УДК 004.4
DOI 10.34822/1999-7604-2021-3-60-67

РЕАЛИЗАЦИЯ ПОЛНОТЕКСТОВОГО ПОИСКА В АВТОМАТИЗИРОВАННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЕ «СТУДЕНТ» С ПОМОЩЬЮ СИСТЕМЫ ELASTICSEARCH

Д. А. Кузин ✉, А. О. Осипов
Сургутский государственный университет, Сургут, Россия
✉ E-mail: kuzin_da@surgu.ru

Представлена архитектура разработанной автоматизированной информационной системы «Студент», в которой реализована технология полнотекстового поиска Elasticsearch. Данная система включает в себя функционал для проведения университетских онлайн- и оффлайн-мероприятий, хакатонов, конкурсов с командным и личным участием, а также для управления проектной деятельностью студенческих команд, подсчета рейтинга и проч.

Описаны различные подходы к реализации поисковых алгоритмов, построенных на основе реляционной системы управления базами данных, приведены основные сведения о поисковом движке, поисковом алгоритме Elasticsearch и процессе его настройки на примере поискового запроса.

Ключевые слова: полнотекстовый поиск, Elasticsearch, Okapi BM25.

IMPLEMENTATION OF FULL-TEXT SEARCH IN THE AUTOMATED INFORMATION SYSTEM “STUDENT” WITH ELASTICSEARCH SYSTEM

D. A. Kuzin ✉, A. O. Osipov
Surgut State University, Surgut, Russia
✉ E-mail: kzuin_da@surgu.ru

The article presents the architecture of the developed automated information system “Student”, which includes functionality to provide online and offline university events, hackathons, competitions with team and one-person participation, to manage project activity of student groups, to calculate ratings, etc., with the full-text search technology Elasticsearch implemented in it.

Different approaches to implement search algorithms developed on relational database management system are described. The article provides the main information about the search engine, the Elasticsearch search algorithm and the process of its setup on the example of search inquiry.

Keywords: full-text search, Elasticsearch, Okapi BM25.

Автоматизированная информационная система (АИС) «Студент» для учета и мониторинга активности участия пользователей в мероприятиях (http://student.surgu.ru) была разработана по заказу бюджетного учреждения высшего образования Ханты-Мансийского автономного округа – Югры «Сургутский государственный университет» (СурГУ) и введена в эксплуатацию в 2019 г. Первоначально система предназначалась для размещения информации и сопутствующего контента о различных мероприятиях, организуемых СурГУ, а также для регистрации участников и ведения общей статистики. После ее доработки в процессе использования введен следующий дополнительный функционал: проведение онлайн-мероприятий на платформе для видеоконференций BigBlueButton и конкурсов с командным участием; поддержка проектной деятельности студентов, формирование проектных команд и функционирование доски задач; реализация системы коммуникаций в виде чатов разного уровня и автоматических уведомлений, аналитических средств с визуализацией данных. На всех этапах работы пользователь

обеспечивается средствами аутентификации и поддержки персонального профиля. Приведенный перечень функций не является исчерпывающим и постоянно дополняется.

АИС «Студент» имеет микросервисную архитектуру с использованием контейнеризации Docker [1] и оркестрации контейнеров Docker Swarm. Пользовательская часть (frontend) системы является современным, функциональным и производительным web-приложением с развитым пользовательским интерфейсом и интерактивностью, использующим концепцию SPA (Single Page Application, одностраничное приложение) [2]. Это web-приложение, оболочка которого в виде необходимых HTML-, CSS- и JavaScript-файлов загружается при первой загрузке страницы, а контент подгружается по необходимости. Для разработки frontend применен популярный JavaScript-фреймворк VueJs [3]. Серверная часть (backend) системы написана на языке PHP с помощью фреймворка Laravel и в текущей версии содержит около 140 методов, защищенных протоколом аутентификации oAuth. Остальные микросервисы реализованы с использованием MySQL, Rabbit MQ, Metabase.

В любой информационной системе, оперирующей большими объемами данных, важное значение имеют средства поиска. Поисковые возможности могут существенно влиять на пользовательские качества системы, особенно в тех случаях, когда поиск необходимо выполнять по сущностям, имеющим сложную структуру. Для осуществления поиска по различным полям документов (записей) или их комбинации с наложением дополнительных условий часто реализуется интерфейс так называемого расширенного поиска в виде сложной формы с большим количеством элементов управления, назвать его дружественным для пользователя сложно. Для того чтобы облегчить пользователю задачу поиска и одновременно повысить полноту и релевантность его результатов, применяется специальный алгоритм, который будем условно называть «полнотекстовый поиск». Такой алгоритм должен обеспечивать поиск на основе введенной пользователем подстроки одновременно по нескольким полям документа с учетом морфологии, орфографии и семантики слов, а также количественную оценку релевантности всех найденных вхождений. Для повышения скорости работы такой алгоритм использует, как правило, собственные средства индексации.

