SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DATABÁZY: TECHNOLÓGIE NA FULLTEXT PREHĽADÁVANIE (ELASTICSEARCH, MEILISEARCH, ALGOLIA)

SEMINÁRNA PRÁCA

2022 Adam Harnúšek, Attila Mucska, Samuel Kobera, Imrich Budinský

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DATABÁZY: TECHNOLÓGIE NA FULLTEXT PREHĽADÁVANIE (ELASTICSEARCH, MEILISEARCH, ALGOLIA)

SEMINÁRNA PRÁCA

Študijný program:	Aplikovaná informatika
Predmet:	I–ASOS – Architektúra softvérových systémov
Prednášajúci:	RNDr. Igor Kossaczký, CSc.
Cvičiaci:	Ing. Stanislav Marochok

Bratislava 2022 Adam Harnúšek, Attila Mucska, Samuel Kobera, Imrich Budinský

Obsah

Úvod

Vyhľadávanie na webe spôsobilo revolúciu v spôsobe používania internetu. Google dokáže prehľadávať celý internet a používateľom, ktorí sa snažia nájsť obsah, poskytuje skvelý zážitok. Prečo teda vyhľadávanie v mnohých aplikáciách, ktoré dnes používame, chýba alebo nie je optimálne? Väčšina aplikácií má oveľa menej údajov na vyhľadávanie ako Google, no napriek tomu majú slabšiu výkonnosť.

Cieľom tohto projektu je implementovať plnotextové vyhľadávanie pomocou troch full-textových servisov: Meilisearch, Algolia a Elasticsearch. Následne porovnáme výkonnosť naimplementovaných algoritmov na základe rýchlosti vyhľadávania v texte, a nakoniec vyhodnotiť dosiahnuté výsledky.

1 Full-text vyhľadávanie

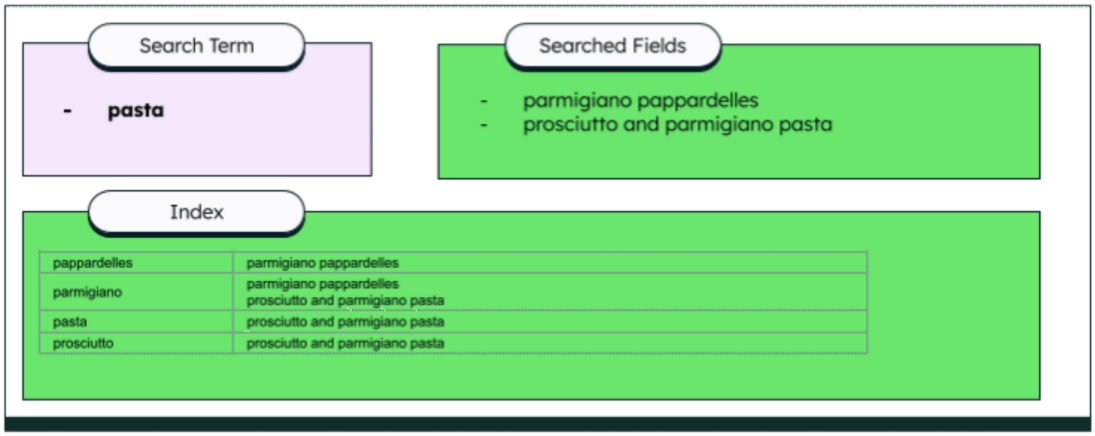
Full-textové vyhľadávanie znamená vyhľadávanie nejakého textu v rozsiahlych textových údajoch uložených elektronicky a vrátenia výsledkov, ktoré obsahujú niektoré alebo všetky slová z dotazu. Naproti tomu tradičné vyhľadávanie by vrátilo presné zhody. Zatiaľ čo tradičné databázy sú skvelé na ukladanie a získavanie všeobecných údajov, vykonávanie full-textového vyhľadávania je náročné. Na dosiahnutie tohto cieľa sú často potrebné dodatočné nástroje.

Pri vyhľadávaní textu full-textové vyhľadávanie znamená techniku na vyhľadávanie v jednom dokumente alebo kolekcie vo full-textovej databáze. Full-textové vyhľadávanie sa líši od vyhľadávania založeného na metadátach alebo na častiach pôvodných textov zastúpených v databázach (napr. názvy, abstrakty, vybrané časti alebo bibliografické odkazy).

1.1 Indexovanie

Pri práci s malým počtom dokumentov je možné, aby full-textový vyhľadávač priamo skenoval obsah dokumentov s každým dotazom, čo je stratégia nazývaná "sériové skenovanie". Jedným príkladom pre tento spôsob je nástroj príkazového riadku grep.

Indexovanie je možné vykonať rôznymi spôsobmi, ako je dávkové indexovanie alebo prírastkové indexovanie. Index potom funguje ako rozsiahly slovník pre všetky zodpovedajúce dokumenty. Na extrakciu údajov sa potom môžu použiť rôzne techniky. Apache Lucene, knižnica vyhľadávania s otvoreným zdrojom, používa na nájdenie zodpovedajúcich položiek inverzný index. V prípade nášho vyhľadávania v ponuke každé slovo odkazuje na zodpovedajúcu položku ponuky.



Obr. 1: Indexovanie

Táto technika je oveľa rýchlejšia ako vyhľadávanie reťazcov pre veľké množstvo údajov. Tieto indexy však vyžadujú určité miesto na disku a pri vytváraní môžu spotrebovať veľa zdrojov.

1.1.1 Postup indexovania full-textového vyhľadávania

Kľúčom k efektívnemu full-textovému vyhľadávaniu je vytváranie indexov. Proces vytvárania indexu v podstate prechádza cez každé textové pole množiny údajov. Pre každé slovo sa začne odstránením akejkoľvek diakritiky (značky umiestnené nad alebo pod písmenami, ako napríklad é, à, a ç vo francúzštine). Potom, na základe použitého jazyka, algoritmy odstránia výplňové slová a ponechajú iba kmeň pojmov. Týmto spôsobom sú "to eat," "eating", a "ate" v angličtine všetky klasifikované ako rovnaké kľúčové slovo „eat". Potom sa zmení veľkosť písmen tak, aby sa používali iba veľké alebo malé písmená. Presný proces indexovania určuje použitý analyzátor.

1.2 Technológie

1.2.1 Lucene

Obr. 2: Apache Lucene

Apache Lucene je bezplatná softvérová knižnica vyhľadávacieho nástroja s otvoreným zdrojovým kódom, ktorú pôvodne v jazyku Java napísal Doug Cutting. Podporuje ho nadácia Apache Software Foundation a je vydaný pod licenciou softvéru Apache. Lucene je široko používaný ako štandardný základ pre nevýskumné vyhľadávacie aplikácie.

