МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА МОДУЛЮ ПОШУКУ З ВИКОРИСТАННЯМ ELASTICSEARCH»**

Виконав: студент 2 курсу, групи 8.1218

спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

освітньої програми інженерія програмного забезпечення
(назва освітньої програми)

А.О. Єременко
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення

(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

Лісняк А.О.

(підпис)

« 29 » 05 2019 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Єременко Артему Олександровичу

(прізвище, ім'я та по батькові)

1. Тема роботи Розробка модулю пошуку з використанням Elasticsearch

керівник роботи Мухін Віталій Вікторович, к.т.н., доцент

(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 29 » травня 2019 року № 811-с

2. Строк подання студентом роботи 29.12.2019

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Програмна реалізація
4. Тестування програмного продукту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 31.05.2019

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	31.05.2019	
2.	Збір вихідних даних.	09.09.2019	
3.	Обробка методичних та теоретичних	12.10.2019	
	джерел.		
4.	Розробка першого та другого розділу.	27.10.2019	
5.	Розробка третього розділу.	12.11.2018	
6.	Оформлення та нормоконтроль	27.12.2019	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	15.01.2020	

Студент _____ А.О. Єременко
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка модулю пошуку з використанням Elasticsearch»: 49 с., 11 рис., 13 джерел, 3 додатки.

ІНДЕКС, ПОШУКОВА СИСТЕМА, ФОРМАТ JSON, ШАРД, APACHE LUCENE, ELASTICSEARCH, CRUD ОПЕРАЦІЇ З ДОКУМЕНТАМИ, PHP, REST API

Об’єкт дослідження – процес пошуку інформації у Інтернет-магазині за заданим критерієм.

Предмет дослідження – розробка потужного інструментарію здійснення пошуку на сайті.

Мета роботи: програмна реалізація модулю пошуку інформації у Інтернет-магазині з використанням пошукової системи Elasticsearch засобами мови програмування PHP.

Методи дослідження – порівняння, аналіз.

Магістерська кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі проведено огляд сучасних систем пошуку даних на сайті, виявлено найпотужніші з них та досліджено переваги бібліотеки Elasticsearch. У другому розділі розглянуто архітектуру пошукової системи Elasticsearch, способи досягнення релевантності запитів та спроектовано схему взаємодії з цією системою. Третій розділ присвячено програмній реалізації та графічному інтерфейсу користувача.

SUMMARY

Master’s Qualification Thesis «Developing a Search Module using Elasticsearch»: 49 pages, 11 figures 13 references, 3 supplements.

INDEX, SEARCH SYSTEM, JSON, SHARD, APACHE LUCENE, ELASTICSEARCH, CRUD DOCUMENTS, PHP, REST API

The object of research is the process of finding information in an online store by a given criterion.

The subject of research is the development of a powerful tool for performing search on the site.

The aim of research is software implementation of the information retrieval module in the online store using Elasticsearch search engine using PHP programming language.

The methods of research are comparison, analysis.

Qualification thesis include an introduction, three sections, conclusions, references.

The first section provides an overview of current data search systems on the site, identifies the most powerful of them, and explores the benefits of the Elasticsearch library. The second section looks the architecture of the Elasticsearch search engine, how to achieve query relevance, and outlines how it interacts with that system. The third section is dedicated to software implementation and graphical user interface.

ЗМІСТ

ВСТУП

У теперішній час стрімкого розвитку інформаційних технологій та великого обсягу інформації перед користувачем мережі Інтернет постає задача швидкого пошуку необхідної інформації. Точність знайдених даних залежить від правильності завдання критеріїв пошуку.

Головним завданням розробників веб-сайтів є розробка зручного та інтуїтивно зрозумілого інструментарію пошуку інформації на сайті. Швидкість та точність результатів пошуку інформації користувачем сайту значно впливає на кількість його відвідувачів. В більшості випадків користувач, який вперше відвідує сайт в пошуках важливої для себе інформації, не буде розбиратись у навігаційних панелях, випадаючих меню та інших елементів навігації, а в поспіху намагатиметься знайти щось схоже на пошуковий рядок. І якщо такого інструментарію бракуватиме або він не впорається з пошуковим запитом, то відвідувач закриє вкладку із сайтом.

Таким чином, зручний інструментарій знаходження даних з наявністю автозаповнювача, фільтрів на основі геолокацій та багаторівневої агрегації дає змогу збільшити кількість клієнтів власника сайту та в подальшому збільшити обсяг продажу товару або послуг і, відповідно, збільшити прибутки. А розробка зручного інструментарію пошуку інформації на сайті є актуальною задачею сьогодення.

До сучасних систем пошуку висуваються певні вимоги, такі як:

- наявність пошуку із врахуванням мовної морфології;
- можливість вказати контекст пошуку;
- індексування змісту сайту;
- механізм ранжирування;
- висока релевантність пошуку;
- можливість пошуку серед даних великого обсягу.

В рамках кваліфікаційної роботи було сформульовано наступну мету: програмно реалізувати модуль пошуку інформації у Інтернет-магазині з використанням пошукової системи Elasticsearch засобами мови програмування PHP.

Для досягнення поставленої мети необхідно виконати наступні задачі:

– провести огляд існуючого інструментарію пошуку інформації на веб-сайті;

– вивчити основні поняття та термінологію пошукової системи Elasticsearch;

– розглянути методи пошуку у системі Elasticsearch;

– розробити алгоритм взаємодії з системою Elasticsearch;

– реалізувати програмний модуль пошуку по сайту з використанням Elasticsearch засобами PHP.

Об'єктом даної кваліфікаційної роботи є процес пошуку інформації у Інтернет-магазині за заданим критерієм.

Предметом роботи є розробка потужного інструментарію здійснення пошуку на сайті.

Структура і обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел. Загальний обсяг магістерської роботи – 49 сторінок, 11 рисунків. Список використаних джерел нараховує 13 найменувань.

1 ОГЛЯД СИСТЕМ ПОШУКУ ІНФОРМАЦІЇ В ІНТЕРНЕТ МАГАЗИНАХ

1.1 Пошукові системи інформації в інтернет магазині

Розглянемо найпопулярніші системи пошуку інформації на сайті. Дослідимо їх на відповідність вимогам, що висуваються до програмного забезпечення такого типу.

Kea Labs [1] – це платформа інструментів для Інтернет-магазинів, які допомагають здійснювати релевантний пошук товарів на сайті.

Пошук за системою Kea Labs розпізнає:

– опечатки в запитах (пдюшевий, ланпа, кровадь);

– синоніми (складаний, що складається, портативний);

– здійснює переклад і транслітерацію (сумісного, samsung, cfvceyu).

Також Kea Labs аналізує фотографії товарів і розпізнає кольори і відтінки.

З моменту установки пошук починає збирати інформацію про поведінку користувачів та формувати статистику, яка дозволяє виявити популярні запити, зміни попиту і надати інформацію те, що відвідувачі шукали, але не знайшли.

Multisearch [2] – пошук з машинним навчанням від українських розробників. Він розуміє синоніми, запити простою людською мовою, помилки і помилки з урахуванням контексту, розкладку клавіатури, пропонує підказки запитів. Вибирає релевантні варіанти.

Multisearch розуміє синоніми, розпізнає помилки, помилки розкладки клавіатури, враховує морфологію і семантику, визначає транслітерації в запиті, знаходить об'єкт пошуку по різним морфологічним формам, визначає параметр за синонімом, наприклад:

– сигналка -> автосигналізація;

– зарядка дефендер -> автомобільне ЗУ Defender;

– клава асус -> клавіатура Asus;

– і т.п.