В подсистеме организации проектной деятельности студентов АИС «Студент» сущность «проект» описывается следующим набором полей:

1) *id* – уникальный номер проекта (unsigned big integer);
2) *approved* – прошел проект модерацию или нет (boolean);
3) *archive_reason* – id причины, по которой отклонен проект (unsigned big integer);
4) *created_at* – дата создания проекта (datetime);
5) *customer* – заказчик проекта (varchar);
6) *deleted_at* – дата удаления проекта (datetime);
7) *participation_purpose* – цель участия заказчика (varchar);
8) *problem* – проблема, которую решает проект (varchar);
9) *project_type_id* – id типа проектов (unsigned big integer);
10) *responsible_user_id* – id пользователя (unsigned big integer);
11) *result* – планируемый результат проекта (varchar);
12) *task* – задача проекта (varchar);
13) *title* – название проекта (varchar);
14) *updated_at* – время последнего изменения проекта (varchar).

Полнотекстовый поиск проектов может быть реализован средствами реляционной системы управления базами данных (СУБД). Для этого в MySQL имеются специальные операторы MATCH и AGAINST. Для их использования необходимо создать индекс FULLTEXT в таблице для полей, которые участвуют в полнотекстовом поиске, тогда SQL-запрос для поиска подстроки «университет» по полям title, customer, problem, result будет выглядеть так:

```
SELECT id, MATCH (title, customer, problem, result) AGAINST
('*университеты*' IN BOOLEAN MODE) AS relevance FROM projects
```

Данный запрос вернет список id найденных проектов, отсортированный в порядке релевантности. Значения оценки релевантности будут находиться во втором столбце результата. Однако такой механизм поиска не будет удовлетворять всем описанным выше требованиям.

Для решения задачи полнотекстового поиска при реализации АИС «Студент» была использована система Elasticsearch – распределенный поисковый и аналитический движок с открытым исходным кодом, написанный на Java, представляющий собой документоориентированную СУБД, оптимизированную под операции поиска с поддержкой словоформ и синонимов, фильтрацией и другими возможностями [4]. В основе работы поискового алгоритма лежит инвертированный индекс – структура данных, в которой для каждого слова коллекции документов в соответствующем списке перечислены все документы, в которых оно встретилось.

Для вычисления релевантности документа запросу Elasticsearch использует формулу Okapi BM25 [5]. BM25 – поисковая функция на неупорядоченном множестве термов («мешке слов») и множестве документов, которые она оценивает на основе встречаемости слов запроса в каждом документе, без учета взаимоотношений между ними (например, близости). Это не одна функция, а семейство функций с различными компонентами и параметрами. Пусть дан запрос Q, содержащий слова q_1, q_2… q_n, тогда функция BM25 дает следующую оценку релевантности документа D запросу Q:

$$\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_i \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})},$$

где $f(q_i,D)$ – частота слова (term frequency, TF) q_i в документе D;

$|D|$ – длина документа (количество слов в нем);

avgdl – средняя длина документа в коллекции;

k_1 и b – свободные коэффициенты, обычно их выбирают как $k_1 = 2$, $b = 0{,}75$;

$IDF(q_i)$ – обратная документная частота (inverse document frequency, IDF) слова q_i.

IDF определяется как $log(N / n(q_i))$, где N – общее количество документов в коллекции, а $n(q_i)$ – количество документов, содержащих q_i.

Прежде чем использовать поисковые возможности Elasticsearch, необходимо внести туда данные. В случае, если в качестве основной СУБД информационная система использует реляционную, необходимо продублировать в Elasticsearch данные из таблиц, по которым будет осуществляться поиск, а в дальнейшем обеспечить их синхронизацию.