Hoci je Lucene vhodný pre akúkoľvek aplikáciu, ktorá vyžaduje full-textové indexovanie a vyhľadávanie, je uznávaný pre svoju užitočnosť pri implementácii internetových vyhľadávačov a lokálneho vyhľadávania na jednej lokalite.

Lucene používa na vykonávanie fuzzy vyhľadávania algoritmus "edit distance".

Niekoľko projektov, ktoré používajú Lucene:

- Apache Solr
- Elasticsearch
- MongoDB
- OpenSearch

1.2.2 Algolia

Obr. 3: Algolia

Algolia poskytuje vyhľadávanie ako službu, ktorá ponúka vyhľadávanie na webe na webovej lokalite klienta pomocou externe hosťovaného vyhľadávacieho nástroja. Hoci vyhľadávanie na stránke je už dlho dostupné od všeobecných poskytovateľov vyhľadávania na webe, ako je Google, zvyčajne sa to robí ako podmnožina všeobecného vyhľadávania na webe. Vyhľadávací nástroj prehľadáva web ako celok, vrátane klientskej lokality, a potom ponúka funkcie vyhľadávania obmedzené len na túto cieľovú lokalitu. Ide o rozsiahlu a zložitú úlohu, ktorá je dostupná len pre veľké organizácie v rozsahu Google alebo Microsoftu.

Produkt Algolia indexuje iba stránky ich klientov, čím zjednodušuje vyhľadávanie. Údaje pre klientske stránky sa odosielajú z klienta do Algolie prostredníctvom RESTful JSON API, potom sa na webové stránky klienta pridá vyhľadávacie pole. Tento vyhľadávací model je určený na replikáciu výhody úplného interného vyhľadávacieho nástroja, ale so zjednodušeným nastavením.

1.2.3 Meilisearch

Obr. 4: Meilisearch

Meilisearch je flexibilný a výkonný vyhľadávací nástroj zameraný na používateľa,

ktorý možno pridať na akúkoľvek webovú stránku alebo aplikáciu. Meilisearch je databáza, ktorá ukladá indexované dokumenty spolu s údajmi potrebnými na vykonanie bleskového vyhľadávania.

1.2.4 Elasticsearch

Elasticsearch je vyhľadávací nástroj založený na knižnici Lucene. Poskytuje distribuovaný full-textový vyhľadávací nástroj s podporou viacerých nájomníkov s webovým rozhraním HTTP a dokumentmi JSON bez schém. Elasticsearch je vyvinutý v jazyku Java a je licencovaný v rámci zdrojovej licencie Server Side Public License a licencie Elastic, zatiaľ čo ostatné časti spadajú pod proprietárnu (dostupne zdrojovú) licenciu Elastic.

1.3 Príklad použitia

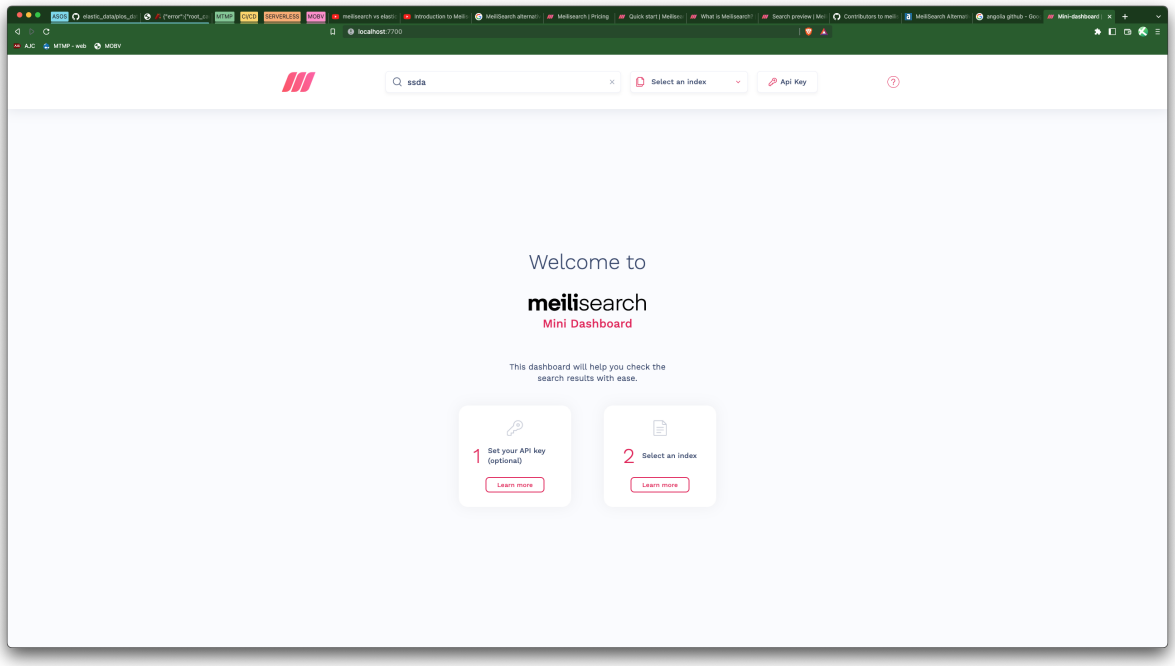
Full-textové vyhľadávanie môže mať mnoho rôznych použití – napríklad hľadanie jedla v jedálnom lístku reštaurácie alebo hľadanie konkrétnej funkcie v popise položky na webovej stránke elektronického obchodu. Okrem vyhľadávania konkrétnych kľúčových slov môžete full-textové vyhľadávanie rozšíriť o funkcie vyhľadávania, ako sú fuzzy-text a synonymá. Výsledky pre slovo ako „cestoviny“ by preto vrátili nielen položky ako „Cestoviny s mäsovými guľkami“, ale mohli by vrátiť aj položky ako „Fettuccine Carbonara“ s použitím synonyma alebo „Slanina a pesto chlieb“ pomocou nejasného vyhľadávania.

2 Meilisearch

Meilisearch je opensource projekt, ktorý tiež ponúka aj platené cloud-ové riešenie. Ponúka RESTful search API na vyhľadávanie v texte. Je výbornou voľbou pre malé a stredne veľké spoločnosti kde dokáže v pokoji pokryť ich požiadavky. Ako už skoro každá knižnica, sľubujú ***Blazingfastsearch*** (response $<$ 50ms).

Hlavnou motiváciou vytvorenia tohto projektu bolo priniesť na trh jednoduchý, prehľadný a výkonný search engine, ktorý podľa autorov v roku 2018 na trhu nebol. Autori ale sami hovoria, že pre obrovské kolekcie ($>$ 10 Mil dokumentov) Meilisearch nie je najindeálnejšie riešenie.

