace parsování dotazů bez použití externí knihovny
- implementace dalšího modelu (použití sémantických prostorů)

2 Aplikace

Semestrální práce byla implementována jako webová aplikace a skládá se z BackEnd (Spring aplikace) a FrontEnd (apliakce Angular verze 11) části. Tyto dvě aplikace spolu komunikují pomocí nadefinovaného REST API.

2.1 BackEnd

Jedná se o standardní Spring aplikaci, která byla implementována pro Javu 11. Aplikace je rozdělena do několika balíků, které obsahují potřebné třídy.

- *config* - balík obsahující třídu s konstantami aplikace
- *crawler* - balík obsahující samostatně spustitelnou metodu, která vytěží data z webu
  - **CrawlerVSCOM** - spustitelná třída pro vytěžení dat z webu
- *data* - balík obsahující třídy pro práci s daty
  - **ArticleRepository** - třída in memory databáze pro ukládání indexovaných článků
- *dtos* - balík obsahující objekty pro komunikaci pomocí REST API
- *indexing* - balík obsahující třídy s logikou pro vytváření indexu a následného vyhledávání
  - **Index** - třída reprezentující index
  - **InvertedList** - třída reprezentující invertovaný seznam
- *preprocessing* - balík obsahující potřebné třídy pro předzpracování vstupních textových dat
  - **BasicPreprocessing** - třída s logikou pro předzpracování textových dat
- *utils* - balík obsahující potřebné třídy pro práci se soubory
- *web* - balík obsahující controller pro komunikaci s FrontEndem
  - **Controller** - třída s kompletní implementací REST controllerů pro ovládání aplikace

- **IRApplication** - hlavní třída Spring aplikace
- **TestTrecEval** - třída pro testování indexování a následnému vyhledávání

2.1.1 REST rozhraní

Implementace REST API je provedena v balíku **web** a třídě **Controller**.

- GET /api/initData - načtení /data/articles.json do indexu
- GET /api/initTrec - načtení /TERC/czechData.bin dat do indexu
- POST /api/query - vyhledávácí dotaz nad indexem, obsahuje tělo typu QueryModel
- GET /api/saveIndex - uložení indexu, parametr fileName s názvem indexu
- GET /api/loadIndex - načtení indexu, parametr fileName s názvem indexu
- GET /api/clearIndex - smazání indexů
- GET /api/indexStatus - načtené indexy v paměti
- GET /api/savedIndex - uložené indexy
- GET /api/articles - všechny články, parametr indexName s názvem indexu
- GET /api/article/id - článek podle id, parametr indexName s názvem indexu
- DELETE /api/article/id - smazání článku z indexu, parametr indexName s názvem indexu
- PUT /api/article - vložení nového článku do indexu, parametr indexName s názvem indexu, tělo obsahuje typ ArticleModel
- POST /api/article - úprava existujícího článku, parametr indexName s názvem indexu, tělo obsahuje typ ArticleModel

2.2 FrontEnd

Webová část aplikace byla naprogramována pomocí Angularu 11. Pomocí OpenAPI definice byly z BackEnd části vygenerovány potřebné metody pro HttpClienta kvůli komunikaci. Jelikož se nejedná o stěžejní část aplikace, bude v této kapitole popsáno jenom stručné členění projektu.

Aplikace je rozdělena do *modulů* a ty jsou následně sestavovány z *komponent*. Tudíž jeden modul může obsahovat více komponent.

Komponenty, které jsou používány ve více modulech, jsou implementovány ve složce *src/app/shared*. Zde se jedná především o nav bar a kompletní balík funkcí pro komunikaci s BackEndem ve složce *./api*.

Nejdůležitější moduly aplikace:

- **articles** - obsahují implementaci zobrazení seznamu zaindexovaných článků v daném indexu a jejich CRUD manipulaci s nimi (*smazání*, *editaci*, *přidání nového*)
- **index** - obsahuje implementaci pro vyhledávání ve zvoleném indexu a procházení výsledků vyhledávání
- **nav-bar** - obsahuje funkce pro manipulaci s indexy (*zaindexování*, *uložení indexu do souboru*, *načtení indexu ze souboru*)

3 Implementace primární funkčnosti

V této kapitole bude popsána implementace jednotlivých primárních částí aplikace.