Основной структурой данных, которая используется в Elasticsearch, является индекс. Индекс – это структура данных, в которой для каждого слова, встречающегося в коллекции документов, хранится список ссылок на документы, в которых оно встречается. Документ – это набор определенных полей в формате JSON. Каждый документ связан с уникальным идентификатором, называемым UID. Elasticsearch позволяет сразу добавлять документы в формате JSON через интерфейс прикладных программ. Для этого делается POST-запрос, в пути указывается индекс, а в теле запроса – сам документ в формате JSON. Например, для добавления проекта в индекс student_surgu_projects необходимо отправить POST-запрос по адресу http://student.surgu.ru:9200/student_surgu_projects, а в тело запроса поместить документ в формате JSON:


```
{
    "id": 36,
    "customer": "Заказчик",
    "responsible_user_id": 705,
    "participation_purpose": "Цель участия",
    "resources": "Ресурсы",
    "created_at": "2021-02-20 19:09:35",
    "updated_at": "2021-02-20 19:09:35",
    "project_type_id": 1,
```

```
    "approved": 1,
    "approved_by_user_id": null,
    "logotype_url": "Ссылка на логотип",
    "deleted_at": null,
    "chat_id": 16,
    "project_archive_reasons_id": null,
    "features": [],
    "free": 0,
    "roles": [
        {
            "id": 41,
            "project_id": 36,
            "name": "Название роли",
            "quota": 1,
            "tasks": "Задачи",
            "created_at": "2021-02-20 19:09:35",
            "updated_at": "2021-02-20 19:09:35"
        }
    ]
}
```

Поскольку сервис Elasticsearch находится во внутренней сети Docker и недоступен для внешних запросов, используется простая (Basic) аутентификация по логину и паролю:

```
$this->client = ClientBuilder::create()
            ->setHosts(['student.surgu.ru:9200'])
            ->setBasicAuthentication(*****, '*****')
            ->build();
public function indexInElastic(Project $project)
    {
        $project->load('roles');
        $this->client->index([
            'index' => 'student_surgu_projects',
            'id' => $project->id,
            'body' => $project->toArray()
        ]);
    }
```

Теперь можно добавить в индекс все проекты из таблицы, кроме архивных и удаленных:

```
$prs = Project::query()->whereNull('deleted_at')-
>whereNull('project_archive_reasons_id')->get();
    foreach ($prs as $p){
        $service->indexInElastic($p);
```

GET-запрос для поиска подстроки «университет» в полях заголовка, результатах, задачах и ресурсах проекта будет выглядеть так:

```
{
    "query":{
        "bool":{
            "should": [
                {
                    "multi_match":{
                        "query": "университет",
                        "fields": [
                            "title",
                            "result",
                            "task",
                            "resources"
                        ]
                    }
```

```
                    }
                ]
            }
        }
    }
```

После выполнения запроса будет получен результат, в котором для каждого найденного проекта в поле _score присутствует значение релевантности, рассчитанное по формуле BM25:

```
{
    "took": 4,
    "timed_out": false,
    "_shards": {
        "total": 1,
        "successful": 1,
        "skipped": 0,
        "failed": 0
    },
    "hits": {
        "total": {
            "value": 7,
            "relation": "eq"
        },
        "max_score": 4.463116,
        "hits": [
            {
                "_index": "student_surgu_projects",
                "_type": "_doc",
                "_id": "117",
                "_score": 4.463116,
                "_source": {
                    "title": "3D Модель Университета"
                }
            },
            {
                "_index": "student_surgu_projects",
                "_type": "_doc",
                "_id": "111",
                "_score": 3.9647207,
                "_source": {
                    "title": "Культурный код СурГУ"
                }
            },
            {
                "_index": "student_surgu_projects",
                "_type": "_doc",
                "_id": "28",
                "_score": 3.6163511,
                "_source": {
                    "title": "Создание портала открытых данных"
                }
            }
        ]
    }
}
```

Для дополнительных возможностей поиска в Elasticsearch имеется анализатор (Analyzer). Analyzer – это конвейер (pipeline), который состоит из нескольких частей: Character Filter, Tokenizer и Token Filter. Character Filter обеспечивает предварительную обработку текста до того, как он попадет в Tokenizer. При помощи Character Filter можно вырезать из строки HTML-тэги или выполнить произвольное посимвольное преобразование строки

