

## **Стажировка**

«Веб-приложение для публикации постов – поиск по фразам»

Выполнил: Ноздряков Богдан Валериевич

Проверил: Пармузин Александр Игоревич

Санкт-Петербург

2024

## Условие:

Необходимо реализовать новый режим поиска для метода получения всех постов – поиск по фразам.

Если есть посты:

1. “кирпичная дорогая изящная стена”
2. “кирпичная дорогая стена”

По фильтру `?filters[title][phraseSearch]=”кирпичная стена”` – был найден только второй пост, так как расстояние между запрашиваемыми словами не больше одного слова.

Должно быть реализовано нативными средствами PostgreSQL.

## Анализ:

Текущая реализация полнотекстового поиска позволяет найти слово, или набор слов в документах. Но также пользователь может быть заинтересованным в том, чтобы искать именно фразу. Например, пользователь ищет что-то вроде “быстро прыгать” или “прыгать очень быстро”. Если искать с помощью текущей реализации – “прыгать & быстро”, то будут найдены документы, содержащие эти слова, но мы получим любую случайную конфигурацию внутри документа, независимо от того, связаны они синтаксически или нет.

Итак, нам нужно внедрить в существующую реализацию полнотекстового поиска “механизм”, который будет не просто искать документы, в которых одновременно содержатся все слова из запроса: “прыгать & быстро”, а одновременно содержатся все слова из запроса в определенном диапазоне. По заданию, этот диапазон – не больше одного слова, то есть между запрашиваемыми словами может либо вообще не быть других слов: “прыгать быстро”, либо может быть только одно слово: “прыгать очень быстро”. Таким образом, мы сохраним преимущество полнотекстового поиска в получении различных форм слова, а также увеличим точность результатов, выдав только те, где запрашиваемые слова находятся максимально рядом друг с другом (задается расстояние между запрашиваемыми словами). Именно в этом и заключается фразовый поиск в контексте полнотекстового поиска.

Решить данную задачу с помощью нативных средств PostgreSQL невозможно (если не пытаться сделать что-то с помощью регулярных выражений, использование которых не является наилучшим решением, так как регулярные выражения будет тяжело правильно внедрить, а также они будут сильно влиять на производительность). Поэтому придется воспользоваться чем-то сторонним, что поможет нам в решении данной задачи. Для этого воспользуемся поисковым движком Elasticsearch.

## **ElasticSearch:**

ElasticSearch – это поисковая система, которая позволяет выполнять поиск документов более гибко, чем нативный PostgreSQL. Данный сервис предоставляет нереляционное хранилище данных, а также методы API, чтобы пользоваться данным хранилищем. Система позволяет работать с данными в формате JSON.

Разберемся подробнее что это все значит:

- 1) Реляционное хранилище данных – хранилище данных, где данные представляются в виде таблиц (система из строк и столбцов), а каждая запись (строка) в таблице связана с другими записями через ключи (таким образом таблицы связываются друг с другом). Здесь каждая запись (строка) имеет уникальный идентификатор (ключ), а столбцы предоставляют атрибуты данных. Данные хранилища используют язык запросов SQL для манипулирования данными. Поэтому такие хранилища называют SQL базы данных.
- 2) Нереляционное хранилище данных – хранилище данных, где для хранения применяется модель, которая оптимизирована для хранения определенного типа содержимого. Например, данные могут храниться в виде документов JSON, графов, а также ключ-значений. Такие хранилища не используют язык запросов SQL, и вместо него запросы осуществляются с помощью иных языков и конструкций. Поэтому такие хранилища называют NoSQL базы данных

Реляционные хранилища используют тогда, когда необходимо, чтобы база данных соответствовала требованиям ACID – атомарность, непротиворечивость, изолированность и долговечность. Это позволяет уменьшить вероятность неожиданного поведения системы и обеспечить целостность базы данных. Достигается подобное путем жесткого определения того, как именно транзакции взаимодействуют с базой данных. Также такие хранилища используются, когда данные, с которыми работают структурированы. То есть в реляционных хранилищах ставится в основу целостность данных.

Нереляционные хранилища используют, когда в основе гибкость и скорость, а не 100% целостность данных. Также нереляционные хранилища чаще используют, когда происходит работа с большими объемами данных из-за лучшей скорости работы с данными.

Разберем ACID:

- Атомарность – все изменения в рамках одной транзакции либо полностью применяются, либо вообще не применяются. Если происходит сбой во время выполнения транзакции, все изменения, сделанные в процессе этой транзакции, откатываются до состояния, которое было до начала транзакции. Это обеспечивает целостность данных, предотвращая появление несогласованных состояний в базе данных.

Примером может служить банковский перевод. Если перевод состоит из двух шагов: списание средств с одного счета и зачисление их на другой, оба этих действия должны быть выполнены успешно. Если один из шагов не удастся (например, из-за технических проблем), весь процесс откатывается, и средства остаются на исходном счете, сохраняя баланс счетов.

- Непротиворечивость - любая транзакция приводит базу данных из одного согласованного состояния в другое согласованное состояние. Все правила ограничений, установленные на данные (например, уникальность ключей, ограничения внешнего ключа), должны соблюдаться после завершения каждой транзакции. Это помогает избежать ситуации, когда данные находятся в неконсистентном состоянии.

Если, например, в базе данных есть правило, что каждый заказ должен иметь хотя бы одно связанное значение в таблице товаров, то попытка удалить последнее связанное значение для данного заказа должна привести к откату транзакции, чтобы сохранить целостность данных.

- Изолированность – гарантирует, что одновременно выполняемые транзакции не влияют друг на друга до тех пор, пока они не будут завершены. Это свойство позволяет транзакциям выполняться независимо друг от друга, даже если они работают с теми же данными. В результате, каждая транзакция видит базу данных в том состоянии, в котором она была в начале транзакции, исключая любые изменения, сделанные другими транзакциями, которые еще не были подтверждены.

Предположим, две транзакции одновременно пытаются увеличить счет пользователя на определенную сумму. Без изоляции одна транзакция могла бы увидеть промежуточное состояние счета, измененное другой транзакцией, что могло бы привести к неправильному итоговому значению счета. С помощью изоляции каждая транзакция видит фиксированное значение счета на протяжении всего своего выполнения, что обеспечивает корректность итогового результата.

- Долговечность - гарантирует, что после успешного завершения транзакции все изменения, внесенные в базу данных, сохраняются и остаются постоянными, даже в случае сбоя системы. Это означает, что даже если система перезагрузится или произойдет сбой, все транзакции, которые были успешно завершены до этого момента, будут сохранены в базе данных.