3.1 Crawler

Pro získání dat, která budou následně indexována bylo využito kódu z prvního cvičení tohoto předmětu. Implementace crawleru je obsažena v balíku **crawler**. Pro vytěžení zadaného webu je využito možností *Selenia* s použitím Chromedriveru. Díky tomu je možno procházet webowé stránky a stahovat z nich potřebný dotaz. V mém případě vybraného webu *https://cz.cw-nn.com/* jsem nenarazil na omezení v maximálním počtu přístupů na stránku za jednotku času a nebylo tedy implementovat zpomalovací mechanismus. Vytěžená data jsou ukládána do objektů *ArticleModel* a následně všechny uloženy do JSON souboru.

3.2 Předzpracování

Při každém indexování článku či dotazu je na zadaný text aplikována logika předzpracování, která je implementována v balíku **preprocessing**. V této fázi je provedeno několik kroků, které upraví vkládaná data do formátu, který je pro potřeby indexace vhodnější. Jedná se především o kroky:

- převedení na malá písmena - pro indexaci a následné vyhledávání i ve slovech, které jsou na začátku věty či názvech
- odstranění diakritiky
- odstranění stop slov - v každém jazyce existují slovní druhy, které není potřeba indexovat (předložky, spojky atd.)
- tokenizace - převedení slov do základního tvaru (infinitiv, kořen slova)

3.3 Index

Třída **Index** uchovává kompletní informace o založeném indexu.

3.3.1 Invertovaný seznam

Jedná se o datovou strukturu (třída **InvertedList**), která se používá k fulltextovému vyhledávání ve velkých datových sadách. Obecně se jedná o posloupnost předzpracovaných slov, ke kterým je přiřazen seznam dokumentů (článků), ve kterých se vyskytují a zároveň je přidána informace o počtu výskytů takového slova v daném dokumentu. Tato posloupnost je ve třídě implementována pomocí HashMapy, ve které jsou na rozdíl od maticové reprezentace ukládána pouze reálná data. Maticová reprezentace invertovaného systému zde nebyl zvolena z důvodu silné paměťové neefektivnosti, jelikož při zvolení tohoto přístupu je potřeba uchovávat informace i pro dokumenty, které dané slovo neobsahují, jelikož se jedná o maticy v reprezentaci [množina slov][množina dokumentů]. Tato mapa je uložena do atributu *invertedList* a má následující reprezentaci:

<term -> <documentId, countOfTermInDocument>> (mapa, kde je použit jako klíč daný term a hodnotou je další mapa, ve které je klíčem id dokumentu a jako hodnota je vložen počet výskytů v daném dokumentu).

3.4 Vyhledávání

3.4.1 Vektorový model

Požadovaný vektorový model vyhledávání byl v rámci práce implementován do třídy invertovaného seznamu. Všechny potřebné hodnoty jsou pro rychlejší vyhledávání vypočteny při indexaci dokumentů. Ve vektorovém modelu vyhledávání jde o princip přiřazení skóre relevance každému z dokumentů na základě ohodnocení vyhledávání, kdy je využito vektorů, které jsou pro každý dokument vytvořeny. V rámci vektorového modelu je i dotaz považován za dokument a je nutno ho reprezentovat daným vektorem. Následně jsou výsledné vektory porovnány pomocí kosinové podobnosti.

IDF

Vektor hodnot *inverse document frequency* je uložen v reprezentaci invertovaného seznamu v atributu **idf**. Tento atribut je reprezentován jako HashMapa, kdy klíčem je term a hodnotou výsledek vzorečku:

$$idf_t = \log \frac{N}{df_t},$$

kde df_t termu t představuje počet indexovaných dokumentů, ve kterém se daný term t vyskytl a N je celkový počet zaindexovaných dokumentů.