по массиву «ключ – значение» или регулярному выражению. Tokenizer разбивает поисковую строку на токены – отдельные слова, или «n-граммы» (n-символьные сочетания). Token Filter принимает на вход массив токенов из токенайзера и может их изменять (например, приводить к нижнему регистру), удалять (стоп-слова) или добавлять новые (синонимы). Таким образом, поиск по словосочетанию «университетские проекты» по умолчанию будет разбит на токены «университетские» и «проекты», и будут найдены все проекты, где в полях есть один из этих термов. В Elasticsearch есть стандартный анализатор «Russian», который убирает стоп-слова (предлоги, частицы и т. д), а также делает стемминг (находит начальную форму слова). Для использования этого анализатора по умолчанию и преобразования каждого проиндексированного документа необходимо задать mapping, сделав POST-запрос на http://student.surgu.ru:9200/student_surgu_projects/_mapping, и передать в теле запроса следующую структуру:

```
{
    "mappings": {
        "properties": {
            "title": {
                "type": "text",
                "analyzer": "russian"
            },
             "task":{
                "type": "text",
                "analyzer": "russian"
            },
            "problem":{
                "type": "text",
                "analyzer": "russian"
            },
            "result":{
                "type": "text",
                "analyzer": "russian"

            },
            "resources":{
                "type": "text",
                "analyzer": "russian"
            }
        }
    },
}
```

Для использования расширенных возможностей обработки запросов необходимо конструировать специальные анализаторы [6]. В качестве примера рассмотрим анализатор для поля «роль», который не только нормализует слова, но и подбирает к ним синонимы («программист», «разработчик», «devops», «frontend-*», «backend-*»), которые будут преобразованы к слову «программист» на этапе индексирования, что не только улучшит поиск, но и сократит индекс:

```
{
    "analysis": {
       "filter": {
            "russian_stop": {
            "type":       "stop",
            "stopwords":  "_russian_"
            },
            "russian_keywords": {
            "type":       "keyword_marker",
            "keywords":    []
```

```
                },
                "russian_stemmer": {
                "type":       "stemmer",
                "language":   "russian"
                },
                 "role_synonym" : {
                        "type" : "synonym_graph",
                        "lenient": true,
                        "synonyms": ["программист, разработчик, devops,
frontend-*, backend-* => программист"]
                        }
          },
        "analyzer": {
          "role_analyzer": {
            "type": "custom",
            "tokenizer": "standard",
            "char_filter": [
              "html_strip"
            ],
            "filter": [
              "lowercase",
              "russian_keywords",
              "russian_stop",
              "role_synonym"
            ]
          }
        }
      }}
```

Важными качествами Elasticsearch также являются:

1) масштабируемость – кластер Elasticsearch расширяется «на лету» добавлением новых серверов, при этом распределение нагрузки по узлам происходит автоматически;

2) отказоустойчивость – в случае сбоя кластерных узлов данные не потеряются, а будут перераспределены, поисковая система сама продолжит работу;

3) гибкость поисковых фильтров, включая нечеткий поиск, возможности работы с восточными языками (китайский, японский, корейский);

4) мультиарендность – в рамках одного объекта Elasticsearch можно динамически организовать несколько различных поисковых систем.

Благодаря наличию встроенных анализаторов текста Elasticsearch автоматически выполняет токенизацию, лемматизацию, стемминг и прочие преобразования текста для решения задач, связанных с поиском данных на естественном языке.

Серия экспериментов, проведенных на имеющемся в базе данных АИС «Студент» наборе данных, показала, что применение технологии полнотекстового поиска с использованием Elasticsearch кратно повышает количество релевантных результатов. Значения колеблются в зависимости от конкретного запроса. Например, по запросам «университет» и «университеты» обычный запрос к MySQL выдавал 2 и 0 проектов соответственно, в то время как применение Elasticsearch позволило в обоих случаях найти 7 проектов (рис.).

Обобщая, можно констатировать, что применение технологии Elasticsearch для поиска целесообразно в тех случаях, когда база данных содержит поля с текстом на естественном языке. Возможность настройки пользовательских анализаторов позволяет оптимизировать поисковый алгоритм для конкретной системы. Управляемость Elasticsearch по HTTP с помощью JSON-запросов упрощает интеграцию с существующими системами. Вместе с тем следует учесть, что поскольку Elasticsearch – это нереляционная СУБД, то при интеграции возможны дополнительные издержки, связанные с необходимостью конвертации базы в формат NoSQL. В некоторых случаях это может являться аргументом для отказа от реляционной модели и хранения всех данных в Elasticsearch [7].