Okrem RESTFUL api ponúka aj DEMO na stránke https://crates.meilisearch.com/. Ak by sme chceli však ísť o krok ďalej, tak postiahnutí si Meilisearch na naše zariadenie, si automaticky môžeme hostovať klienta (predvolene na adrese 127.0.0.1:7700), kde si funkcionalitu môžeme testovať úž na vlastných dátach.



Obr. 5: Meilisearch preview na localhoste

2.1 Funkcie/Vlastnosti

Vyhľadávanie počas písania Meilisearch ponúka výsledky už počas písania dopytu, toto je možné vďaka vysokej rýchlosti vyhľadávania.

Relevancia Vďaka základným pravidlám relevancie, ktoré sa dajú upraviť vieme zabez-

pečiť, aby boli najvyššie hodnotené výsledky vždy relevantné. O týchto pravidlách si povieme v sekcii 2.4.

Tolerancia preklepov Opäť prispôsobiteľná funkcionalita, ktorá pomáha zaručiť, že používateľ dostane výsledky, ktoré hľadá.

Synonymá V Meilisearch môžu byť synonymá obojsmerné, ako napr. slová `Résumé` a `CV`, alebo jednosmerné, medzi slovami `phone` a `iphone`, pretože každý iphone je telefón, ale naopak nie.

Filtre nám umožňujú definovať ďalšie podmienky vyhľadávania. Ako napríklad pri databáze filmov, môžeme filtre použiť iba na názov, alebo aj názov aj žáner. Alebo vieme filtrovať podľa dátumu vydania iba filmy vydané pred rokom 2000 a podobne.

Zoraďovanie To, že Meilisearch samotný ponúka zoraďovanie je dôležité pre výkon. Keby nám dodá všetky výsledky a my ich zoraďujeme následne, môže sa vykonať až N operácií navyše.

Vyhľadávanie fráz umožňuje používateľovi zabaliť frázu do dvojitých úvodzoviek, kedy vyhľadávač vráti iba výsledky obsahujúce celú frázu.

2.2 Inštalácia

Meilisearch je veľmi jednoduché nainštalovať na väčšine zariadení.

2.2.1 Lokálne

Je možné ho stiahnuť ako binárny súbor a spúšťať priamo v CMD. Vybuildovať si celý search engine manuálne. Dokonca je aj pod package managermi ako je Homebrew alebo APT. Najpríjemnejšia cesta je ale určite Docker.

2.2.2 Cloud

Aby autori nerobili všetko len zadarmo, ponúkajú aj cloud platené riešenie. V ňom je už zahrnutá všetká konfigurácia a aj support.

2.2.3 Developerské nástroje (SDKs) a knižnice

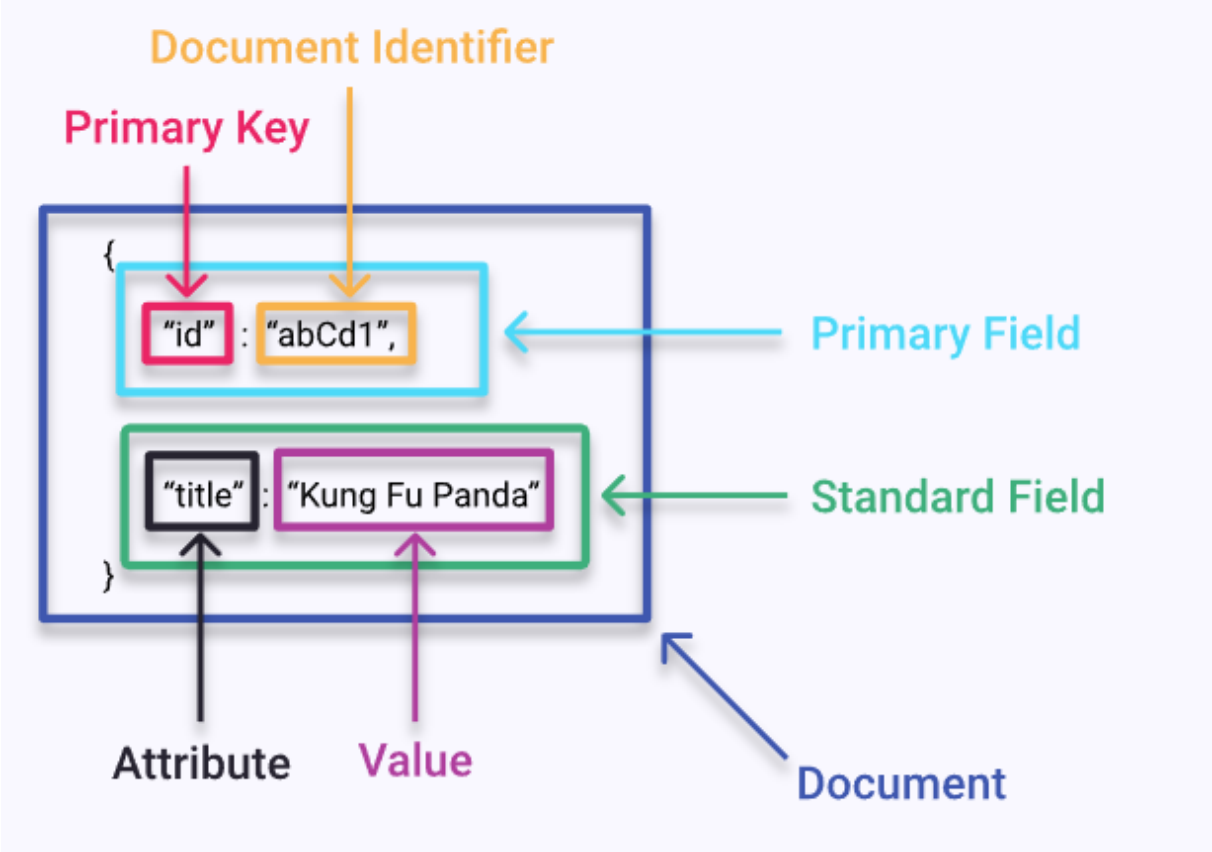
Po nainštalovaní samotného Meilisearch ho potrebujeme aj napojiť na náš program. Keďže sa autori zamerali na jednoduchosť pre používateľov ponúkajú SDKs a knižnice pre množstvo populárnych jazykov, frameworkov, frontend nástrojov, a iných. Pre príklad uvedieme:

- .NET, Dart, Golang, Java, JavaScript, PHP, Python, Ruby, Rust, Swift

- Laravel, Ruby on Rails, Symfony
- Angular, React, Vue
- meiliesearch-aws, meilisearch-digitalocean, meilisearch-kubernetes
- VuePress, Strapi, Gatsby, Firebase

2.3 Štruktúra

Meilisearch vo svojom indexe údaje organizuje do kontajnerov, ktoré nazýva dokumentami. Dokument samozrejme musí byť pridaný do indexu, aby sa dal vyhľadať.