Если пользователь отправил платеж через интернет-банк, и эта транзакция была успешно завершена, но затем произошел сбой сервера перед тем, как информация о платеже была записана в базу данных, принцип долговечности гарантирует, что информация о платеже все равно будет сохранена и зарегистрирована в базе данных после восстановления системы.

Целостность данных – свойство базы данных, при котором всегда соблюдаются правила и ограничения, которые определяют допустимые состояния данных. Целостность данных важна для обеспечения точности и надежности информации, хранящейся в базе данных.

Например, если в базе данных есть ограничение, что возраст человека не может быть отрицательным числом, любая попытка вставить или обновить запись с отрицательным возрастом будет отклонена, чтобы поддерживать целостность данных.

Транзакция – это набор CRUD операций по работе с базой данных, объединенных в одну атомарную пачку. Чтобы обратиться к базе данных – нужно сначала открыть соединение с ней. Это называется коннект. Коннект – это просто труба, по которой мы посылаем запросы. Чтобы сгруппировать запросы в одну атомарную пачку, используем транзакцию. Транзакцию надо:

1. Открыть.
2. Выполнить все операции внутри.
3. Закрыть.

Как только мы закрыли транзакцию, труба освободилась. И ее можно переиспользовать, отправив следующую транзакцию.

Можно, конечно, каждый раз закрывать соединение с БД. И на каждое действие открывать новое. Но эффективнее переиспользовать текущие. Потому что создание нового коннекта — тяжелая операция, долгая. Каждая транзакция удовлетворяет ACID.

NoSQL базы позволяют выполнять то, чего не умеют SQL базы:

- База данных NoSQL хранит большие объемы неструктурированной информации. То есть такая база данных не накладывает ограничений на типы хранимых данных. Более того, при необходимости в процессе работы можно добавлять новые типы данных
- Использование облачных хранилищ — требует, чтобы данные можно было легко распределить между несколькими серверами для обеспечения масштабирования. Использование, для тестирования и разработки, локального оборудования, а затем перенос системы в облако, где она и работает — это именно то, для чего созданы NoSQL базы данных
- Нереляционные базы данных работают быстрее реляционных баз, так как они не используют язык запросов SQL для работы с данными, а используют другие конструкции и языки, которые позволяют работать с данными быстрее

Как было сказано ранее, Elasticsearch относится к категории NoSQL, то есть предоставляет нереляционное хранилище данных. Также, как было сказано ранее, в базу данных Elasticsearch данные с помощью API отправляются в виде документов JSON и хранятся в таком же виде:

```
{  
  "id": "223d526d-5064-455e-9daf-6e7e3ad3e77d",  
  "title": "Футбол в начале нулевых",  
  "authorId": "93243b0e-6fbf-4a68-a6c1-6da4b4e3c3e4"  
}
```

Такой выбор был обусловлен тем, что необходимо было добиться хорошей скорости поиска при очень больших данных, что как-раз и дают нереляционные хранилища. Все дело в том, что намного быстрее читать из объектов, содержащих все необходимое здесь и сейчас. И намного проще вносить изменения в неструктурированную схему данных.

ElasticSearch сохраняет документ в индекс кластера и делает его доступным для поиска. После этого можно найти и извлечь документ, используя API ElasticSearch.

Основой для работы с текстовыми документами является анализатор. Он представляет собой цепочку последовательных обработчиков. Сначала поступивший в анализатор текст проходит символьные фильтры, которые убирают, добавляют или заменяют отдельные символы в потоке. Например, с помощью символьных фильтров можно заменить арабские цифры (٠ ١ ٢ ٣ ٤ ٥ ٦ ٧ ٨ ٩) на современные арабские цифры (0123456789), перевести текст в нижний регистр или удалить HTML-теги.

Затем Elasticsearch передает обработанный текст токенизатору, который очищает поток символов от знаков препинания и разбивает по определенным правилам на отдельные слова — токены. В зависимости от токенизатора можно получить как набор слов, так и набор, где будут лишь основы слов (например, корни).

После токенизатора система передает набор слов в один или несколько фильтров токенов, которые могут добавлять, удалять или менять слова в наборе. Например, фильтр стоп-токенов может удалять часто используемые служебные слова, такие как артикли *a* и *the* в англоязычном тексте. На выходе из анализатора мы имеем набор токенов, который помещается в индекс. Это позволяет сохранять максимум смысла при минимуме знаков.

Elasticsearch ищет слова из запроса уже по индексу. При этом поисковые индексы можно разделить на сегменты — шарды. На каждом узле (запущенном экземпляре Elasticsearch) может быть размещено несколько сегментов. Каждый узел действует как координатор для делегирования операций правильному сегменту, а перебалансировка и маршрутизация выполняются автоматически.

Такие системы, как Elastic, могут использоваться как самостоятельные хранилища, так и в купе с базой данных. Когда база данных (например, Postgres) используется вместе с ElasticSearch, то подразумевается, что данные сначала ищутся в Elastic, а затем по найденным id извлекаются данные из базы данных Postgres, так как в ней хранится больше данных, чем в Elastic. На больших объемах это может быть полезно, так как поиск через движок происходит быстрее, однако нужна постоянная синхронизация, а данные дублируются и занимают вдвое больше места. Синхронизация подразумевает, что данные пишутся в основную базу данных Postgres и сразу же после этого — в базу данных ElasticSearch. С удалением также. Здесь могут быть проблемы. Если процесс оборвался на

середине, то поскольку нет поддержки транзакций – набор данных в двух системах начинает отличаться.

### Индексы в Elasticsearch:

Ранее мы много упоминали индекс в контексте Elasticsearch. Однако как именно он работает и что это вообще такое:

Шард в Elasticsearch – это логическая единица хранения данных на уровне базы, которая содержит часть данных индекса. Шарды позволяют распределить данные по различным узлам в кластере для обеспечения масштабируемости и отказоустойчивости. Когда создается индекс, определяется количество первичных и реплицированных шардов. Первичные шарды отвечают за обработку всех операций записи и чтения, в то время как реплики используются для увеличения доступности данных и балансировки нагрузки. Все документы хранятся именно в шардах.

Индекс в Elasticsearch – это одновременно и распределенная база и механизм управления и организации данных, это именно логическое пространство. Индекс содержит один или более шардов, их совокупность и является хранилищем. То есть это место, где хранятся все данные. Индекс состоит из одного или нескольких шардов и используется для организации данных по схожести. Можно представить индекс как аналог таблицы в реляционной базе данных, где каждая строка представляет собой документ, а столбцы — поля документа.