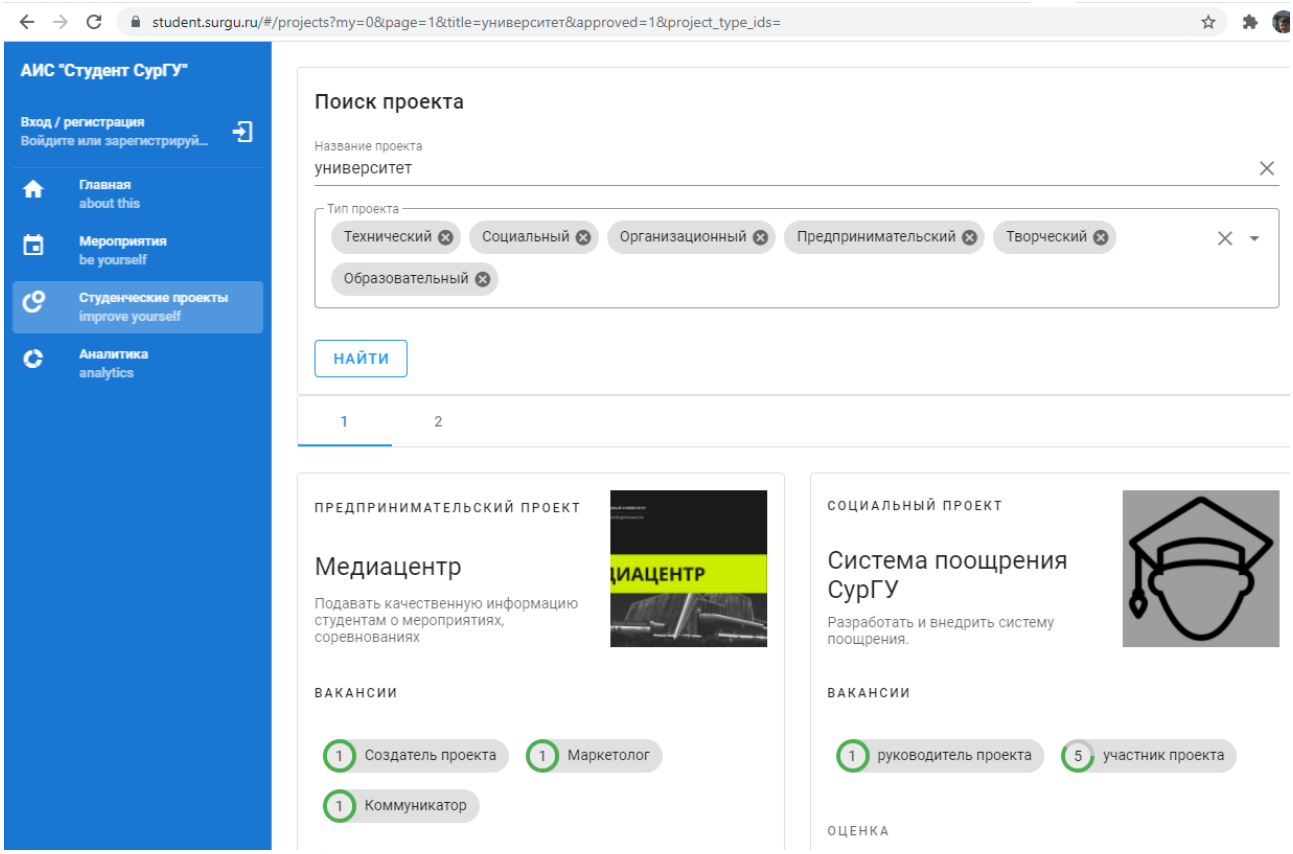

Рис. Результаты поиска проектов в АИС «Студент»
Примечание: скриншот авторов.

Литература

1. Docker: Empowering App Development for Developers. URL: https://www.docker.com/ (дата обращения: 03.09.2021).

2. The Single Page Interface Manifesto. URL: http://itsnat.sourceforge.net/php/spim/spi_manifesto_en.php (дата обращения: 05.09.2021).

3. API Reference. URL: https://v3.vuejs.org/api/ (дата обращения: 03.09.2021).

4. Elastic Enterprise Search. URL: https://www.elastic.co/enterprise-search (дата обращения: 07.09.2021).

5. Craswell N., Zaragoza H., Robertson S. Microsoft Cambridge at TREC-14: Enterprise Track // Proceedings of the Fourteenth Text REtrieval Conference. 2005.

6. Строим продвинутый поиск с Elasticsearch. URL: https://dou.ua/lenta/columns/building-advanced-search-with-elasticsearch/ (дата обращения: 17.09.2021).

7. Elasticsearch как NoSQL база данных. URL: https://habr.com/ru/company/percolator/blog/222765/ (дата обращения: 10.09.2021).