TF-IDF

Vektor TF-IDF váhy dokumentů je v invertovaném seznamu uložen v atributu **tfIdf**. Tuto váhu dostaneme pomocí vzorce:

$$tfidf = wf_{t,d} * idf_t$$

, kdy váha $wf_{t,d}$ je vypočítána vzorcem:

$$wf_{t,d} = 1 + \log tf_{t,d}$$

pokud se daný term t vyskytl v dokumentu d alespoň jednou, pokud ne je váha rovna 0 a váha idf_t je uložena v atributu **idf** viz. 3.4.1 IDF.

Kosinová podobnost

Implementace výpočtu kosinové podobnosti dotazu a dokumentu odpovídá vzorci:

$$\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{||\vec{q}|| \cdot ||\vec{d}||}$$

kdy $\vec{q}$ je vektor termů dotazu a $\vec{d}$ je vektor termů dokumentu.

3.4.2 Booleovský model

Booleovský model vyhledávání byl implementován za použití knihovny *Apache Lucene*, která se stará o zpracování dotazu. Výsledné dokumenty jsou všechny ohodnoceny skórem 1, jelikož odpovídají hledanému dotazu. Tento typ vyhledávání akceptuje logické operátory **AND**, **OR** a **NOT** a je možno vynutit prioritu operátorů pomocí závorek. Používání závorek je silně doporučováno i při jednodušších dotazech, jelikož ne vždy se použitá knihovna zachová podle očekávání uživatele. Použití *Lucene* knihovny podporuje zadávání dotazů ve formátu: operand operátor operand, což odpovídá **infixové notaci**. Knihovna se následně postará o převedení na **prefixovou notaci** (operátor operand operand). Pro reprezentaci výsledné notace je použit strom, kdy jsou rekurzivně přidány slova dotazu jako listy daného operátoru. Následně je takto vytvořený strom rekurzivně procházen od listů po kořen stromu a podle typu operátoru je nad odpovídajícími seznamy příslušných dokumentů provedena logická matematická operace:

- AND - průnik
- OR - sjednocení
- NOT - doplněk

4 Implementace nadstandardní funkčnosti

V rámci vypracování práce byly splněny i některé požadavky na nadstandardní funkcionalitu. Všechny tyto implementované požadavky budou vyjmenovány v této kapitole.

4.1 File-based index

Tato funkcionalita byla implementována jako možnost uložení indexu z paměti do souboru a možnost opětovného načtení. Do souboru je v tomto případě uložena datová reprezentace instance invertovaného seznamu za pomoci rozhraní **Serializable**

4.2 GUI/webové rozhraní

Implementace kompletního webového rozhraní v Angularu. Webové rozhraní následně komunikuje s alikací pomocé REST API.

4.3 Pozdější doindexování dat

Tato možnost byla implementována v rámci implementace další nadstandardní funkcionality viz 4.3. Kdy bych ji považoval za splněnou v případě možnosti vkládat nový článek do již existujícího indexu. V rámci přidání je následně opětovně přepočítána hodnota **TF-IDF**.

4.4 CRUD indexovaných dokumentů

Zaindexované dokumenty podporují CRUD operace:

- create - vytvoření nového článku a jeho zaindexování
- read - všechny zaindexované články je možné zobrazit přes webové rozhraní
- update - možnost již zaindexovaný článek upravit

- delete - možnosti zaindexovaný článek smazat

V případech create, update a delete je přepočítána hodnota **TF-IDF** pro invertovaný seznam.

4.5 Podpora více polí pro dokument

Indexované články podporují více polí, která je možno zobrazovat a editovat. Nicméně nad těmito poli nejsou tvořeny indexy pro vyhledávání.

4.6 Zvýraznění hledaného textu v náhledu výsledků

V rámci implementace webového rozhraní bylo v náhledu výsledků zajištěno zvýraznění vyhledávaných slov v dotazu. Z dotazu byly odfiltrovány pomocné znaky *AND*, *OR*, *NOT*, *(*, *)* a tudíž jsou zvýrazňovány pouze operandy.

4.7 Dokumentace psaná v TEXu

Kompletní dokumentace byla psána pomocí projektu Overleaf.