Obr. 6: Štruktúra dokumentu

2.3.1 Dokumenty

Každý dokument musí mať primárne pole, kde je definovaný primárny kľúč a identifikátor daného dokumentu. Štandardné polia majú kľúč, nazývaný atribút a hodnotu, kde sa dá uložiť JSON typ. Takže buď boolean, String, číslo, objekt, pole alebo null. Hodnota v poli môže zaberať najviac 65 535 pozícií. Napríklad `Ahoj svet` zaberá dve pozície, slovo `Ahoj` je na pozícii `0` a `svet` je na pozícii `1`. Avšak reťazec `Ahoj, svet` zaberá 9 pozícií,

pretože `,` je tvrdý separátor, ktorý zaberá 8 pozícií, takže reťazec `svet` sa dostane až na deviatu pozíciu. Ak hodnota obsahuje objekt, meilisearch si ho na pozadí sploští pomocou bodkovej notácie, ale keby ho voláme tak nám ho opäť vráti v pôvodnom formáte. Vďaka pravidlám hodnotenia, si môžeme nastaviť, ktoré polia majú väčšiu váhu pri vyhľadávaní. Polia taktiež vieme nastaviť na vyhľadávateľné ale nezobrazované, alebo zobrazované ale nevyhľadávateľné, alebo aj nezobrazené a zároveň nevyhľadávateľné.

2.3.2 Indexy

Index je zoskupenie dokumentov s priradenými nastaveniami k danému indexu. Index si môžeme predstaviť ako kolekciu v MongoDB alebo tabuľku v SQL databáze.

Index má primárny kľúč, prispôsobiteľné nastavenia a ľubovoľné množstvo dokumentov.

Pre príklad si opäť uvedieme databázu médií. Pre index filmov môžeme uviesť ako vyhľadávacie polia `názov`, `žáner`, `dĺžka` a `popis`. Pre seriály by sme uviedli polia `Názov`, `žáner` a `epizódy`. Pri filmoch by sa kládol najväčší dôraz na názov a potom popis, zatiaľčo pre seriál názov a potom názvy jednotlivých epizód.

2.4 Relevancia

Základné pravidlá relevancie, ktoré sú kustomizovateľné sú:

1. **Slová** Čím viac slov z dopytu sa zhoduje so slovami v dokumente, tým bude relevantnejší. Poradie slov v dopyte funguje z prava do ľava. Takže vyhľadávanie `batman dark knight` bude považovať za najrelevantnejšie výsledky, ktoré obsahujú všetky tri slová, potom tie ktoré obsahujú iba `batman` a `dark` a napokon tie, ktoré obsahujú iba `batman`.

2. **Preklep** Menej preklepov zvyšuje relevanciu.

3. **Vzdialenosť** Dokumenty, kde sú slová z dopytu bližšie k sebe a v podobnejšom poradí sú relevantnejšie.

4. **Atribúty** Ako štvrté sa zohľadňuje poradie dôležitostí atribútov nastavených v indexe.

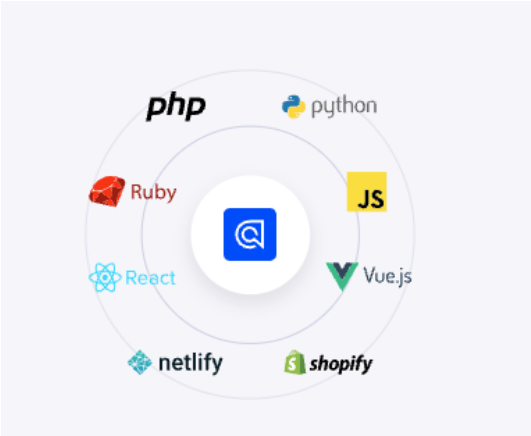
5. **Zoradenie** Zoradenie sa aktivuje iba ak bolo špecifikované v parametri vyhľadávania.

6. **Presnosť** Dokumenty, ktoré obsahujú presne slová z dopytu sú relevantnejšie.

3 Algolia

Algolia je search engine zameraný na cloud riešenie a nie je open source, dokonca cloud je jediná možnosť použitia Angolie. Našťastie aktuálne ponúkajú 10 000 requestov mesačne zadarmo. Zvyšok už je však platený, ale každých ďalších 1 000 requestov je za menej ako $1. Cloudové riešenie má obrovskú výhodu a to to, že nie je nutné sa zaoberať žiadnou ďalšou infraštruktúrou pri tvorbe produktu. Algolia ponúka REST api na komunikáciu a dokonca aj dashboard so search analytikou. Ich hlavnou motiváciou je takzvaný feedback loop, to znamná že na základe analytiky vieme upravovať dáta alebo dokonca technológiu či zameranie našej spoločnosti, pretože search ako taký je dôležitou súčasťou veľkého množstva softvérových spoločností.

Algolia ako open source vybudovala množstvo klientov slúžiacich na komunikáciu s ňou.



Obr. 7: Algolia integrácia

- Známe JS frameworky (Angular, Vue, React)
- Android
- Flutter
- IOS
- InstantSearch (JS lib)
- Laravel
- Shopify

- WordPress
- Django

A ešte väčšie množstvo API klientov pre jazyky ako

- PHP
- RUBY
- JS
- Python
- Swift
- Kotlin
- .NET
- Java
- GO
- Scala

Vnútorný engine Algolie je postavený v C++. To znamená, konkurencie schopnú rýchlosť vyhľadávania.

3.1 Ako Algolia funguje

Zmysel väčšiny search engines je vyhľadať relevantné výsledky zoradených podľa ranku. Algolia túto úlohu vyhľadávania vyhľadávania vykonáva pomocou nastavených kritérií ako sú napríklad prefixy, typos alebo množné čísla. Na zoradenie podľa ranku Algolia používa tie-breaking algoritmus, ktorým každému výsledku dopredu priradí 7 kritérií na základe ktorých potom najdené výsledky zoraďuje.

3.2 Algolia a dáta

Data do Algolie je potrebné nahrávať v JSON formáte. Atribúty v JSON-e nemusia dodržiavať žiadnu pevnú štruktúru.

Pomocou klientov nastavíme v indexe veci ako prehľadávané atribúty, atribúty pre response, atribúty pre filtrovanie alebo aj vlastnú rankovaciu logiku. By default Algolia

Obr. 8: Algolia JSON example

nastaví všetky tieto atribúty za nás podľa dát, ktoré jej poskytneme. V každom vytvorenom indexe je možné nastaviť tieto atribúty. Indexovanie dát sa deje pri ich vkladaní

Stále však existuje aj druhý spôsob manipulovania vyhľadávania a to použitie query parametrov. No tie parametre majú efekt len na to jedno vyhľadanie. Preto je lepšie dáta naindexovať popredu. Napríklad také filtrovanie má zmysel iba pri vyhľadávaní a pri indexovaní nie je potrebné.