AnyQuery [3] – інструмент пошуку по сайту з наступними функціями і можливостями:

– автоматичне поповнення словника синонімами (індивідуально для кожного каталогу товарів);

– виправлення помилок;

– підказки при введенні перших літер;

– виправлення мови.

Це інструмент, який навчає пошук розуміти мову покупців: виправляє помилки. Наприклад, якщо покупець пише Smsung замість Samsung, пошук його розуміє і видає потрібні моделі телефонів; розпізнає синоніми. Пошук навчається розуміти, що покупець мав на увазі, розшифровує запити з помилками і рекомендує супутні товари. Для цього використовуються технології машинного навчання.

Textocat E-commerce [6] – інструментарій пошуку з наступними можливостями:

– висока релевантність пошукових результатів;

– наявність інтерфейсу для роботи аналітиків;

– пошук на російській та англійській мовах.

Висока релевантність результатів пошуку здійснюється на основі багатофакторної моделі машинного навчання і власної платформи обробки природної мови Textocat API.

Модуль Workbench дозволяє оцінити ефективність змін в пошуку через A / B тестування.

Здійснюється вибір стратегій ранжирування для груп товарів, проводиться аналіз низькоефективних пошукових запитів, існує можливість редагування пошукових тезаурусів та виправлення помилок в запитах, наявна рекомендація пошукових запитів при наборі.

Detectum [7] – система пошуку на природній мові за табличними параметрами. Особливість пошукача Detectum – це облік структури каталогу інтернет-магазину і параметрів товарів з бази даних.

Система виконує наступні функції:

– визначає контекст для користувача запитів;

– веде облік параметрів, помилок і неточних значень.

Sphinx [8]– потужний і швидкий відкритий інструментарій пошуку, який має пряму інтеграцію з популярними базами даних і підтримує такі можливості пошуку, як ранжування і стемінг для російської та англійської мов. Також підтримується і нетривіальна можливість розподіленого пошуку та кластеризації, і має високу швидкість індексації та пошуку.

Sphinx може працювати з великими обсягами даних. Має API для різних мов, в першу чергу, для PHP. Але сам пошуковик необхідно компілювати і встановлювати окремо, тому для звичайного хостингу він непридатний – необхідний власний сервер, з великою кількістю пам'яті.

Nutch [10] – це пошуковий проект на базі бібліотеки Lucene. Це веб-пошуковий інструментарій суміщений з розподіленою системою зберігання даних Hadoop. Може працювати з віддаленими вузлами в мережі, індексує не тільки HTML, але і MS Word, PDF, RSS, PowerPoint і навіть MP3 файли.

Xapian [11] – пошукова система з наявністю «живого» індексу, що не потребує перебудови при додаванні документів, з дуже потужною мовою запитів, включаючи вбудований стемінг, перевірку орфографії, і навіть підтримку синонімів. У пакет входить Omega – надбудова над бібліотекою, яка використовується в якості самостійного пошукача і відповідає за можливості індексації різних типів документів і CGI інтерфейс.

1.2 Пошукові системи сімейства Apache Lucene

Lucene – вільна бібліотека для високопродуктивного повнотекстового пошуку фонду Apache, орієнтована саме на вбудову в інші програми.

Solr [9]– пошукач на базі бібліотеки Lucene, який значно розширює її можливості.

Це самостійний сервер корпоративного рівня, що приймає документи по протоколу HTTP в форматі XML і повертає результат також через HTTP (XML, JSON або інший формат). В ньому повністю підтримується кластеризація і реплікація на кілька серверів, є підтримка додаткових полів в документах, фасетного пошуку і фільтрації, розвинені засоби адміністрування, а також можливості кешування і бекапу індексу в процесі роботи.

Elasticsearch [4] – це потужна система пошуку на базі бібліотеки пошуку Lucene [5], що дозволяє організувати ефективний пошук в базі даних.

Основним призначенням Elasticsearch є складний повнотекстовий пошук в базі з урахуванням морфології мови, контролем помилок і ранжування результатів за релевантністю.

Elasticsearch – це noSQL база даних з керуванням через HTTP запити і щільною інтеграцією з системою візуалізації даних (Kibana), системою збору подій і логів (Logstash) і різними інструментами аналізу даних.

Elasticsearch має аналітичну складову, що має широке застосування і допомагає вирішити велику кількість нетривіальних бізнес-завдань.

Elasticsearch використовує документо-орієнтовану модель зберігання даних, зберігаючи все в JSON-форматі. Розподіл і зберігання інформації засновано на, так званих індексах і типах. Їх може бути безліч.

1.3 Висновки за розділом 1

Отже, розглянувши ряд пошукових систем, існуючих на теперішній час, можна виділити декілька з них. Це системи Solr та Elasticsearch, які працюють на базі бібліотеки Lucene та здійснюють швидкий, масштабований запит з високою релевантністю. Окрім цього, вони мають увесь арсенал загальних функцій сучасних пошукачів.

В рамках даного дипломного проекту для реалізації модулю пошуку було обрано систему Elasticsearch, яка здійснює швидкий релевантний пошук та працює з великими обсягом даних.

Система Elasticsearch має низку переваг, серед них можна виділити наступні:

– масштабованість і відмовостійкість. Elasticsearch легко масштабується. До вже існуючої системи можна на ходу додавати нові сервери, і пошуковий інструментарій зможе сам розподілити на них навантаження. При цьому дані будуть розподілені таким чином, що при відмові одного з них дані не будуть загублені і сама пошукова система продовжить роботу без збоїв;

– мультиарендність (англ. Multitenancy) – можливість організувати кілька різних пошукових систем в рамках одного об'єкта Elasticsearch. Причому організувати їх можна динамічно;

– операційна стабільність – на кожну зміну даних в сховищі ведеться логування відразу на декількох осередках кластеру для підвищення відмовостійкості і збереження даних у разі різного роду збоїв.

– відсутність схеми (schema-free) – Elasticsearch дозволяє завантажувати в нього звичайний JSON-об'єкт, а далі він вже сам все проіндексує, додасть в базу пошуку;

– RESTful API – Elasticsearch практично повністю керується по HTTP за допомогою запитів в форматі JSON.

2 ПРИНЦИПИ ТА СПОСОБИ РОБОТИ З ПОШУКОВОЮ СИСТЕМОЮ ELASTICSEARCH

2.1 Основні поняття та термінологія пошукової системи Elasticsearch

Elasticsearch [12] – високо масштабована пошукова система з відкритим вихідним кодом та аналітичним механізмом, який може надати не тільки пошук, але і складні агрегації. У системі підтримуються такі функцій, як автозаповнення, фільтри на основі геолокації, багаторівнева агрегація.

У Elasticsearch дані зберігаються у вигляді документів JSON (Javascript Object Notation). Більшість сховищ даних NoSQL використовують JSON для зберігання своїх даних, оскільки формат JSON дуже лаконічний, гнучкий і зрозумілий людям. Документ в Elasticsearch дуже схожий на рядок в порівнянні з реляційною базою даних.

Elascticsearch побудований для обробки неструктурованих даних і може автоматично визначати типи даних полів документу.

Для здійснення швидкого пошуку дані у Elasticsearch індексуються. Індекс – структурований набір даних, що дозволяє здійснювати швидкий доступ до них.

Логічний поділ різних видів даних здійснюється за їх типом. Наприклад, якщо створюється додаток для блогу, створюється тип для статей і коментарів.

Elasticsearch – це розподілена система, тобто вона складається з одного або декількох вузлів, які діють як одне ціле, що дозволяє масштабувати і обробляти навантаження, що перевищує те, що може обробити один сервер.