Разберемся с кластером:

В Elasticsearch за операции поиска и индексации отвечает отдельный инстанс Lucene(*шард*). Для того, чтобы обращаться к распределенной системе шардов, нам необходимо иметь некий координирующий узел, именно он будет принимать запросы и давать задания на запись или получение данных. То есть помимо хранения данных мы выделяем еще один вариант поведения программы — координирование.

Каждый запущенный экземпляр **Elasticsearch** является отдельным узлом (node). **Cluster** — это совокупность определенных нод. Когда запускается один экземпляр - кластер будет состоять из одной ноды.

Таким образом мы изначально ориентируемся на два вида узлов — CRUD-узлы и координирующие узлы, но их может быть больше. У нас есть куча машин, объединенных в сеть и все это очень напоминает кластер. Можно сказать, что:

- Узел – это сервер (отдельный экземпляр Elasticsearch), который хранит данные и участвует в обработке запросов к этим данным – предоставляет API для работы с документами. Узлы могут быть разных типов, включая узлы данных (хранят и обрабатывают данные), координаторские узлы (координируют запросы между

клиентами и узлами данных), управляющие узлы (отвечают за управление кластером) и т.д.

- Кластер — это коллекция одного или нескольких узлов Elasticsearch, которые работают вместе для обеспечения индексации данных и выполнения запросов к этим данным. Кластеры обеспечивают высокую доступность и масштабируемость, позволяя распределение данных и нагрузки между узлами.

### **Преимущества Elasticsearch:**

- Хорошее качество и скорость обработки текста
- Возможность сочетать в поиске как полнотекстовый поиск, так и фразовый поиск (учитываются расстояния между запрашиваемыми словами)
- Легкое управление
- Отказоустойчивость (при сбое одного из узлов индекс перераспределяется на оставшиеся узлы, используя внутренний механизм репликации данных. Кластеры Elasticsearch продолжают работать, даже если возникают аппаратные ошибки типа сбоя узла или неполадок сети)
- Высокая горизонтальная масштабируемость (система может быть запущена на десятках или сотнях узлов вместо одного мощного сервера, а добавить в кластер еще один узел можно практически без дополнительных настроек)

### **Недостатки Elasticsearch:**

- Необходимость в постоянной синхронизации с основной базой данных
- Необходимость хранить данные, помимо основной базы данных, в базе данных Elastic. То есть данные занимают вдвое больше места
- Нет поддержки транзакций, из-за чего может нарушиться синхронизация с основной базой данных в случае какого-нибудь сбоя
- Использование клиента Elasticsearch может быть нелучшим решением, так как при построении сложных запросов – могут появиться огромные нечитаемые циклы

### **ElasticSearch – нативный PostgreSQL:**

Используя нативный PostgreSQL – мы получим ограниченную функциональность. В некоторых случаях ее будет достаточно, но не всегда (например, нативный PostgreSQL не сможет реализовать фразовый поиск, который требуется реализовать по текущему заданию). И как-раз в случаях, когда такой функциональности будет недостаточно – на помощь приходит Elasticsearch.



## Решение:

### Настройка Docker:

Как было описано ранее, заданный фразовый поиск будет реализован с помощью поискового движка Elasticsearch. Чтобы не устанавливать Elasticsearch на свой компьютер – воспользуемся Docker для запуска экземпляра Elasticsearch в Docker-контейнере. Мы воспользуемся docker-compose для развертывания экземпляра Elasticsearch. Для этого создадим файл docker-compose.yaml:

```
🐙 docker-compose.yaml
1  version: '3.6'
2  services:
3    elasticsearch:
4      container_name: elasticsearch_container
5      image: elasticsearch:8.14.1
6      volumes:
7        - esdata:/usr/share/elasticsearch/data
8      environment:
9        - discovery.type=single-node
10       - xpack.security.enabled=false
11      logging:
12        driver: none
13      ports:
14        - 9200:9200
15        - 9300:9300
16      networks:
17        - esnet
18  volumes:
19    esdata:
20  networks:
21    esnet:
```

Разберем данный файл:

- version '3.6' – указываем версию используемого docker-compose
- в services мы определяем все сервисы, то есть контейнеры, которые будут запущены. В данном случае – будет запущен только один контейнер elasticsearch
- в elasticsearch мы указываем параметры для данного контейнера
- container\_name – указываем имя данного контейнера, которое отображается в списке запущенных контейнеров

- `image` – образ Docker, из которого будет создан контейнер. В данном случае – образ ElasticSearch версии 8.14.1
- `volumes` – задаем тома, которые будут монтированы в контейнер. В данном случае монтируется том `esdata`, использующийся для хранения данных ElasticSearch (если не создавать тома, то после перезагрузки Docker – все данные из ElasticSearch будут потеряны)
- `environment` – устанавливаем переменные окружения в контейнер. Здесь `discovery.type=single-node` – устанавливаем тип обнаружения как одноузловый; `xpack.security.enabled=false` – так как нам не нужно иметь дело с SSL, поскольку приложение запускается локально (данная конфигурация подходит только для локальной разработки. Для производственного развертывания – нужны будут другие параметры)
- `logging` – задаем настройки логирования. В данном случае – мы отказываемся от вывода логов в какой-нибудь файл или консоль
- `ports` – определяем порты, которые будут проброшены из контейнера на хост-машину. В данном случае мы пробрасываем порты 9200 и 9300, которые будут использоваться для доступа к ElasticSearch
- `networks` – указываем сеть, в которой должен работать контейнер. В данном случае – это сеть `esnet`

После создания `docker-compose.yml`, мы можем запустить экземпляр ElasticSearch в Docker через команду `docker-compose up`. Для этого внесем в `package.json` новый скрипт:

```
"start:compose": "docker-compose up",
```

Теперь, перед запуском приложения, нужно запустить `npm run start:compose` для запуска экземпляра ElasticSearch.