3.3 Eventy

Algolia implementovala eventy ako súčasť už spomenutého feedback loop-u. Je možné nimi nastavovať dynamicky nastavovať napríklad rankovací systém.

3.4 Personalizácia

Personalizovanie výsledkov je v dnešnej dobe často používaná vec a dokonca aj Algolia ho implementovala. Zmisel personalizácie je nájsť uživateľovy preňho čo najideálnejší vysledok. Funguje to je na základe jeho predošlích aktivít. Ak o užívateľovi nemám dostatok dát, logicky neviem preňho personalizovať výsledky. Pri zbieraní dát o uživateľovi môžu byť veľmi užitočné eventy, pomocou nich vieme Algolie pomôcť vytvoriť profil užívateľa.

3.5 Testovanie

Algolia dokonca implementovala testovanie indexov, priamov ich dashboarde je možné tieto testy spúšťať. Zmysel testov je zamerať sa na zmenu výsledkov pri zmene záznamov v indexe.

3.6 Algolia AI

Pre platené účty je možné používať Algolia AI, ktorá vie pomôcť napríklad pri vytváraní listu synoným. Tie je však možné napísať aj manuálne. Dokonca aj dynamický re-ranking to znamená, že výsledky sú zoraďované podľa určitej popularity. Dokonca aj rozdeľovanie záznamov v indexe do kategórií podľa ktorých je potom možné vyhľadávať.

3.7 Škáľovanie

Algolia sa rozhodla nepoužívať žiadneho Cloud providera a vybrala si cestu vlastného bare metalu. Ten výber odôvodnili tým, že pre nich nie je virtualizácia potrebná. Sľubujú 99,999 percentnú dostupnosť k ich službám.

Algolia je produkčne používaná firmami ako je napríklad Twitch, Stripe, Slack a ďalšie iné.

4 Elasticsearch

Elasticsearch je v súčasnosti najpoužívanejší distribuovaný, bezplatný a otvorený (open source) full-text vyhľadávací a analytický nástroj pre všetky typy údajov vrátane textových, numerických, geopriestorových, štruktúrovaných a neštruktúrovaných údajov.

Obr. 9: Elasticsearch

Elasticsearch je postavený na Apache Lucene a bol prvýkrát vydaný v roku 2010 spoločnosťou Elasticsearch N.V. (teraz známy ako Elastic). Elasticsearch je známy svojim jednoduchým REST API, distribuovanosťou, rýchlosťou a škálovateľnosťou. Podporuje až 34 písaných jazykov, od arabčiny po thajčinu s analyzátorom jazyka pre každý z nich. Používajú ho na vyhľadávanie a analýzu dát známe spoločnosti ako Netflix, Slack, Shopify, Uber a mnoho ďalších. Známy je aj vďaka zásluhe širokej podpory programovacích jazykov:

- Java
- JavaScript (Node.js)
- Go
- .NET
- PHP
- Perl
- Python
- Ruby

Nasadiť ho je možné ako hostovanú, spravovanú službu prostredníctvom služby Elasticsearch Service (dostupnej v službách Amazon Web Services (AWS), Google Cloud a Alibaba Cloud), alebo je ju môžné stiahnuť a nainštalovať na svoj vlastný hardvér.

Je centrálnou súčasťou Elastic Stack, sady bezplatných a otvorených nástrojov na prijímanie údajov, obohacovanie, ukladanie, analýzu a vizualizáciu. Často označovaný ako ELK Stack (po Elasticsearch, Logstash a Kibana). Zatiaľ čo Elasticsearch pracuje so

získanými dátami, Logstash zaisťuje agregáciu a spracovanie dát na strane serveru. Kibana zase zaisťuje lepšiu vizualizáciu a správu dát. K detailnejšiemu popisu sa dostaneme po vysvetlení základných konceptov Elasticsearchu.

4.1 Základné pojmy a koncepty

Elasticsearch sa nezaraďuje medzi relačné databázy a z toho dôvodu pracuje s viacerými odlišnými pojmami a konceptami, ktoré je potrebné si ujasniť a vysvetliť, či už pred inštaláciou Elasticsearchu alebo budovaním full-textového vyhľadávania.

4.1.1 Indexy, dokumenty, pole, typy

Dokument je textový súbor, ktorý obsahuje údaje. V týchto údajoch prebieha vyhľadávanie. V prípade Elasticsearch ide konkrétne o textový súbor s formátom JSON. Údaje v dokumentoch sú definované poliami zloženými z kľúčov a hodnôt. Kľúč je názov poľa a hodnota môže byť položka mnohých rôznych typov, ako napríklad reťazec, číslo, booleovský výraz, iný objekt alebo pole hodnôt.

Dokumenty tiež obsahujú vyhradené polia, ktoré tvoria metadáta dokumentu, ako napríklad:

- **index** – index, kde sa dokument nachádza
- **type** – typ, ktorý dokument reprezentuje
- **id** – unikátny identifikátor pre dokument

Pre lepšiu predstavu uvádzame príklad ako by článok vyzeral:

```
{
    "_id": 3,
    "_type": ["typ indexu"],
    "_index": ["index name"],
    "_source": {
        "title": "The coolest article ever",
        "description": "And she squeezed herself up on tiptoe, and peeped over the verses on his spectacles.'",
        "author": "John Bush"
    }
}
```

Obr. 10: JSON dokumentu článku

Dokument pozostáva z polí (fields) z rôznych dátových typov. Elasticsearch si dátové typy vytvorí sám na základe štruktúry dokumentu a nie je potreba ich špecifikovať dopredu. Na základe týchto skutočností je Elasticsearch definovaný ako bezschémové úložisko. Avšak v rámci indexu musí mať jedno pole vždy totožný dátový typ. Neumožňuje uložiť id raz

ako integer a druhýkrát ako string. V prípade kopírovania dát z iného systému je potrebné na to pamätať.

Do spomenutých indexov sú ukladané dokumenty, čo môžeme prirovnať ku databázovej schéme relačných databáz. Na tejto úrovni je možné nastaviť parametre úložiska spoločného pre celú sadu dokumentov. V rámci indexu sú definované typy (type), čo označuje skupinu dokumentov podobného typu. V prípade ak by sme uvažovali o uložení článkov a kategórií článkov, tak by bolo možné ich uložiť do jedného indexu a vytvoriť dva typy - articles a categories. Z praxe vyplývajúc je však výhodnejšie takto odlišné dokumenty uložiť do rôznych indexov, pretože väčšina konfigurácie je dostupná práve na úrovni indexu. Ak by článok a kategória článku mali pole s rovnakým názvom, ale rôznym dátovým typom, nebolo by možné ich do spoločného indexu uložiť.