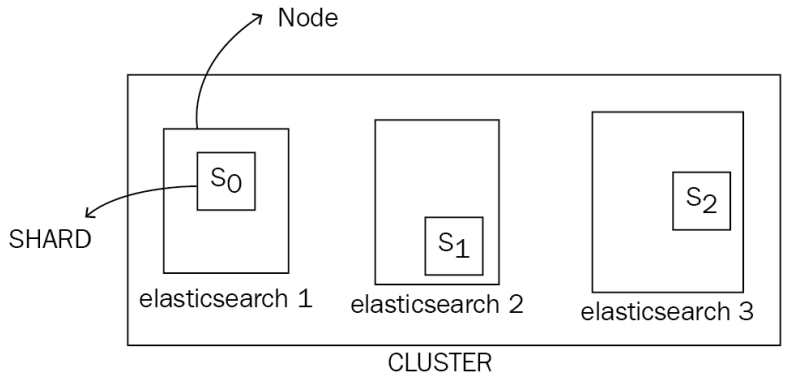


Рисунок 2.1 – Розподілення системи Elasticsearch по вузлах

На рисунку 2.1 кластер має три вузли з іменами elasticsearch1, elasticsearch2, elasticsearch3. Ці три вузла працюють разом, щоб обробляти всі запити індексування та вилучення даних. Залежно від потреб програми можна додавати і видаляти вузли (сервери).

Індекс являє собою набір з одного або декількох шардів. За рахунок чого Elasticsearch може зберігати інформацію обсяг якої перевищує можливості одного серверу. Elasticsearch використовує бібліотеку Apach Lucene для індексування та обробки запитів. Шард – це екземпляр бібліотеки Apache Lucene.

2.2 Архітектура вузлів у системі Elasticsearch

Коли використовується екземпляр бібліотеки Elasticsearch, використовується один кластер з одним вузлом. При створенні іншого екземпляру Elasticsearch з тим же cluster.name що й перший екземпляр, маємо є кластер з двома вузлами. Вузли кластеру «бачать» один одного та можуть взаємодіяти через TCP. Така топологія вузлів називається повнозв'язаною топологією (рисунок 2.2).

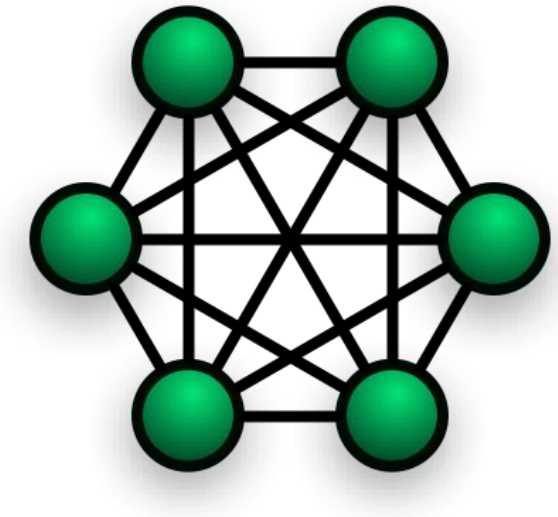


Рисунок 2.2 – Топологія вузлів у Elasticsearch

Крім того, вузли можуть спілкуватися із зовнішніми об’єктами засобами мови JSON через HTTP.

Отже, Elasticsearch – це однорангова децентралізована мережа вузлів, які спілкуються рівноправно.

Кожен вузол кластера має одну або кілька ролей; це може бути головний вузол, вузол даних, вузол клієнта або трайб вузол. Кожна роль має своє призначення.

Головний вузол відповідає за створення, видалення індексів, додавання вузлів або видалення вузлів із кластеру. Щоразу, коли стан кластеру змінюється, головний вузол передає зміни іншим вузлам кластера. Одночасно є лише один головний вузол кластеру.

Вузол даних відповідає за зберігання даних у шардах та виконання операцій, пов’язаних з даними, такими як створення, читання, оновлення, видалення, пошук та агрегація. У кластері може бути багато вузлів даних. Якщо один із вузлів даних зупиняється, кластер все ще працює і переорганізовує дані по інших вузлах.

Клієнтський вузол відповідає за маршрутизацію запитів, пов'язаних з кластером, до головного вузла та запитів, пов'язаних з даними, до вузлів даних,

він виконує функцію маршрутизатору (рис. 2.3, 2.4). Клієнтський вузол не містить жодних даних, також він не може стати головним вузлом.

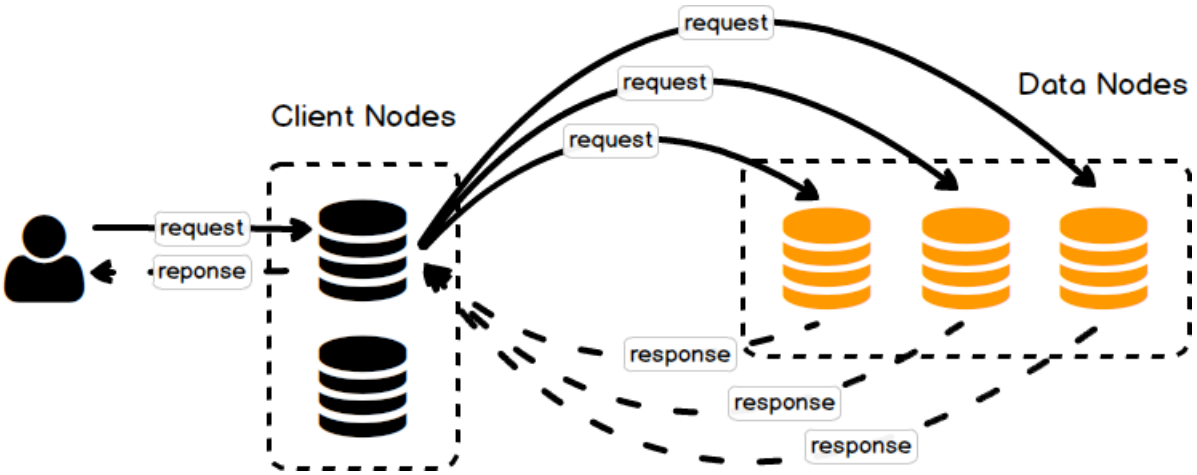


Рисунок 2.3 – Взаємодія клієнтських вузлів з вузлами даних

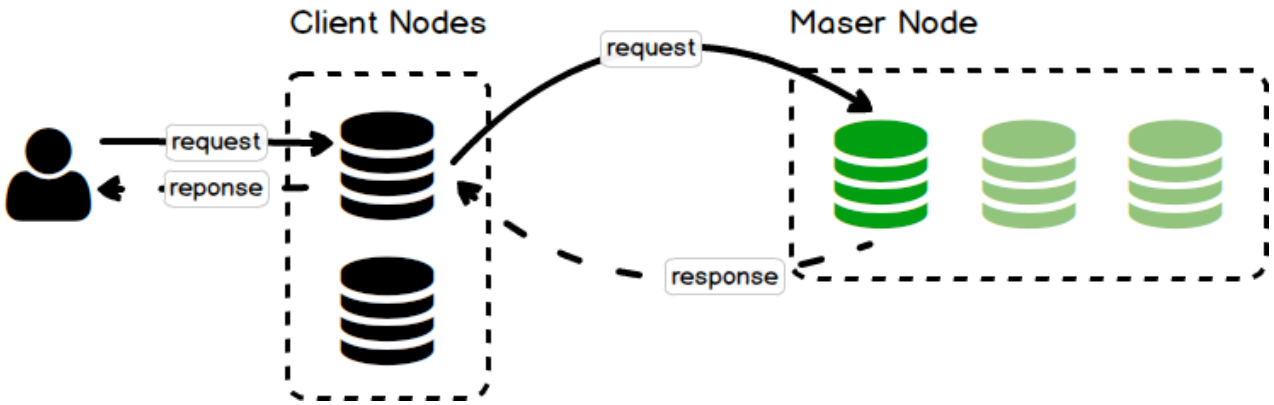


Рисунок 2.4 – Взаємодія клієнтських вузлів з головними вузлами

Трайб вузол – це особливий тип клієнтського вузла, який здатний спілкуватися з декількома кластерами для виконання пошукових та інших операцій.