5 Závěr

5.1 Výsledky evaluace

Otestování aplikace jsem provedl pomocí přidaného výkonného kódu ve třídě **TestTrecEval**. V této třídě jsem definoval svůj index a zaindexoval data ze souboru *czechData.bin*. Následně jsem nad takto zaindexovanými daty prováděl několik vyhledávání, kdy jsem zkoušel různé kombinace polí, přes která se vyhledávalo. Výsledky jsem následně vložil do připraveného C scriptu, který nad nimi provedl pro mě neznámou transformaci a vyhodnocení. Výsledky tohoto scriptu pro různé kombinace polí vyhledávání (pro jednoduchost uvádím jen hodnotu **map**:

- pole **title, description, narrative** *map = 0.2388*
- pole **title, narrative** *map = 0.2337*
- pole **description, narrative** *map = 0.2180*
- pole **title, description** *map = 0.1968*
- pole **description** *map = 0.1698*
- pole **narrative** *map = 0.1676*
- pole **title** *map = 0.1647*

Podrobnější výsledky scriptu jsou přiloženy spolu s prací v projektu BackEnd a složce **TREC**.

5.2 Zhodnocení

Hlavní část práce byla zpracovávána již v průběhu semestru, kdy jsem plnil dílčí úkoly na jednotlivé části práce. Což je podle mě pro studenta příjemné pojetí cvičení. Následně je potřeba tyto fragmenty zkompletovat a utvořit z nich funkční celek. Tento celek jsem následně doplnil o možnost vyhledávání pomocí booleovského modelu. Zde bylo potřeba alespoň trochu nastudovat dokumentaci knihovny což také pokládám za velké plus, jelikož jsem se s tím v rámci jiných semestrálních prací nesetkal. Semestrální práce splňuje minimální akceptační kritéria z kapitoly 1.1 a několik dalších bodů z nadstandardní funkcionality, které jsou popsány v kapitole 3.4.2.

Uživatelská příručka

C Spuštění aplikace

Před spuštěním aplikace je potřeba do složky TREC v projektu BackEnd nakopírovat zdrojový soubor **czechData.bin**. Následně je kompletní semestrální práce spustitelná v Dockeru pomocí jednoduchého příkazu v rootu odevzdávaného archivu:

docker compose up -d –no-deps –build –force-recreate

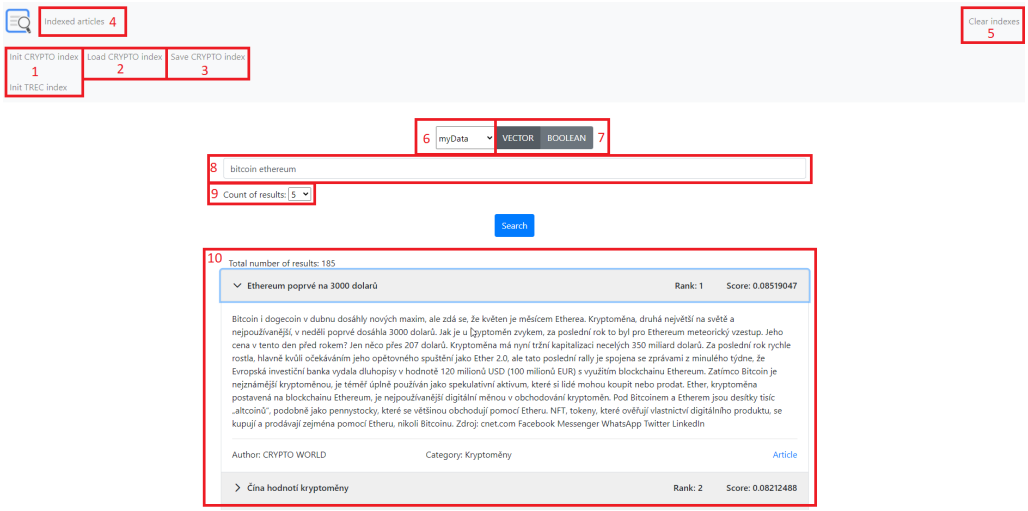
Tento příkaz zkompiluje FrontEnd a BackEnd, vytvoří containery a ty nasadí do Dockeru. Následně vystaví obě dvě aplikace na příslušné porty:

- http://localhost:80 - webové rozhraní aplikace
- http://localhost:8080 - BackEnd aplikace pro testování pomocí Postman

D Index

V této kapitole budou vysvětleny veškeré ovládací prvky hlavní obrazovky viz obr. 5.1.