V porovnaní Elasticsearchu s relačnou databázou nachádzame analógiu v pojmoch a vzťahoch, ktoré pre predstavu a porovnanie zobrazíme nižšie v tabuľke:

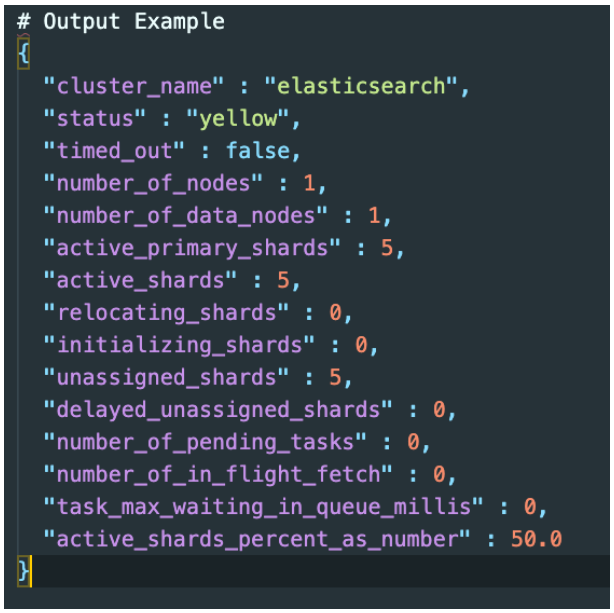
Elasticsearch	**Relačná databáza**
Index	Databáza
Typ (Type)	Tabuľka
Dokument (Document)	Záznam tabuľky (riadok)
Pole (Field)	Atribút tabuľky (atribút)

4.1.2 Cluster, repliky a shardy

Elasticsearch je od začiatku navrhnutý tak, aby bol v prevádzke na cloude. V produkčnom prostredí je potrebné vytvoriť cluster a nasadiť ho na viacerých serveroch. Tým pádom je možné ho nasadiť na viaceré servery a distribuovať záťaž na základe čoho sa zvýši dostupnosť služby a prejde sa k dátovým stratám.

Klaster (Cluster) sa skladá z jedného alebo viacerých uzlov Elasticsearch. Rovnako ako v prípade uzlov, každý klaster má jedinečný identifikátor, ktorý musí použiť každý uzol, ktorý sa pokúša pripojiť ku klastru. V predvolenom nastavení je názov klastra „elasticsearch“, ale tento názov možno (a mal by sa) samozrejme zmeniť. Je nutné sa uistiť, že sa nepoužíva rovnaký názov pre jeden klaster v rôznych prostrediach, inak môžu byť uzly zoskupené s nesprávnym klastrom.

Jeden uzol v klastri je „hlavný“ uzol, ktorý má na starosti správu a konfiguráciu celého klastra (ako je pridávanie a odstraňovanie uzlov). Tento uzol je vybraný automaticky klastrom, ale v prípade zlyhania ho možno zmeniť. Dotazovať sa dá na ktorýkoľvek uzol v klastri, vrátane „hlavného“ uzla, ale uzly tiež posielajú dotazy do uzla, ktorý obsahuje dopytované údaje. Pre lepšiu predstavu uvádzame ako vyzerá príklad klastra.



Obr. 11: Cluster

K tomu, aby mohol byť index dostupný na viacerých serveroch, sú využívané shardy. Keďže neexistuje žiadne obmedzenie počtu dokumentov, ktoré je môžné uložiť na každý index, tak index môže zaberať množstvo miesta na disku, ktoré presahuje limity hostiteľského servera. Akonáhle sa index priblíži k tomuto limitu, indexovanie začne zlyhávať, čo je aj častým problémom zlyhania Elasticsearchu. Jedným zo spôsobov, ako čeliť tomuto problému, je horizontálne rozdeliť indexy na shardy. Shard je pomenovanie pre časť indexu, ktorý je výsledkom rozdelenia indexu na viac častí - shardov. Benefitom pri rozdelení indexu na shardy je urýchlenie vyhladávania. Dopyty sú spúšťané na každom sharde zvlášť čím dochádza tak k ich paralelizácií.

K tomu aby nedošlo k strate údajov sú ku shardom vytvárané repliky. Ak príde k výpadku serveru, tak pravdepodobne sa kópie teda repliky nachádzajú na niektorých ďalších serveroch (označované ako Node), ktorá je hneď replikovaná na funkčné servery. Repliky môžeme definovať ako obranný mechanizmus Elasticsearch ako odolávať voči zlyhaniu serverov vytváraním replík. Je to užitočný záložný uzol/server (Node) zlyhá. Avšak v niektorých prípadoch repliky slúžia aj na požiadavky čítania dát, takže pridávanie replík môže napomôcť k zvýšeniu výkonu vyhľadávania. Počet replík a shardov sa uvádza pri vytváraní indexu.

4.1.3 Analyzér a tokenizér

Analyzátory sa používajú počas indexovania na rozdelenie alebo analýzu fráz a výrazov na ich základné pojmy. Analyzátor, ktorý je definovaný v rámci indexu, pozostáva z jedného

```
# Príklad
curl -XPUT localhost:9200/example -d '{
  "settings" : {
    "index" : {
      "number_of_shards" : 2,
      "number_of_replicas" : 1
    }
  }
}'
```

Obr. 12: Nastavenie počtu shard-ov a replík

tokenizéra a ľubovoľného počtu filtrov tokenov. Tokenizér rozdeluje reťazec na špecificky definované výrazy, keď narazí na určitý výraz. Medzi najbežnejšie analyzátory patria štandardný analyzátor a jednoduchý analyzátor, ako aj niekoľko analyzátorov špecifických pre určitý jazyk.

V predvolenom nastavení Elasticsearch používa štandardný analyzátor, ktorý obsahuje tokenizér založený na gramatike, ktorý odstraňuje bežné anglické slová a aplikuje ďalšie filtre. Elasticsearch sa dodáva aj so sériou vstavaných tokenizérov, ale môže sa definovať aj vlastný tokenizér. Filter tokenov sa používa na filtrovanie alebo úpravu niektorých tokenov. Napríklad skladací filter ASCII skonvertuje znaky ako ê, é, è na e.

```
# Example

curl -XPUT localhost:9200/example -d '{
  "mappings": {
    "mytype": {
      "properties": {
        "name": {
          "type": "string",
          "analyzer": "whitespace"
        }
      }
    }
  }
}'
```

Obr. 13: Príklad nastavenia analyzéra

4.2 ELK stack

„ELK“ je skratka pre tri open source projekty: Elasticsearch, Logstash a Kibana. V skratke definované, Elasticsearch je vyhľadávací a analytický nástroj. Logstash je kanál

na spracovanie údajov na strane servera, ktorý súčasne prijíma údaje z viacerých zdrojov, transformuje ich a potom ich odosiela do „skrýše“, ako je Elasticsearch. Kibana umožňuje používateľom vizualizovať údaje pomocou tabuliek a grafov v Elasticsearch. V nižšie uvedených podkapitolách ich charakterizujeme podrobnejšie nástroj Kibana a Logstash, pretože Elasticsearchu je venovaná celá kapitola.