Вузол, що приймається, відповідає за попередню обробку документів за врахування фактичної індексації.

Коли вузол запускається, він посилає пінг усім вузлам кластеру для пошуку головного вузла. Після того як головний вузол знайдено, він запитує у нього дозвіл на приєднання, надіславши відповідний запит; головний вузол приймає його як новий вузол кластеру, а потім повідомляє всі вузли кластеру

про приєднання нового вузла, і нарешті новий вузол підключається до всіх інших вузлів.

Якщо приєднаний вузол – це вузол даних, головний вузол буде перерозподіляти дані рівномірно по вузлах.

Якщо вузол зупинено або вузол кластеру не відповідає за певний проміжок часу, головний вузол видаляє його з кластеру і перерозподіляє дані, якщо видалений вузол є вузлом даних.

Головний вузол має механізм виявлення несправностей, він записує всі вузли в кластері та перевіряє чи доступні вони.

Кожен вузол кластеру також має механізм виявлення несправностей, він посилає пінг головному вузлу, щоб перевірити його наявність. Якщо головний вузол не відповідає, то у якості головного назначається інший вузол, який відповідає вимогам.

2.3 Алгоритм взаємодії модулю пошуку з системою Elasticsearch

Взаємодія з пошуковою системою Easticsearch реалізується за допомогою класів Storage, Request, Filter, які відповідно наслідуються від об’єктів AbstractStorage, AbstractRequest, AbstractFilter. Схему взаємодії класів цих об’єктів показано у додатку А.

Вказані об’єкти реалізують роботу з індексами (створюють, оновлюють та вилучають їх), запити до індексів та фільтрацію даних за допомогою наступних методів:

а) Storage:

1) getMaping();
2) getName();
3) create();
4) drop();

5) clear();

6) addItem();

7) addItems();

8) updateItem();

9) dropItem();

б) Request:

1) getStorage();

2) setRequestParams();

3) addFilters();

4) search();

в) Filter:

1) setParams();

2) getParams();

3) prepareParams().

2.4 Взаємодія з Elasticsearch засобами мови JSON

Основний спосіб взаємодії з Elasticsearch – REST API. За замовчуванням API – інтерфейс Elasticsearch працює на порту 9200.

Розглянемо основні CRUD операції API документів. Для організації пошуку у Інтернет-магазині з використанням Elasticsearch створюється індекс.

Індексування документу в Elasticsearch задається наступним чином:

```
PUT products
{
   "mappings" : {
      "properties" : {
         "category_id" : {
            "type" : "integer"
```

```
        },
        "manufacturer_id" : {
           "type" : "integer"
        },
        "color_id" : {
           "type" : "integer"
        },
        "price" : {
           "type" : "double"
        },
        "name" : {
           "type" : "text"
        },
        "description" : {
           "type" : "text"
        },
        "created_at" : {
           "type" : "date",
           "format": "yyyy-MM-dd HH:mm:ss"}} }}
```

Документ унікально ідентифікується індексом products.

Додавання документу до індексу відбувається командою POST, наприклад:

```
POST products/_doc
{
   "category_id": 1,
   "manufacturer_id": 1,
   "color_id": 4,
   "price": 1445.01,
   "name": "iPhone 11",
   "description": "iPhone Description",
```

```
    "created_at": "2019-11-01 10:00:19"
}
```

Для оновлення документу у індексі використовується команда PUT:

```
PUT products/_doc/<document_id>
{
    "category_id": 7,
    "manufacturer_id": 2,
    "color_id": 4,
    "price": 1645.01,
    "name": "iPhone 11",
    "description": "iPhone Description",
    "created_at": "2019-11-01 10:00:19"
}
```

Знаходження документу у створеному індексі за заданими критеріями виконується командою GET.

Для Інтернет-магазинів задаються наступні фільтри (програмний код фільтрів представлено у додатку Б):

– фільтр за категоріями;

– фільтр за категоріями, ціною і з назвою товару.

Для видалення документу відправляється запит:

```
DELETE products/_doc/<document_id>
```

Для отримання усіх документів індексу використовується запит:

```
GET products/_search
```

2.5 Мова запитів DSL

Однією з найпотужніших функцій Elasticsearch є DSL (Domain specific Language) або мова запитів. Вона дуже виразна і може використовуватись для визначення фільтрів, запитів, сортування, розбивки на сторінки та агрегації в одному запиті. Для виконання пошукового запиту, використовується HTTP-запит до _search Api. Індекс і тип, за яким потрібно виконати запит, вказується в URL-адресі. Індекси та тип не є необхідними. Якщо індекс / тип не вказаний, Elasticsearch виконує запит за всіма індексами в кластері. Пошуковий запит у Elasticsearch може бути виконаний двома різними способами:

– передаючи запит на пошук у якості параметрів запиту;

– передаючи запит пошуку у тілі запиту.

Простий пошуковий запит із застосуванням параметрів запиту:

```
GET example6/product/_search?q=product_name:куртка
```

Простий запит може бути виконаний із використанням параметрів URL-адреси. Попередній запит, переданий як тіло запиту, виглядає наступним чином:

```
POST example6/product/_search
{
  "query": {
    "match": {
      "product_name" : "куртка"}}}
```

Запит також може виконуватися одночасно за кількома індексами / типами, як показано тут:

```
POST example5,example6/product,product_reviews/_search
{
  "query": {
    "match": {
      "product_name" : "куртка"}}}
```

Базова структура запиту наступна:

```
{
   "size": // Кількість документів у відповіді. За замовчуванням 10.
   "from": // Зсув результатів
   "timeout": // Тайм-аут запиту. За замовчуванням його немає.
   "_source": // Поля, які потраплять у відповідь.
   "query": {// Запит}
   "aggs": {// Агрегація}
   "sort": {// Сортування результату}
}
```

Структура відповіді буде наступною:

```
{
  "took": // Час витрачений на отримання даних
   "timed_out": // Чи був перевищений тайм-аут, якщо його було встановлено для запиту.
  "_shards": {
   "total": // Кількість шардів задіяних у запиті
   "successful": // Кількість шардів, які повернули успішний результат
   "failed": // Кількість шардів з помилкою
  },
  "hits": {
   "total": // Загальна кількість документів
   "max_score": // Максимальний бал всіх документів
   "hits": [ // Перелік документів]}
```

Щоб обмежити кількість результатів у відповіді використовується пагінація, яка підтримується за допомогою полів from і size полів в запиті. Розмір сторінки за замовчуванням 10. Наприклад:

```
POST example6/_search
 {
  "from" : 0,
```

```
  "size" : 2,
  "query" : {
    "match" : {
      "product_name" : "куртка"}}}
```

Також існує можливість впорядкувати результати на основі одного або декількох полів. Наприклад, для сортування продукту за ціною:

```
POST example6/_search
 {
  "query": {
    "match": {
      "product_name": "куртка"
    }
  },
  "sort": {
    "unit_price": {
      "order": "desc"}}}
```

Так само можна сортувати за декількома полями відразу:

```
POST example6/_search
 {
  "query": {
    "match": {
      "product_name": "куртка"}},
  "sort": [
    {
      "unit_price": {
        "order": "desc"}},
   {
      "reviews": {
        "order": "desc"}}
```

При сортуванні з використанням декількох полів результати спочатку впорядковуються відповідно за першим полем, а потім- за іншими полями. Якщо значення першого поля унікальні, такі поля не матимуть ефекту. Тільки якщо два або більше документів мають одне і те ж значення першого поля, друге поле також буде використовуватися для сортування.