- 1. tlačítka pro inicializaci indexů
  - Init CRYPTO index - vytvoří index z dokumentů v /data/articles.json
  - Init TREC index - vytvoří index z dat v souboru /TREC/czechData.bin
- 2. tlačítko se zobrazí pokud je index uložený v souboru a lze ho načíst
- 3. tlačítko pro uložení seznamu z paměti aplikace do souboru
- 4. tlačítko pro přesun na obrazovku s výpisem zaindexovaných dokumentů (viz.D)
- 5. vymazání všech indexů z paměti aplikace
- 6. vybrání indexu, ve kterém bude aplikace hledat
  - myData - index souboru articles.json
  - trecData - index souboru czechData.bin



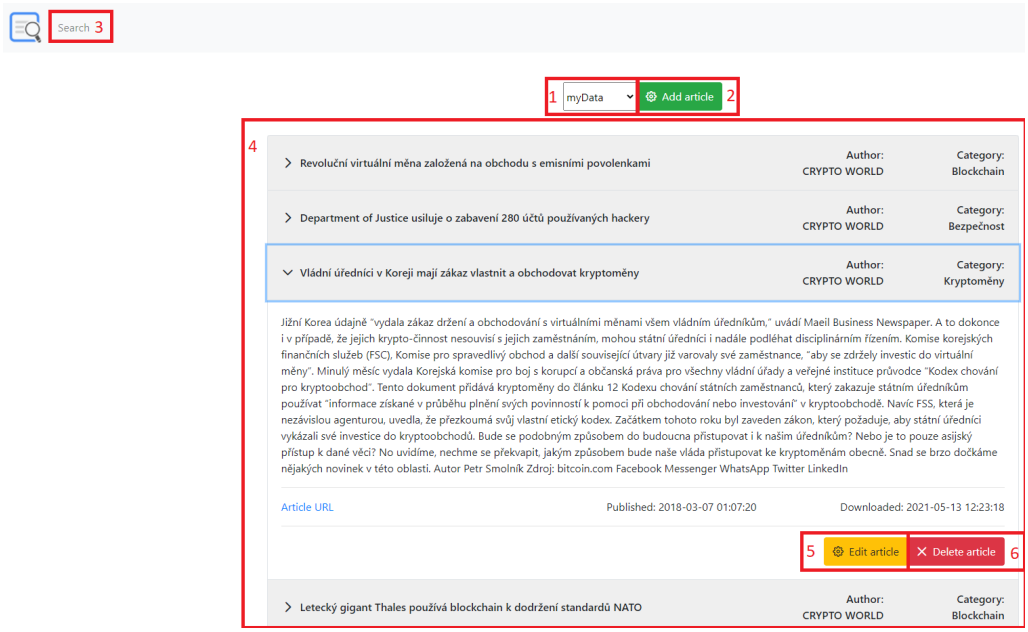
Obrázek 5.1: Hlavní obrazovka.

- 7. přepínací tlačítko, určující použitý vyhledávací model
- 8. pole pro zadávání dotazu
- 9. počet zobrazených dokumentů výsledku vyhledávání
- 10. seznam výsledků vyhledávání