### **Внедрение ElasticSearch в приложение:**

После настройки Docker, можно переходить к написанию функционала, который позволит нам обращаться к базе данных ElasticSearch. Но для начала – отменим миграцию, которую мы создали при реализации полнотекстового поиска нативными средствами PostgreSQL (напомню, что эта миграция создавала колонку `text_tsvector` типа `tsvector` в таблице `Post`, создавала триггер для обновления `text_tsvector` и создавала индекс GIN. Для этого в месте применения всех миграций (метод `syncWithDb` класса `SequelizeService` - `./src/database/sequelize/service/sequelize.service.ts`) – отменим данную миграцию:

```

public async syncWithDb(): Promise<void> {
  try {
    await this.sequelize.sync();
    await this.umzug.up();
    await this.umzug.down({
      to: '20240719151942-add_text_tsv_column_to_post_table.js'
    });
  }
  catch (err) {
    console.log(err);
  }
}
}

```

Создадим новый класс ElasticSearchProvider (/src/utils/lib/elasticsearch/elasticsearch.provider.ts), который будет провайдером к клиенту ElasticSearch, что позволит нам работать с базой данных ElasticSearch:

```

src > utils > lib > elasticsearch > TS elasticsearch.provider.ts > ElasticSearchProvider > createIndex
1  import { Client } from '@elastic/elasticsearch';
2  import {
3    SearchRequest,
4    SearchResponse,
5    BulkRequest,
6    IndicesCreateRequest
7  } from '@elastic/elasticsearch/lib/api/types';
8
9  class ElasticSearchProvider {
10   private readonly client: Client;
11
12   constructor() {
13     this.client = new Client({
14       node: process.env.ELASTIC_URL_DEV
15     });
16   }
17
18   public async isIndexExist(indexName: string): Promise<boolean> {
19     return (
20       await this.client.indices.exists({
21         index: indexName
22       })
23     );
24   }
25
26   public async createIndex(indexSettings: IndicesCreateRequest): Promise<void> {
27     await this.client.indices.create(indexSettings);
28   }
29

```

```

30 public async populateIndex(docs: BulkRequest[]): Promise<void> {
31     await this.client.bulk({
32         refresh: true,
33         operations: docs
34     });
35 }
36
37 public async deleteIndex(indexName: string): Promise<void> {
38     await this.client.indices.delete({
39         index: indexName
40     });
41 }
42
43 public async indexDocument(indexName: string, documentId: string, documentBody: object): Promise<void> {
44     await this.client.index({
45         index: indexName,
46         id: documentId,
47         document: documentBody
48     });
49 }

```

```

51 public async updateDocument(
52     indexName: string,
53     documentId: string,
54     newDocumentBody: object
55 ): Promise<void>
56 {
57     await this.client.update({
58         index: indexName,
59         id: documentId,
60         doc: newDocumentBody
61     });
62 }
63
64 public async deleteDocument(indexName: string, documentId: string): Promise<void> {
65     await this.client.delete({
66         index: indexName,
67         id: documentId
68     });
69 }
70
71 public async searchByRequest(searchRequest: SearchRequest): Promise<SearchResponse> {
72     return (
73         await this.client.search(searchRequest)
74     );
75 }
76 }
77 export default ElasticSearchProvider;

```

Разберем данный класс:

- В конструкторе мы создаем клиент ElasticSearch через подключение к серверу ElasticSearch по его URL (определяем URL в переменной ELASTIC\_URL\_DEV файла env). Данный клиент позволит нам работать с базой данных ElasticSearch
- isIndexExist – метод, который через созданный клиент, проверяет – существует ли заданный индекс в базе данных ElasticSearch

- `createIndex` - метод, который через созданный клиент, создает индекс в базе данных Elasticsearch через переданную конфигурацию индекса
- `populateIndex` - метод, который через созданный клиент, заполняет индекс множеством переданных документов в базе данных Elasticsearch
- `deleteIndex` - метод, который через созданный клиент, удаляет указанный индекс в базе данных Elasticsearch
- `indexDocument` - метод, который через созданный клиент, добавляет в указанный индекс один документ в базе данных Elasticsearch
- `updateDocument` - метод, который через созданный клиент, обновляет указанный документ в указанном индексе базы данных Elasticsearch
- `deleteDocument` - метод, который через созданный клиент, удаляет указанный документ в указанном индексе базы данных Elasticsearch
- `searchByRequest` - метод, который через созданный клиент, производит поиск в базе данных Elasticsearch

Также не забудем создать модуль `ElasticSearchModule` (`./src/utls/lib/elasticsearch/elasticsearch.module.ts`), который будет создавать синглтон `ElasticSearchProvider` и делать его доступным для инъектирования:

```
src > utls > lib > elasticsearch > TS elasticsearch.module.ts > ...
1  import dependencyContainer from '../dependencyInjection/dependency.container';
2  import ElasticSearchProvider from './elasticsearch.provider';
3
4  class ElasticSearchModule {
5      constructor() {
6          dependencyContainer.registerInstance('esProvider', new ElasticSearchProvider());
7      }
8  }
9  export default ElasticSearchModule;
10
```

Также нужно создать сам объект `ElasticSearchModule` в главном модуле приложения `AppModule` (`./src/app.module.ts`):

```
dependencyContainer.registerInstance('esModule', new ElasticSearchModule());
```

После создания провайдера `ElasticSearch`, нужно написать функцию, которая создаст индекс в базе данных Elasticsearch и добавит все существующие посты в этот индекс (напомню, что данный поиск применяется только к постам пользователя). Для этого создадим в конфигураторе приложения (`./src/app.config.ts`) функцию `setupElasticSearch`:

```

43 export async function setupElasticSearch() {
44   const postService = dependencyContainer.getInstance<PostService>('postService');
45   const esProvider = dependencyContainer.getInstance<ElasticSearchProvider>('esProvider');
46
47   const postIndex = postService.getEsIndex();
48   const isPostIndexExist = await esProvider.isIndexExist(postIndex);
49
50   if (!isPostIndexExist) {
51     await esProvider.createIndex({
52       index: postIndex,
53       mappings: {
54         properties: {
55           id: {
56             type: 'keyword'
57           },
58           title: {
59             type: 'text',
60             analyzer: 'russian'
61           },
62           content: {
63             type: 'text',
64             analyzer: 'russian'
65           },
66           authorId: {
67             type: 'keyword'
68           }
69         }
70       }
71     });

```

```

73   const docs: object[] = [];
74   const posts = await postService.getAllUserPosts({});
75
76   posts.forEach(post => {
77     docs.push({
78       index: {
79         _index: postIndex,
80         _id: post.id
81       }
82     });
83     docs.push({
84       id: post.id,
85       title: post.title,
86       content: post.content,
87       authorId: post.authorId
88     });
89   });
90
91   await esProvider.populateIndex(docs);
92 }
93 }

```

В данном методе мы получаем индекс для постов из сервиса PostService, где он был задан (подробнее разберем далее). После мы проверяем – существует ли полученный индекс в базе данных Elasticsearch. Если индекс существует, то метод прекращает свою работу, так как индекс уже был внедрен. Если же индекса не существует, то в таком случае он создается в базе данных Elasticsearch, а после в него добавляются все существующие документы (все существующие посты). Все это возможно за счет созданного ранее провайдера ElasticsearchProvider.