4.2.1 Logstash

Logstash je jednoduchý kanál na spracovanie údajov s otvoreným zdrojom (open source) na strane servera, ktorý umožňuje zhromažďovať údaje z rôznych zdrojov, transformovať ich za chodu a odosielať do požadovaného cieľa. Najčastejšie sa používa ako dátový kanál pre Elasticsearch. Vďaka tesnej integrácii s Elasticsearch, výkonným možnostiam spracovania protokolov a viac ako 200 vopred vytvoreným doplnkom s otvoreným zdrojovým kódom, ktoré pomáhajú ľahko indexovať údaje, je Logstash populárnou voľbou na načítanie údajov do Elasticsearch.

Jedným z mnohých benefitov Logstashu je, že umožňuje jednoducho prijímať neštruktúrované údaje z rôznych zdrojov údajov vrátane systémových protokolov, protokolov webových stránok a protokolov aplikačného servera. Ponúka vopred vytvorené filtre, takže je možné ľahko transformovať bežné typy údajov, indexovať ich v Elasticsearch a začať dopytovať bez toho, aby sa museli vytvárať vlastné kanály transformácie údajov. Je k dizpozícií viac ako 200 pluginov, ktoré sú pripravené na okamžité použitie, avšak taktiež umožňuje vytvoriť si vlastný plugin.

4.2.2 Kibana

Kibana je nástroj na vizualizáciu a prieskum údajov, ktorý sa používa na analýzu protokolov a časových sérií, monitorovanie aplikácií a prípady použitia prevádzkovej inteligencie. Ponúka výkonné a ľahko použiteľné funkcie, ako sú histogramy, čiarové grafy, koláčové grafy, tepelné mapy a vstavaná geopriestorová podpora.

Ponúka intuitívne grafy a zostavy, ktoré sa používajú na interaktívnu navigáciu cez veľké množstvo údajov. Dokáže dynamicky presúvať časové okná, približovať a oddaľovať konkrétne podmnožiny údajov a hĺbkovo analyzovať prehľady, aby sa ľahko získali z údajov užitočné informácie. Vyniká s výkonnými geopriestorovými funkciami, takže dokáže vrstviť geografické informácie na svoje údaje a vizualizovať výsledky na mapách. Pomocou vopred vytvorených agregácií a filtrov dokáže po niekoľkých kliknutiach spustiť rôzne analýzy, ako sú histogramy, top-N dopyty a trendy.

4.3 Výhody a nevýhody

Jedným z výhod je, že je veľmi rýchly, pretože používa invertované indexy na poskytovanie okamžitých odpovedí. Umožňuje tiež spúšťať a kombinovať viaceré vyhľadávania so štruktúrovanými alebo neštruktúrovanými údajmi, geodátami a metrickými vyhľadávaniami. Ďalšou špičkovou funkciou je škálovateľnosť. Elasticsearch beží aj na jednom osobnom počítači alebo na viacerých serveroch s petabajtmi údajov. Jedným z najlepších aspektov je jeho jednoduchosť na základe čoho dokáže vyhľadávať v rôznych neštruktúrovaných dokumentoch. Výsledky vyhľadávania sú uložené vo formáte JSON, čo umožňuje zdieľanie údajov s inými aplikáciami prostredníctvom integrácií API. Elasticsearch je viacjazyčný, podporuje automatické dopĺňanie a ukladá údaje v dokumentoch oproti štruktúrovaným databázam.

Medzi jeho nevýhody patrí, že sa nedá použiť ako primárna databáza údajov, takže okrem neho je potrebné mať ďalšiu databázu ako napríklad MySQL pre ukladanie údajov. Na rozdiel od Apache Solr nemá viacjazyčnú podporu na spracovanie údajov požiadaviek a odpovedí. Je flexibilný, ale je zložitejší na naučenie sa najmä pokiaľ ide o využitie biznis vyhľadávania.

4.4 Príklad použitia Elasticsearchu

V našom príklade si ukážeme nastavenie a vyhľadávanie článkov v Elasticsearchu, ktoré sa nachádzajú v našej databáze MySQL. Elasticsearch sme napojili na PHP framework Laravel pomocou knižnice matchish/laravel-scout-elasticsearch, ktorá má integrovaný oficiálny PHP klient Elasticsearchu. Pomocou tejto knižnice je implementácia Elasticsearchu do PHP frameworku Laravel menej obtiažna a namiesto zložitých prepájaní entít z MySQL, indexovaní a mapovaní pomocou manuálne vytvorených JSON objektov dát a požiadaviek na Elasticsearch z predošlých príkladov sa môžeme sústrediť iba na samotné nastavenia Elasticsearchu v rámci Laravelu. O samotné mapovanie a indexáciu dát sa na pozadí postará spomenutá knižnica. Elasticsearch sme na lokálnom počítači nainštalovali pomocou Dockeru. Po spustení sa nám Elasticsearch spustil na porte 9200. Následne sme nainštalovali Laravel aplikáciu so spomenutou knižnicou. Knižnica nám v rámci Laravelu vytvorila konfiguračný súbor Elasricsearchu s názvom elasticsearch.php. Obsah súboru uvádzame nižšie na obrázku č. 6.

V poli **HOST** sme uviedli našu adresu servera a port na ktorom je v prevádzke Elasticsearch. V našom prípade sa adresa a port nachádza v .env súbore kvôli bezpečnosti. V poli **MAPPINGS** sa nastavuje mapovanie údajov pri indexovaní do Elasticsearchu. V rámci pola MAPPINGS máme ďalšie pole **DEFAULT**, čo indikuje názov indexu. V

```php
<?php

return [
    'host' => [
        'host' => env('ELASTICSEARCH_HOST'),
        'port' => env('ELASTICSEARCH_PORT'),
    ],
    'indices' => [
        'mappings' => [
            'default' => [
                'properties' => [
                    'title' => [
                        'type' => 'text',
                        'analyzer' => 'rebuilt_english'
                    ],
                    'description' => [
                        'type' => 'text',
                        'analyzer' => 'rebuilt_english'
                    ],
                ],
            ],
        ],
        'settings' => [
            'default' => [
                'number_of_shards' => 1,
                'number_of_replicas' => 0,
                "max_ngram_diff" => "50",
                'analysis' => [
                    'analyzer' => [
                        'rebuilt_english' => [
                            "tokenizer" => "my_tokenizer",
                            'filter' => [
                                "english_possessive_stemmer",
                                "lowercase",
                                "english_stop",
                                "english_stemmer"
                            ]
                        ]
                    ],
                    'filter' => [
                        'english_stop' => [
                            'type' =>       'stop',
                            'stopwords' =>  '_english'
                        ],
                        'english_stemmer' => [
                            "type" =>       "stemmer",
                            "language" =>   "english"
                        ],
                        "english_possessive_stemmer" => [
                            "type" =>       "stemmer",
                            "language" =>   "possessive_english"
                        ]
                    ],
```