Коли виконується запит, за замовченням у відповідь повертається вихідний документ JSON. Є можливість включити тільки певні поля у відповідь, особливо коли документ великий або має багато полів.

Це можна зробити, вказавши поля в _source під час запиту. Значення полів витягуються з документа _source:

```
POST example6/product/_search
 {
   "_source": [
     "product_name"
   ],
   "query": {
     "term": {
       "product_name": "куртк"}}}
```

Відповідь буде наступною:

```
{
   "hits": {
     "total": 3,
     "max_score": 0.2876821,
     "hits": [
       {
         "_index": "example6",
         "_type": "product",
         "_id": "2",
         "_score": 0.2876821,
         "_source": {
```

"product_name": " Чоловіча водостійка куртка "}},
{
"_index": "example6",
"_type": "product",
"_id": "1",
"_score": 0.2876821,
"_source": {
"product_name": " Чоловіча якісна шкіряна куртка "}},
{
"_index": "example6",
"_type": "product",
"_id": "3",
"_score": 0.2876821,
"_source": {
"product_name": " Куртка жіноча вовняна "}}]}}

Поле _source також підтримує символи узагальнення для імен полів. Можна включити поля, які починаються з pr:

```
POST example6/product/_search
{
  "_source": [
    "pr*"
  ],
  "query": {
    "term": {
      "product_name": "куртк"}}}
```

Якщо необхідно повернути результат певних чисельних обчислень, запит виглядає наступним чином:

```
POST example6 / product / _search
{
  "_source": [],
```

```
  "query": {
    "match_all": {}
  },
  "script_fields": {
    "price_including_tax": {
      "script": {
        "source": "params [ '_ source'] [ 'unit_price'] * 1.1"}}}}
POST example6 / product / _search
{
  "_source": [],
  "query": {
    "match_all": {}
  },
  "script_fields": {
    "price_including_tax": {
      "script": {
        "source": "params [ '_ source'] [ 'unit_price'] * 1.1"}}}}
```

Мова сценаріїв за замовчуванням для Elasticsearch називається Painless. Використовуючи мову Painless, можна отримати доступ до _source документу для отримання unit_price і множення на 1.1 щоб розрахувати ціну, включаючи податок. Відповідь на попередній запит виглядатиме наступним чином:

```
{
  ....
  "hits": {
    "total": 3,
    "max_score": 1,
    "hits": [
      {
        "_index": "example6",
        "_type": "product",
```

```
   "_id": "2",
   "_score": 1,
   "_source": {
     "product_name": "Чоловіче водостійка куртка",
     "description": "Забезпечує комфорт під час їзди на велосипеді",
     "unit_price": 69.99,
     "reviews": 5,
     "release_date": "2017-03-02"
  },
  "fields": {
    "price_including_tax": [
     76.989]}},
{
  "_index": "example6",
  "_type": "product",
  "_id": "1",
  "_score": 1,
  "_source": {
    "product_name": "Чоловіче якісна шкіряна куртка",
    "description": "Кращий вибір. Всесезонная шкіряна куртка",
    "unit_price": 79.99,
    "reviews": 250,
    "release_date": "2016-08-16"
  },
  "fields": {
    "price_including_tax": [
     87.989]}},
{
  "_index": "example6",
  "_type": "product",
```

```
      "_id": "3",
      "_score": 1,
      "_source": {
        "product_name": "Куртка жіноча вовняна",
        "description": "Чи зігріє вас взимку",
        "unit_price": 59.99,
        "reviews": 10,
        "release_date": "2018-01-15"
      },
      "fields": {
        "price_including_tax": [
          65.989]}}]}
{
  "hits": {
    "total": 3,
    "max_score": 1,
    "hits": [
      {
        "_index": "example6",
        "_type": "product",
        "_id": "2",
        "_score": 1,
        "_source": {
          "product_name": "Чоловіче водостійка куртка",
          "description": "Забезпечує комфорт під час їзди на велосипеді",
          "unit_price": 69.99,
          "reviews": 5,
          "release_date": "2017-03-02"
      },
      "fields": {
```

```
    "price_including_tax": [
      76.989]}},
{
  "_index": "example6",
  "_type": "product",
  "_id": "1",
  "_score": 1,
  "_source": {
    "product_name": "Чоловіче якісна шкіряна куртка",
    "description": "Кращий вибір. Всесезонная шкіряна куртка",
    "unit_price": 79.99,
    "reviews": 250,
    "release_date": "2016-08-16"
  },
  "fields": {
    "price_including_tax": [
      87.989]}},
{
  "_index": "example6",
  "_type": "product",
  "_id": "3",
  "_score": 1,
  "_source": {
    "product_name": "Куртка жіноча вовняна",
    "description": "Чи зігріє вас взимку",
    "unit_price": 59.99,
    "reviews": 10,
    "release_date": "2018-01-15"
  },
  "fields": {
```

```
"price_including_tax": [
  65.989]}}]}
```

Відповідь включає первісну ціну і ціну, включаючи податок. Хоча сценарії дуже ефективні, кожен документ повинен аналізуватися для вилучення значення поля, яке може мати вартість виконання.

2.6 Релевантність пошуку у Elasticsearch

Традиційна база даних зазвичай містить структуровані дані. Запит в базі даних обмежує дані в залежності від різних умов, заданих користувачем. Кожна умова в запиті оцінюється як true / false, а рядки, які не задовольняють умовам, усуваються. Однак повнотекстовий пошук набагато складніше. Дані не структуровані.

Часто доводиться шукати один і той же текст в одному або декількох полях. Документи можуть бути досить великими, і слово запиту може з'являтися кілька разів в одному документі і в декількох документах. Відображення всіх результатів пошуку не допоможе, так як їх може бути сотні, якщо не більше, і більшість документів можуть навіть не мати відношення до пошуку.

Щоб вирішити цю проблему, всім документам, відповідним запитом, присвоюється оцінка. Оцінка присвоюється в залежності від того, наскільки є релевантним кожен документ для запиту. Потім результати оцінюються за критерієм релевантності. Результати на вершині, швидше за все, ті, що користувач шукає.

Давайте запитаємо example6 індекс, який ми створили на початку цього уроку. Розглянемо простий запит:

```
POST example6/_search
 {
```

```
  "query": {
    "match": {
      "product_name" : "шкіряна куртка"}}}
```

Відповідь запиту виглядає наступним чином:

```
{
  "hits": {
    "total": 3,
    "max_score": 0.5753642,
    "hits": [
      {
        "_index": "example6",
        "_type": "product",
        "_id": "1",
        "_score": 0.5753642,
        "_source": {
          "product_name": "Чоловіча якісна шкіряна куртка",
          "description": "Кращий вибір. Всесезонна шкіряна куртка",
          "unit_price": 79.99,
          "reviews": 250,
          "release_date": "2016-08-16" }},
      {
        "_index": "example6",
        "_type": "product",
        "_id": "2",
        "_score": 0.2876821,
        "_source": {
          "product_name": "Чоловіча водостійка куртка",
          "description": "Забезпечує комфорт під час їзди на велосипеді",
          "unit_price": 69.99,
          "reviews": 5,
```

```
    "release_date": "2017-03-02"}},
  {
   "_index": "example6",
   "_type": "product",
   "_id": "3",
   "_score": 0.2876821,
   "_source": {
    "product_name": "Куртка жіноча вовняна",
    "description": "Чи зігріє вас взимку",
    "unit_price": 59.99,
    "reviews": 10,
    "release_date": "2018-01-15"}}]}}
```


Документ з ID 1 оцінюється трохи вище документів 2 і 3. Оцінка розраховується з використанням BM25 алгоритму подібності. За замовчуванням результати сортуються з використанням _score значень.