При создании индекса:

```
await esProvider.createIndex({
  index: postIndex,
  mappings: {
    properties: {
      id: {
        type: 'keyword'
      },
      title: {
        type: 'text',
        analyzer: 'russian'
      },
      content: {
        type: 'text',
        analyzer: 'russian'
      },
      authorId: {
        type: 'keyword'
      }
    }
  }
});
```

Мы также определяем структуру документов, которые будут храниться в данном индексе. В данном случае – каждый документ будет содержать:

- id своего поста
- title своего поста
- content своего поста
- authorId своего поста

-title и content мы добавили, так как сам фразовый поиск осуществляется именно по столбцам title и content постов;

-id мы добавили, чтобы пользователь после применения фразового поиска, видел id каждого найденного поста и мог найти определенный интересующий его пост через id этого поста (документ в базе данных Elasticsearch содержит не все данные поста. Пользователь может захотеть просмотреть дополнительные данные поста);

-authorId мы добавили, чтобы было возможно выполнять фразовый поиск именно для постов, которые принадлежат текущему пользователю (обычный пользователь может искать по фразовому поиску только свои посты).

-Мы не заложили в документ других данных поста, так как это не требуется. Мы добавили только те данные, которые действительно нужны

Для title и content мы установили тип text, так как именно такой тип имеют строки в базе данных Elasticsearch. Для id и authorId мы установили тип keyword – это специальный тип в базе данных Elasticsearch, который подходит для идентификаторов (если вдруг будет выполняться поиск по такой колонке, то там будет происходить поиск точного совпадения без учета морфологии слов или их порядка).

Также для title и content мы установили analyzer: 'russian'. Это нужно, так как наш фразовый поиск должен не только учитывать расстояние между запрашиваемыми словами. Фразовый поиск должен еще выполнять полнотекстовый поиск, то есть учитывать регистры и разные формы слова. Так как все посты – на русском языке, мы установили специальный анализатор russian, который будет решать данный вопрос. Без установления такого анализатора – фразовый поиск будет не способен выполнять полнотекстовый поиск.

После создания индекса, мы получаем все существующие посты и добавляем их в созданный индекс. Для этого мы определяем специальный массив docs и добавляем туда все посты следующим образом:



```
posts.forEach(post => {
  docs.push({
    index: {
      _index: postIndex,
      _id: post.id
    }
  });
  docs.push({
    id: post.id,
    title: post.title,
    content: post.content,
    authorId: post.authorId
  });
});

await esProvider.populateIndex(docs);
```

Сначала добавляется специальный объект, который содержит имя индекса и его id (которое будет равно id текущего поста):

```
docs.push({
  index: {
    _index: postIndex,
    _id: post.id
  }
});
```

Имя индекса мы добавляем, чтобы было понятно — в какой индекс добавляется документ. А id мы добавляем, чтобы в последствии можно было обращаться к документу индекса по id этого документа (если самому не указать id документа при его добавлении в индекс, то Elasticsearch автоматически добавит свой id документу. Мы не будем знать этот id. Соответственно, когда мы захотим получить какой-то документ из базы данных Elasticsearch, например, чтобы обновить его данные, или удалить его, мы не сможем его получить, так как мы не знаем его id. А когда мы указываем документу id, равный id поста документа, при его добавлении в индекс, мы легко сможем получить нужный нам документ, так как будем знать его id).

После мы добавляем объект, содержащий данные поста (id, title, content, authorId):

```
docs.push({
  id: post.id,
  title: post.title,
  content: post.content,
  authorId: post.authorId
});
```

И так для каждого существующего поста. Мы перед каждым постом в массиве добавляем тот объект конфигурации, чтобы указать методу, который будет добавлять данный пост в индекс, что мы хотим именно добавить в этот заданный индекс следующий элемент в массиве (то есть пост), а также установить ему данный id. После того, как все посты и все конфигурации под все посты будут добавлены в массив, мы все это добавим в созданный индекс через метод `populateIndex` провайдера `ElasticSearchProvider`.

Таким образом, мы создадим в базе данных `ElasticSearch` индекс под посты пользователя, а также добавим в этот индекс все существующие на данный момент посты. Теперь нужно вызвать эту функцию в главной точке входа в приложение `main.ts` (`./src/main.ts`):

```
await setupElasticSearch();
```

Внедрим функционал, позволяющий выполнять фразовый поиск. Для этого расширим функционал класса `PostService` (`./src/domain/post/post.service.ts`). Сначала добавим переменную, которая содержит индекс постов в базе данных `ElasticSearch`, а также инъектируем провайдер `ElasticSearchProvider`:

```
class PostService extends DomainService {
  private readonly esIndex: string = 'post_idx';
  private readonly esProvider: ElasticSearchProvider;
```

```
  constructor(esProvider: ElasticSearchProvider) {
    super();
    this.esProvider = esProvider;
  }
```

Определим метод, позволяющий получать индекс постов:

```
public getEsIndex(): string {  
    return this.esIndex;  
}
```

После создадим новый метод phraseSearch:

```
83     public async phraseSearch(  
84         user: IUserPayload,  
85         searchParam: 'title' | 'content',  
86         searchString: string  
87     ): Promise<unknown[]>  
88     {  
89         const matchPhrase = {  
90             [searchParam]: {  
91                 query: searchString,  
92                 slop: 1  
93             }  
94         };  
95     }  
96 }
```

```

96  ✓    const phraseSearchSettings: IPhraseSearch = {
97  ✓      admin: {
98          index: this.esIndex,
99  ✓      query: {
100         match_phrase: matchPhrase
101       }
102     },
103  ✓    user: {
104         index: this.esIndex,
105  ✓    query: {
106  ✓      bool: {
107  ✓        must: [
108  ✓          {
109             match_phrase: matchPhrase
110           },
111  ✓          {
112  ✓            term: {
113                 authorId: user.id
114             }
115           }
116        ]
117      }
118    },
119    _source_excludes: [ 'authorId' ]
120  }
121  };

```

```

122
123    const searchResult = await this.esProvider.searchByRequest(phraseSearchSettings[user.role]);
124    return searchResult.hits.hits.map(hit => hit._source);
125  }

```

Разберем данный метод. Метод принимает три параметра:

- user – данные текущего пользователя, который выполняет фразовый поиск
- searchParam – параметр, по которому осуществляется фразовый поиск (либо поле title (заголовки постов), либо поле content (текст постов))
- searchString – строка, по которой пользователь осуществляет фразовый поиск

Далее создаются конфигураторы – `matchPhrase` и `phraseSearchSettings`, на основе которых строится запрос фразового поиска. Конфигуратор `phraseSearchSettings` содержит конфигурацию поиска для админов и для обычных пользователей.