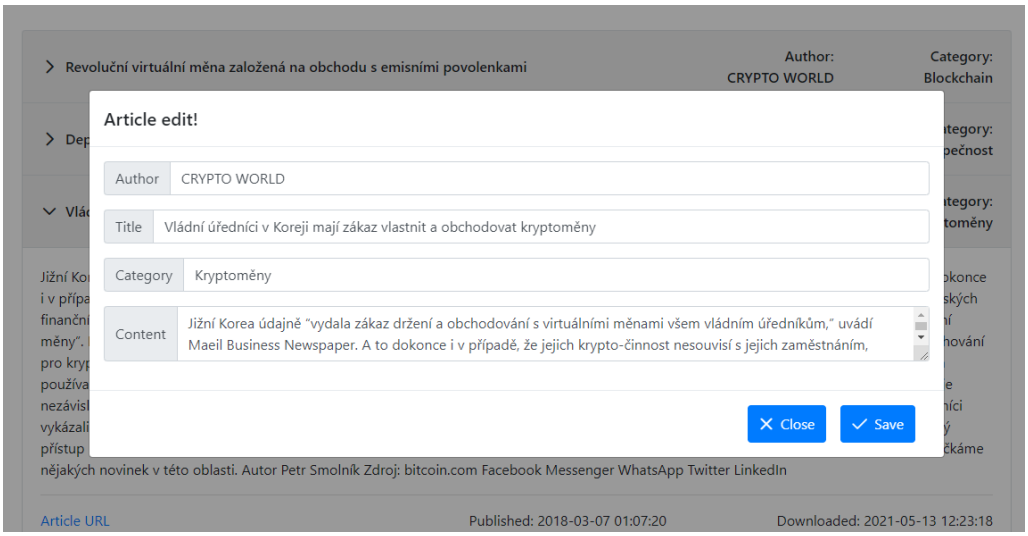
E Články

Vysvětlení ovládacích prvků stránky se zaindexovanými články viz. obrázek 5.2. Veškeré akce s články jsou prováděny nad vybraným indexem.

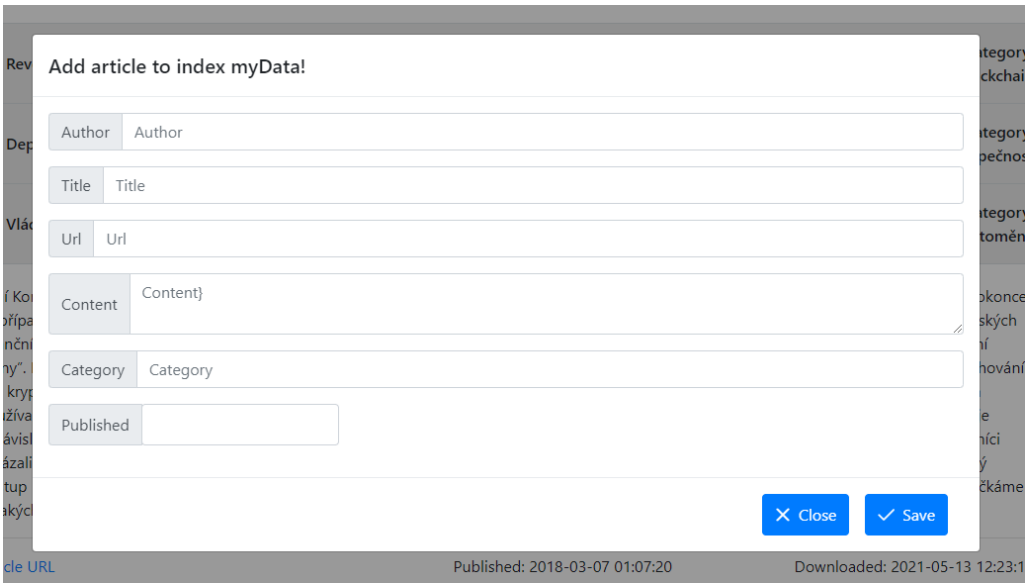
- 1. vybrání indexu se kterým chci pracovat
- 2. tlačítko pro přidání článku do indexu viz. obrázek 5.4
- 3. tlačítko pro návrat na obrazovku s vyhledáváním
- 4. zaindexované články
- 5. tlačítko pro zobrazení editace článku viz. obrázek 5.3
- 6. tlačítko pro smazání článku z indexu



Obrázek 5.2: Obrazovka se zaindexovanými články.



Obrázek 5.3: Úprava článku.



Obrázek 5.4: Přidání článku.