На дуже високому рівні BM25 обчислюється оцінка на основі наступного:

– як часто термін з'являється в документі – тимчасова частота (tf);

– наскільки поширений термін для всіх документів – частота зворотнього документа (idf);

– документи, що містять всі або більшість умов запиту, оцінюються вище, ніж документи, що містять менше умов;

– нормалізація заснована на довжині документа, коротші документи оцінюються краще, ніж довші.

Не кожен запит потребує релевантності. Можна шукати документи, які точно відповідають значенням, наприклад статусу або пошуку документів в заданому діапазоні. Elasticsearch дозволяє комбінувати як структурований, так і повнотекстовий пошук в одному запиті. Запит Elasticsearch може бути виконаний в контексті запиту або в контексті фільтру. В контексті запиту релевантність _score обчислюється для кожного документу, що відповідає запиту.

2.7 Маршрутизація у шардах Elasticsearch

Індекс може складатися з одного або декількох шардів. Під час індексації ідентифікатор документу використовується для визначення того, до якого шарду віднести документ, використовуючи наступну формулу:

hash (document_id)% no_of_shards

При запиті документу використовується та сама формула для визначення шарду (рис.2.5).

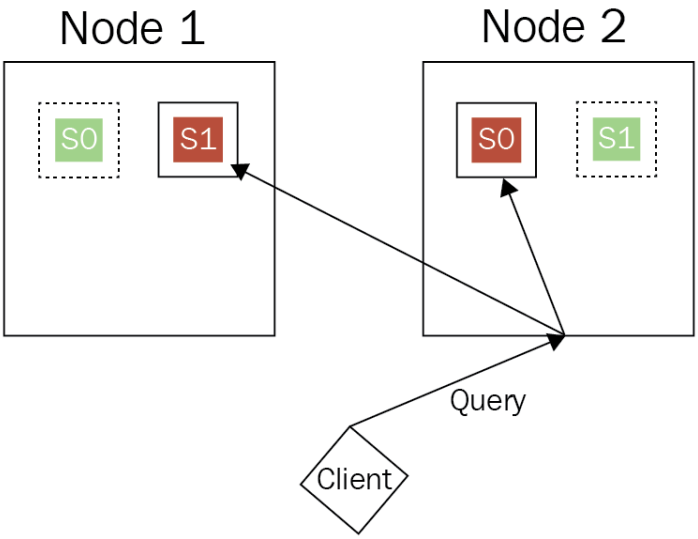


Рисунок 2.5 – Схема пошуку документу

При виконанні пошукового запиту, вузол який отримує запит, відомий як координаційний вузол. Координаційний вузол (Node2) відправляє запит до всіх шардів індексу, агрегує результати та відправляє їх клієнтові.

За замовчуванням запит повинен виконуватись на всіх шардах індексу. Але якщо є дані, які можна згрупувати за шардами, маршрутизація може використовуватись для відправки запитів на один шард замість усіх шардів у індексу.

Наприклад, якщо використати Elasticsearch для зберігання історій, закупок на сайті, користувач повинен мати можливість запитувати свої замовлення. Можна використовувати маршрутизацію для зберігання всіх

замовлень, що належать користувачеві в одному й тому ж шарді. При запиті замовлень користувачем, Elasticsearch використовує значення маршрутизації (ідентифікатор користувача) для визначення шарду та виконання запиту лише на одному шарді. Без маршрутизації запит буде виконуватись на всіх шардах.

Під час індексування вказується ідентифікатор, який використовується для визначення шарду. Значення маршрутизації використовується для визначення шарду. Наприклад, якщо всі замовлення індексуються за допомогою ідентифікатора користувача в якості значень маршрутизації, в такому випадку запит з використанням user_id буде виконуватись на одному шарді.

Задавати обов'язкову маршрутизацію можна так:

```
PUT example3/_mapping/order
{
 "_routing": {
   "required": true
  },
  "properties": {
   "order_id": {
     "type": "text"
   },
   "user_id": {
     "type": "keyword"
   },
   "order_total": {
     "type": "integer"}}}
```

Після цього індексується документ і встановлюється маршрутизація наступним чином:

```
PUT example3 / order / 1? Маршрутизація = user1
{
  "order_id": "23y86",
```

```
  "user_id": "user1",
  "order_total": 15
 }
```

Отримати документ замовлення, який було проіндексовано, можна наступним чином:

```
GET example3/order/1?routing=user1
```

Так як у маппінгу налагоджено обов'язкову маршрутизацію та при додаванні вказано значення маршрутизації рівне значенню ідентифікатору користувача, то при запиті документу треба вказати значення маршрутизації, щоб визначити шард, де лежить документ.

Данна техніка здатна зменшити навантаження на кластер, але використовувати її треба дуже обережно.

Наприклад, якщо данні користувачів можуть дуже відрізнятись за розміром від користувача до користувача, є вірогідність, що всі великі користувачі можуть бути в одному шарді. Це приводить до нерівномірного розподілу даних і недовикористання ресурсів і потенційно може привести до руйнування всього кластеру.

2.8 Висновки за розділом 2

Таким чином, у другому розділі розглянуто основні поняття, якими необхідно володіти для роботи з системою пошуку Elasticsearch на базі бібліотеки Apache Lucene. Також наведено алгоритм взаємодії з системою пошуку починаючи від етапу створення індексу до етапу пошуку у ньому даних. Розглянуто способи фільтрації даних та яким чином досягається релевантність пошуку у системі Elasticsearch.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Програмна реалізація модулю пошуку у Інтернет–магазині

Розроблено клієнт-серверний додаток, який взаємодіє з системою Elasticsearch засобами мови програмування PHP. На стороні «клієнта» реалізується інтерфейс на мові HTML, доступний через будь-який стандартний браузер під операційні системи сімейства Windows, у вигляді пошукової сторінки (рисунок 3.1).

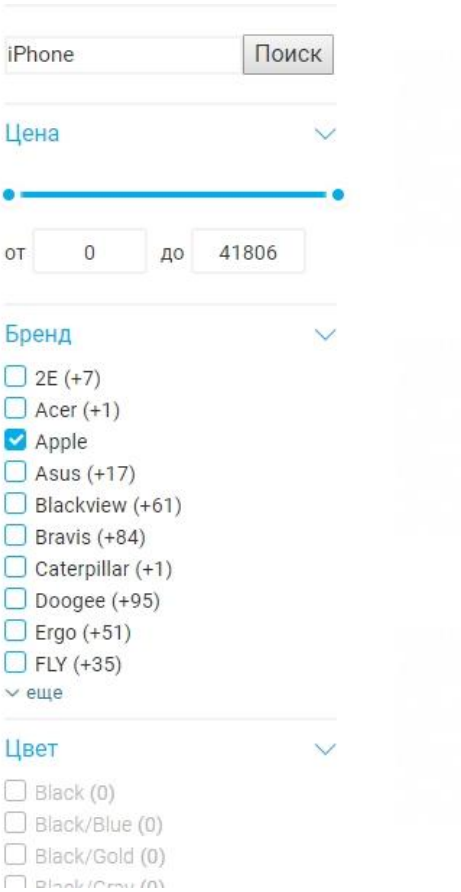


Рисунок 3.1 – Пошукова сторінка модулю

Взаємодія з системою Elasticsearch здійснюється через інтерфейс REST API.