Рассмотрим конфигурацию для админов:

```
admin: {  
  index: this.esIndex,  
  query: {  
    match_phrase: matchPhrase  
  }  
},
```

-Здесь указывается индекс, в котором нужно вести поиск (в данном случае – это индекс постов в базе данных Elasticsearch, который мы ранее определили в поле `esIndex`). Также здесь мы описываем сам запрос в поле `query`. В данном случае – используется `match_phrase` запрос – тип запроса, который используется для поиска документов, содержащих указанную фразу. Данный запрос создается на основе определенного ранее конфигулятора `matchPhrase`:

```
const matchPhrase = {  
  [searchParam]: {  
    query: searchString,  
    slop: 1  
  }  
};
```

-Этот конфигулятор устанавливает поле, по которому будет происходить `match_phrase` запрос. В данном случае – это то поле, которое было передано в функцию в параметре `searchParam`. Также данный конфигулятор говорит, что поиск должен проходить по переданной в функцию строке `searchString` (`query: searchString`), а также между запрашиваемыми словами в этой строке либо вообще нет других слов, либо есть только одно слово (`slop: 1`).

-Таким образом, мы получим `match_phrase` запрос, который будет искать указанную строку, где слова в указанной строке должны идти в том же порядке, в котором определены в указанной строке, а также расстояние между запрашиваемыми словами – не более одного слова.

Теперь рассмотрим конфигурацию для обычных пользователей:

```

user: {
  index: this.esIndex,
  query: {
    bool: {
      must: [
        {
          match_phrase: matchPhrase
        },
        {
          term: {
            authorId: user.id
          }
        }
      ]
    }
  },
  _source_excludes: [ 'authorId' ]
}

```

-Здесь также указывается индекс, в котором нужно вести поиск (в данном случае — это индекс постов в базе данных Elasticsearch, который мы ранее определили в поле `esIndex`). Также создается запрос на основе критериев, определенных в `query`. Мы указали `bool`-запрос, чтобы можно было выполнить несколько критериев поиска, а `must` является как-раз массивом всех условий, которые нужно выполнить. В данном случае здесь два условия:

- Используется такой же `match_phrase` запрос, как в конфигурации для админов
- Используется `term` запрос, который ищет документы, содержащие точное значение для указанного поля. В данном случае — ищем документы, где `authorId` равен `id` текущего пользователя, который в данный момент выполняет фразовый поиск. Благодаря этому — пользователь может выполнять фразовый поиск только среди своих постов, и не может получить посты другого пользователя

Также мы указываем `_source_excludes` — это параметр, который указывает поля, которые следует исключить из возвращаемых документов. В данном случае, мы из всех полученных результатов исключаем поле `authorId`, так как обычный пользователь и так получит в результате только свои посты, и ему не нужно знать свой собственный `id` в системе.

В конце мы выполняем, созданный конфигураторами, запрос к базе данных Elasticsearch через описанного ранее провайдера `ElasticSearchProvider`:

```
const searchResult = await this.esProvider.searchByRequest(phraseSearchSettings[user.role]);
```

То есть мы получаем из конфигуратора phraseSearchSettings соответствующий запрос для фразового поиска, передав в него роль текущего пользователя.

Далее мы просто извлекаем документы из полученного ответа от Elasticsearch и возвращаем массив полученных документов:

```
return searchResult.hits.hits.map(hit => hit._source);
```

Не будем забывать, что пользователь может создать новый пост, изменить существующий, удалить старый. Теперь все эти изменения нужно вносить также и в базу данных Elasticsearch. Для этого используем ElasticsearchProvider:

- В методе createPost будем добавлять новый созданный пост не только в нашу базу данных, но и в базу данных Elasticsearch:

```
public async createUserPost(user: IUserPayload, postDataCreate: IPostCreate): Promise<Post> {
    const newPost = await Post.create({
        ...postDataCreate
    });

    await this.esProvider.indexDocument(this.esIndex, newPost.id, {
        id: newPost.id,
        title: newPost.title,
        content: newPost.content,
        authorId: newPost.authorId
    });

    return (
        await this.getPostByUniqueParams({
            where: {
                id: newPost.id
            }
        }, user)
    );
}
```

- В методе updatePost будем проверять – обновляет ли пользователь title и content поста. Если обновляет, то обновим эти данные в базе данных Elasticsearch:

```

173     public async updateUserPost(post: Post, newPostData: IPostUpdate): Promise<Post> {
174         if (newPostData.title) {
175             await this.esProvider.updateDocument(this.esIndex, post.id, {
176                 title: newPostData.title
177             });
178         }
179
180         if (newPostData.content) {
181             await this.esProvider.updateDocument(this.esIndex, post.id, {
182                 content: newPostData.content
183             });
184         }
185
186         Object.assign(post, newPostData);
187         await post.save();
188         return post;
189     }

```

- В методе deletePost будем удалять пост не только в нашей базе данных, но и в базе данных ElasticSearch:

```

public async deleteUserPost(post: Post): Promise<void> {
    await this.esProvider.deleteDocument(this.esIndex, post.id);
    await post.destroy();
}

```

Теперь нужно изменить контроллер PostController (./src/domain/post/post.controller.ts), так как именно он работает с PostService. В данном контроллере за фразовый поиск отвечает метод getAllPosts. Значит его и изменим:

```

@ValidateReqParam('searchParam', PostSearchParamSchema)
public async getAllUserPosts(req: Request, res: Response, next: NextFunction): Promise<Response | void> {
    try {
        const user = req.user as IUserPayload;
        const searchParam = (req.query.searchParam as 'title' | 'content') || 'title';
        const searchString = req.query.searchString as string;

        let posts = [];

        if (!searchString) {
            posts = await this.postService.getAllUserPosts({}, user);
        } else {
            posts = await this.postService.phraseSearch(user, searchParam, searchString);
        }

        return res.status(200).json({ status: 200, data: posts, message: "List of all posts" });
    } catch (err) {
        next(err);
    }
}

```



Разберем изменения:

- Теперь мы проводим валидацию параметра `searchParam` за счет декоратора `ValidateReqParam` по схеме `PostSearchParamSchema` (`./src/domain/post/validation/schema/post.search.schema.ts`):

```
src > domain > post > validation > schema > TS post.search.schema.ts > ...
1  import Joi, { Schema } from 'joi';
2
3  const PostSearchParamSchema: Schema<string> = Joi.string().valid('title', 'content').optional();
4  export default PostSearchParamSchema;
5  |
```

-То есть валидация будет заключаться в том, чтобы проверить что пользователь передал в `searchParam`. Если в `searchParam` будет передано что-то отличное от 'title', 'content', undefined, то будет ошибка валидации. Таким образом, мы разрешаем пользователю проводить фразовый поиск только по столбцам 'title' или 'content', или же вообще не проводить поиск (undefined)

- Теперь, в `searchParam` будет записано значение по умолчанию 'title', если пользователь ничего не передаст. Это нужно для случая, когда пользователь захотел выполнить фразовый поиск (указал `searchString`), но не указал `searchParam`. В таком случае – по умолчанию фразовый поиск будет проходить по столбцу `title`
- Теперь вызывается метод `phraseSearch`, определенной ранее в `PostService`, если пользователь захочет провести фразовый поиск

## Тестирование:

### Запуск экземпляра Elasticsearch в контейнере Docker:

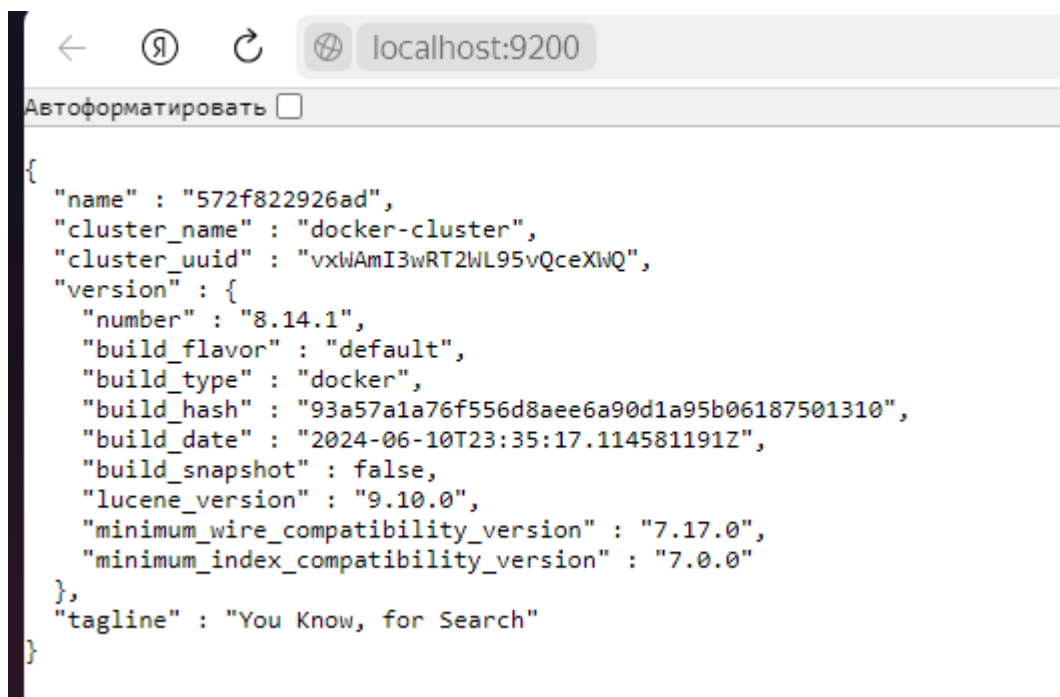
Сначала запустим контейнер `ElasticSearch`. Для этого выполним команду `npm run start:compose` в корне нашего приложения:

```
PS C:\Users\Богдан\Desktop\Консорцеум Кодекс\Стажировка\daccord> npm run start:compose
```

```
time="2024-07-30T19:27:15+03:00" level=warning msg="C:\\Users\\Богдан\\Desktop\\Консорцеум Кодекс\\Стажировка\\daccord\\docker-compose.yml: 'version' is obsolete"
[+] Running 1/0
✓ Container elasticsearch_container Created 0.0s
Attaching to elasticsearch_container
elasticsearch_container | Jul 30, 2024 4:27:19 PM sun.util.locale.provider.LocaleProviderAdapter <clinit>
elasticsearch_container | WARNING: COMPAT locale provider will be removed in a future release
elasticsearch_container | {"@timestamp":"2024-07-30T16:27:20.319Z", "log.level": "INFO", "message": "Using [jdk] native provider and native methods for [Linux]", "ecs.version": "1.2.0", "service.name": "ES_ECS", "event.dataset": "elasticsearch.server", "process.thread.name": "main", "log.logger": "org.elasticsearch.nativeaccess.NativeAccess", "elasticsearch.node.name": "572f822926ad", "elasticsearch.cluster.name": "docker-cluster"}
elasticsearch_container | {"@timestamp":"2024-07-30T16:27:20.663Z", "log.level": "INFO", "message": "Java vector incubator API enabled; uses preferredBitSize=256; FMA enabled", "ecs.version": "1.2.0", "service.name": "ES_ECS", "event.dataset": "elasticsearch.server", "process.thread.name": "main", "log.logger": "org.apache.lucene.internal.vectorization.PanamaVectorizationProvider", "elasticsearch.node.name": "572f822926ad", "elasticsearch.cluster.name": "docker-cluster"}
```

```
QfOqpqvjAFwnXw", "elasticsearch.node.name": "572f822926ad", "elasticsearch.cluster.name": "docker-cluster")
elasticsearch_container | {"@timestamp": "2024-07-30T16:27:32.081Z", "log.level": "INFO", "message": "Node [{572f82
2926ad}{cmHfNz6KQfOqpqvjAFwnXw}] is selected as the current health node.", "ecs.version": "1.2.0", "service.name": "
ES_ECS", "event.dataset": "elasticsearch.server", "process.thread.name": "elasticsearch[572f822926ad] [management] [T#3]
", "log.logger": "org.elasticsearch.health.node.selection.HealthNodeTaskExecutor", "elasticsearch.cluster.uuid": "vxWA
mI3wRT2WL95vQceXWQ", "elasticsearch.node.id": "cmHfNz6KQfOqpqvjAFwnXw", "elasticsearch.node.name": "572f822926ad", "ela
sticsearch.cluster.name": "docker-cluster"}
elasticsearch_container | {"@timestamp": "2024-07-30T16:27:32.118Z", "log.level": "INFO", "current.health": "YELLOW
W", "message": "Cluster health status changed from [RED] to [YELLOW] (reason: [shards started [[post_idx][0]]]).", "p
revious.health": "RED", "reason": "shards started [[post_idx][0]]", "ecs.version": "1.2.0", "service.name": "ES_ECS", "
event.dataset": "elasticsearch.server", "process.thread.name": "elasticsearch[572f822926ad] [masterService#updateTask]
[T#1]", "log.logger": "org.elasticsearch.cluster.routing.allocation.AllocationService", "elasticsearch.cluster.uuid":
"vxWAmI3wRT2WL95vQceXWQ", "elasticsearch.node.id": "cmHfNz6KQfOqpqvjAFwnXw", "elasticsearch.node.name": "572f822926ad"
, "elasticsearch.cluster.name": "docker-cluster"}
View in Docker Desktop View Config Enable Watch
```