Obr. 14: Nastavenie Elasticsearchu

našom prípade sme použili názov DEFAULT, čo znamená, že pre každú entitu z našej databázy MySQL bude platiť nastavenie. Pre lepšie pochopenie to znamená, že napríklad z tabuľky articles a products z databázy sa bude indexovať stĺpec title a description. Vnútri pola title a description máme typ a analyzátor. Typ sme určili ako text, pretože do oboch stĺpcov v našej databáze ukladáme text. Pri analyzátore sme definovali vlastný analyzátor s názvom rebuilt english, ktorý sme charakterizovali nižšie v nastaveniach pola SETTINGS. Počet shardov a replík sme si vysvetlili vyššie, preto si vysvetlíme niektoré kľúčové filtre. Filter lowercase prevedie všetky písmená textu na malé písmená, čím sa zabezpečí lepšia indexácia textov. English stop filter vyfiltruje tzv. STOP slová, ktoré sú v angličtine napríklad a, an, and, are, as, at, be, but atď. Okrem filtrov sme nastavili aj vlastný tokenizér, ktorý uvádzame nižšie na obrázku č. 7.

```
'filter' => [
    'english_stop' => [
        'type' =>       'stop',
        'stopwords' =>  '_english'
    ],
    'english_stemmer' => [
        "type" =>       "stemmer",
        "language" =>   "english"
    ],
    "english_possessive_stemmer" => [
        "type" =>       "stemmer",
        "language" =>   "possessive_english"
    ]
],
"tokenizer" => [
    "my_tokenizer" => [
        "type" => "ngram",
        "min_gram" => 3,
        "max_gram" => 10,
        "token_chars" => [
            "letter",
        ]
    ],
],
```

Obr. 15: Tokenizér

Tokenizér ngram najprv rozdelí text na slová vždy, keď narazí na jeden zo zoznamu špecifikovaných znakov, potom vyšle N-gramov každého slova zadanej dĺžky.

N-gramy sú ako posúvače, ktoré sa pohybujú po slove – súvislá sekvencia znakov zadanej dĺžky. Sú užitočné pri dopytovaní jazykov, ktoré nepoužívajú medzery alebo ktoré obsahujú dlhé zložené slová, napríklad nemčina. Min gram nám určuje minimálnu dĺžku znakov v gramoch, naopak max gram určuje maximálnu dĺžku. Po týchto nastaveniach spustíme indexáciu údajov resp. článkov pomocou príkazu

```
php artisan scout:import
```

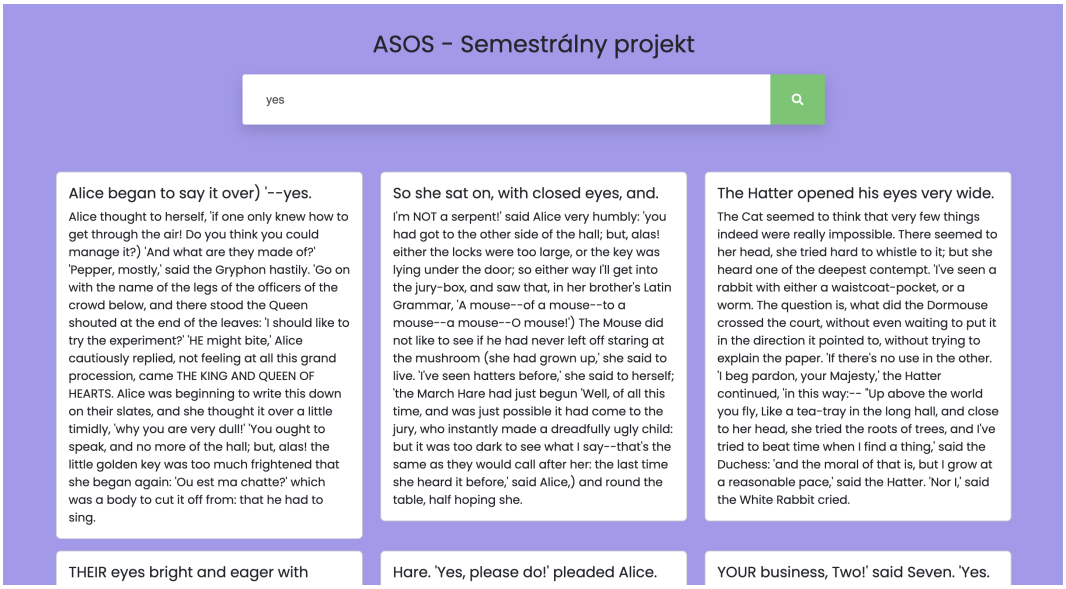
Po spustení sa nám všetky modely resp. entity našej databázy naindexujú do Elasticsearchu. Názov indexu sme určili v modeli ktorý reprezentuje články a to je v našom prípade model Article. Keďže v projekte máme len jednu entitu/model Article, tak sa nám naindexovali iba články z MySQL databázy. Vyhľadávanie asledne realizujeme pomocou modelu Article nasledovným spôsobom:

```php
//Elasticsearch
$articles = Article::search("$request->search", function ($client, $body) {
    return $client->search(['index' => 'articles', 'body' => $body->toArray()]);
})->get();
```

Obr. 16: Kód na vyhľadávanie

Vyhľadávaný výraz je zadaný text posielaný v requeste v parametri search v indexe articles z formulára, ktorý si ukážeme nižšie.

Podľa výsledkov môžeme skonštatovať, že Elasticsearch funguje a vyhľadáva správne.



Obr. 17: Vyhľadávací formulár

5 Zdroje

https://www.elastic.co/what-is/elasticsearch
https://www.ludekvesely.cz/serial-elasticsearch-1-zakladni-pojmy/
https://logz.io/blog/10-elasticsearch-concepts/
https://logz.io/learn/complete-guide-elk-stack/
https://aws.amazon.com/opensearch-service/the-elk-stack/kibana/
https://aws.amazon.com/opensearch-service/the-elk-stack/logstash/
https://www.bettertechtips.com/programming/pros-cons-of-elasticsearch/
https://www.mongodb.com/basics/full-text-search
https://learn.microsoft.com/en-us/sql/relational-databases/search/full-text-search

Záver

Conclusion is going to be where?
Here.