Модуль здійснює відбір даних за наступними критеріями:

– фільтр за категорією;

– фільтр за виробником;

– фільтр за ціною;

– фільтр за кольором.

Методи, написані на мові програмування PHP, для реалізації роботи з даними розміщено у додатку В.

3.2 Інструкція з використання модулю

Розроблений модуль реалізує діалог між користувачем і механізмом пошуку, який реалізовано у системі Elasticsearch. Користувач вказує певні обмеження, які описують релевантні обмеження, використовуючи інтерфейс модулю.

Інтерфейс складається з пошукової сторінки, де вводиться ключове слово пошуку, та маркерів вибору ціни, категорії, кольору та виробника товару Інтернет магазину.

Приклади результатів застосування фільтрів представлено на рисунках 3.2 - 3.4.

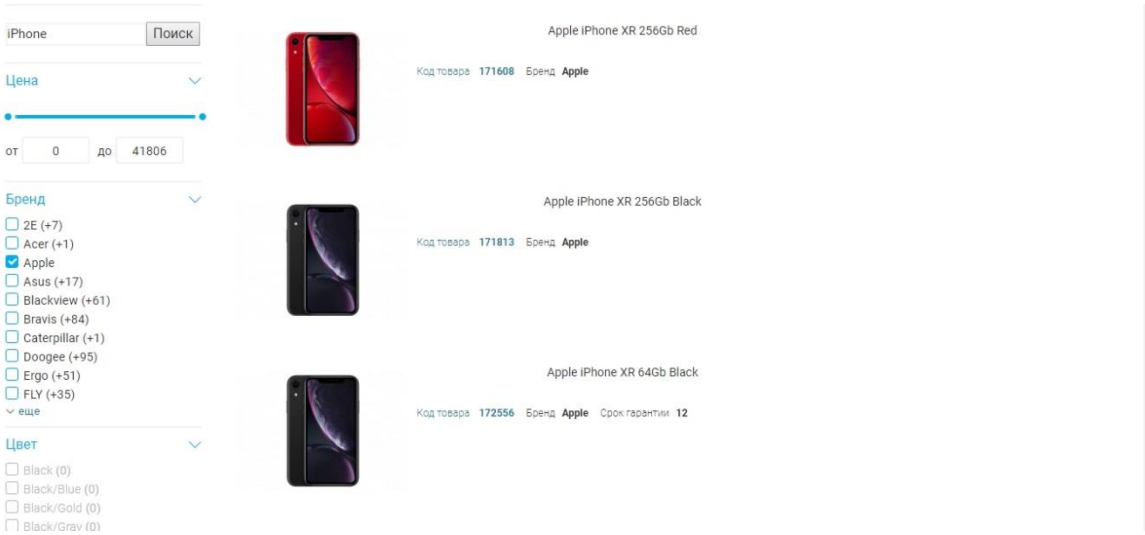


Рисунок 3.2 – Приклад використання фільтру за виробником

Рисунок 3.3 – Приклад використання фільтру за ціною

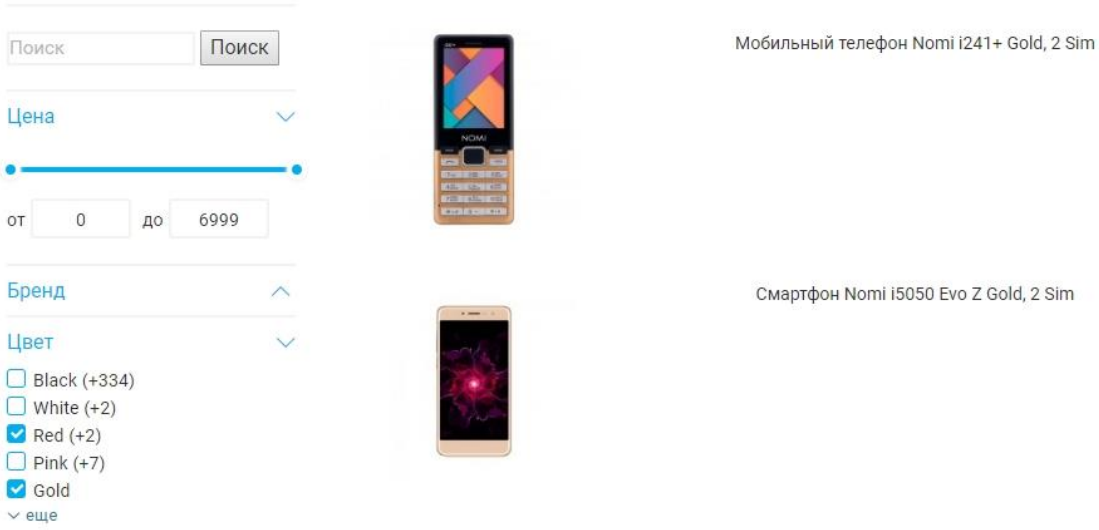


Рисунок 3.4 – Приклад використання фільтру за кольором

Механізм системи пошуку Elasticsearch використовує задані обмеження для збору збігів для передачі їх користувачеві, як показано на рис.3.2 – 3.4. Якщо знайдені збіги не задовольняють користувача, він переходить до уточнення критеріїв пошуку та повторює спробу.

Рисунок 3.6 – Доповнення запиту користувача

Головна взаємодія користувача та пошукового додатку відбувається у рядку введення запиту. Модуль надає інформацію під час введення, пропонуючи доповнення строки запиту (рис. 3.5).

3.3 Висновки за розділом 3

Представлений програмний модуль написано на мові програмування PHP. Він надає зручний користувацький інтерфейс доступу до механізму пошуку, реалізований у системі Elasticsearch. Модуль надає доступ до всього інструментарію бібліотеки Elasticsearch, реалізуючи гнучкий, швидкий та релевантний пошук даних за введеними критеріями пошуку.

Реалізований модуль може бути використаний Інтернет – магазинах, як потужний засіб пошуку у великих обсягах даних.

ВИСНОВКИ

У кваліфікаційній роботі було реалізовано модуль пошуку даних за введеними фільтрами. Для досягнення поставленої мети вирішено наступні завдання:

– проведено огляд існуючого інструментарію пошуку інформації на веб-сайті;

– вивчено основні поняття та термінологію пошукової системи Elasticsearch;

– розглянуто методи пошуку у системі Elasticsearch;

– розроблено алгоритм взаємодії з системою Elasticsearch;

– реалізовано програмний модуль пошуку по сайту з використанням Elasticsearch засобами PHP.

Розроблений модуль пошуку може бути використаний у Інтернет-магазинах, як потужний засіб здійснення швидкого, гнучкого та релевантного пошуку даних.

ПЕРЕЛІК ПОСИЛАНЬ

1. Розумний пошук KeaLabs. URL : https://kealabs.ru/ru/search (дата звернення: 4.10.2019)

2. Пошукова система Multisearch URL : https://multisearch.io/ (дата звернення : 4.10.2019)

3. Пошукова система AnyQuery URL: http://anyquery.diginetica.com/ (дата звернення : 5.10.2019)

4. Пошукова система ElasticSearch URL: https://www.elastic.co/ (дата звернення : 5.10.2019)

5. Erik Hatcher, Otis Gospodnetic. Lucene in Action. Stamford: Manning, 2010. 528 p.

6. Пошукова система Textocat URL: http://textocat.ru/ecommerce-search.html (дата звернення : 5.10.2019)

7. Пошукова система Detectum URL: http://detectum.com/ (дата звернення : 5.10.2019)

8. Пошукова система Sphinx URL: http://sphinxsearch.com/ (дата звернення : 5.10.2019)

9. Пошукова система Solr URL: https://lucene.apache.org/solr/ (дата звернення : 5.10.2019)

10. Пошукова система Nutch URL: https://nutch.apache.org/ (дата звернення : 5.10.2019)

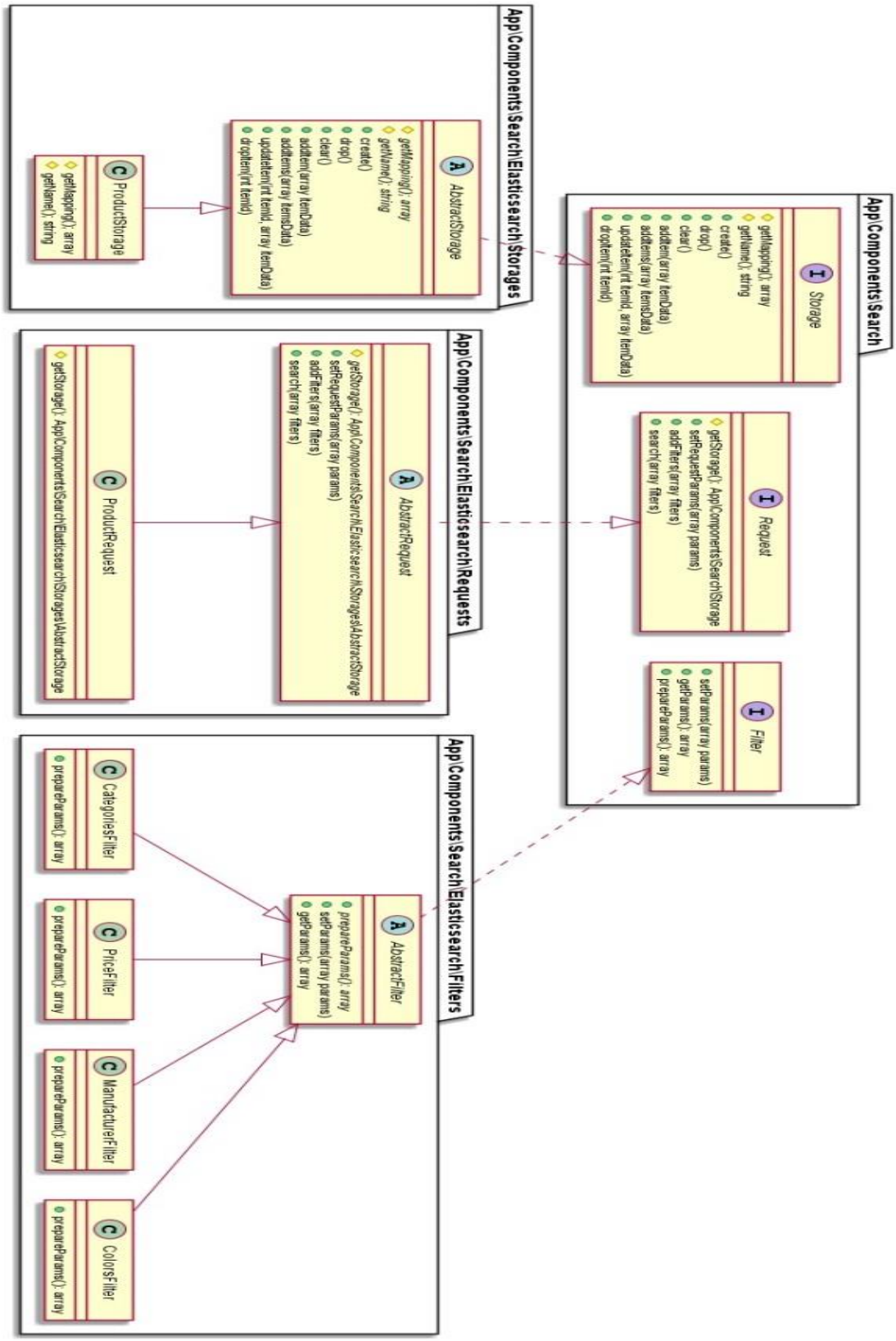
11. Пошукова система Xapian URL: https://xapian.org/(дата звернення : 5.10.2019)

12. Пранав Ш., Шарат К. Elasticsearch, Kibana, Logstash и поисковые системы нового поколения. Санкт-Петербург : Питер, 2019. 352 с.

13. Тарнбулл Д., Берримен Дж. Релевантный поиск с использованием Elasticsearch и Solr. Москва: ДМК-Пресс. 2016. 408 с.

ДОДАТОК А

UML-діаграма взаємодії класів



ДОДАТОК Б

HTTP – запити пошуку інформації за фільтром

– фільтр за категоріями:

```
GET products/_search
{
   "query": {
      "bool": {
         "must": [
            {
               "terms": {
                  "category_id": [
                     1,
                     2,
                     5]}}]}}}
```

– фільтр за категоріями і ціною:

```
GET products/_search
{
   "query": {
      "bool": {
         "must": [
            {
               "terms": {
                  "category_id": [
                     1,
                     2,
                     5
                  ]
```

```
            }
        },
        {
            "range": {
                "price": {
                    "gte": 1000,
                    "lte": 2500
                }
            }
        }
    ]
  }
 }
}
```

– фільтр за категоріями, ціною і з назвою товару:

```
GET products/_search
{
  "query": {
    "bool": {
      "must": [
        {
          "query_string": {
            "query": "iPhone"
          }
        },
        {
          "terms": {
            "category_id": [
              1,
```

```
                2,
                5
            ]
        }
    },
    {
        "range": {
            "price": {
                "gte": 1000,
                "lte": 2500
            }
        }
    }
]
}
}
}
```

ДОДАТОК В

Методи мови програмування PHP для роботи з фільтрами

– метод для отримання фільтрів:

```
public function getFilters(): array
{
    return [
        new CategoriesFilter($this->get('categories')),
        new PriceFilter($this->get('price_range')),
        new ManufacturerFilter($this->get('brands')),
        new ColorFilter($this->get('colors')),
    ];
}
```

– метод контролеру пошуку:

```
public function search(Request $request): array
{
    $filters = $request->getFilters();
    $query = $request->getQuery();
    $request = new ProductRequest();
    $request->setFilters($filters);
    $request->setQuery($query);
    $result = $request->search();

    return $result;
}
```

– метод для підготовки даних для фільтрації за кольором:

```
/**
 * @return array
 */
public function prepareParams(): array
{
    return [
        'terms' => [
            'color_id' => $this->getColors(),
        ],
    ];
}
```

– метод для підготовки даних для фільтрації за ціною:

```
/**
 * @return array
 */
public function prepareParams(): array
{
    return [
        'range' => [
            'price' => [
                'gte' => $this->getStartPrice(),
                'lte' => $this->getEndPrice(),
            ],
        ],
    ];
}
```

– метод для застосування фільтрів:

```
public function applyFilters()
```

```
{
    foreach ($this->filters as $filter) {
        $this->query['query']['bool']['must'][] = $filter->getParams();
    }
}
    public function autocompleteRequestBody()
    {
        return [
            'suggest' => [
                'text' => $this->getQuery(),
                'simple_phrase' => [
                    'phrase' => [
                        'field' => 'name',
                        'highlight' => [
                            'pre_tag' => '<b>',
                            'post_tag' => '</b>',
                        ]]]],];}
```