Проверим, что порт 9200 отвечает:



```
{
  "name" : "572f822926ad",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "vxWAmI3wRT2WL95vQceXWQ",
  "version" : {
    "number" : "8.14.1",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "93a57a1a76f556d8aee6a90d1a95b06187501310",
    "build_date" : "2024-06-10T23:35:17.114581191Z",
    "build_snapshot" : false,
    "lucene_version" : "9.10.0",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

Экземпляр Elasticsearch в контейнере Docker был успешно запущен.

### Запуск приложения:

После запуска сервера Elasticsearch – можно запустить само приложение. Для этого откроем в корне проекта новый терминал и введем туда `npm run start:dev`:

```
PS C:\Users\Вордан\Desktop\Консорцеум Кодекс\Стажировка\daccord> npm run start:dev
```

```
{
  event: 'reverted',
  name: '20240719151942-add_text_tsv_column_to_post_table.js',
  durationSeconds: 0.022
}
Executing (default): SELECT "id", "name", "role", "email", "password", "isActivated", "refreshToken", "verifToken"
, "createdAt", "updatedAt", "rating" FROM "Users" AS "User" WHERE "User"."isActivated" = false;
Executing (default): SELECT "id", "name", "role", "email", "password", "isActivated", "refreshToken", "verifToken"
, "createdAt", "updatedAt", "rating" FROM "Users" AS "User" WHERE "User"."isActivated" = true AND "User"."verifTok
en" IS NOT NULL;
Server running on http://localhost:5000 - development
```

Сервер был успешно запущен. Также миграция для полнотекстового поиска успешно была отменена.

Дальше проведем серию тестов через Swagger (<http://localhost:5000/api>).

## Авторизация:

Сначала авторизируемся в системе:

auth

GET /auth/payload Get a user payload

POST /auth/signin Login to account

Parameters

No parameters

Request body <sup>required</sup>

application/json

Examples:

[Modified value]

```
{
  "email": "awesom-e4000@mail.ru",
  "password": "jdHGDjg7dHq?!!!"
}
```

Execute Clear

Responses

### Response body

```
{
  "status": 200,
  "data": {
    "name": "jason_statham",
    "email": "awesom-e4000@mail.ru"
  },
  "message": "Authorization was successful"
}
```

Download

### Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 118
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 16:37:38 GMT
etag: W/"76-KfCoMwHX41/nDuSn4D1uF31D9So"
keep-alive: timeout=5
x-powered-by: Express
```

Мы успешно авторизовались в системе как обычный пользователь.

**Получение всех постов пользователя:**

Сначала получим все текущие посты пользователя (чтобы понимать далее при фразовом поиске, какие посты мы должны получать). Для этого выполним метод GET /api/posts/public без передачи параметров:

posts

GET /api/posts/public Get all user posts

Parameters

Name	Description
searchParam	Search parameter that sets among whom the search substring will be searched - titles or contents
string (query)	--
searchString	String that is searched among titles or contents
string (query)	searchString

Execute

Responses

Responses

Curl

curl -X 'GET' \n http://localhost:5000/api/posts/public' \n -H 'accept: application/json'

Request URL

http://localhost:5000/api/posts/public

Server response

Code

Details

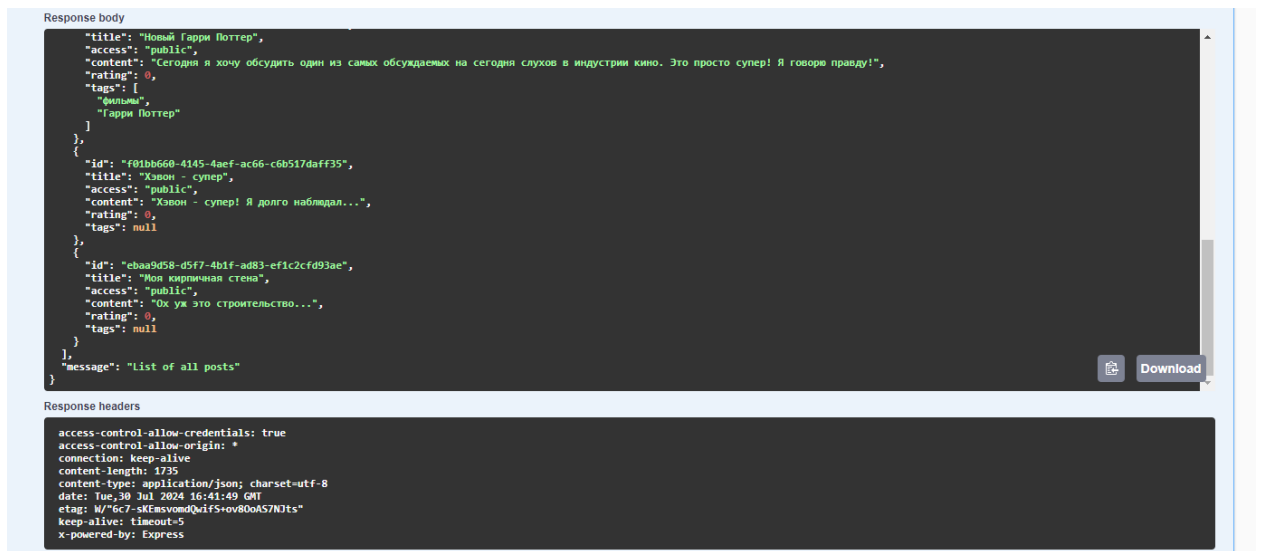
200

Response body

```
{
  "status": 200,
  "data": [
    {
      "id": "cfdbf3b2-6a16-4e02-a94b-c3d17a1d33dd",
      "title": "Кирпичная дорогая измазанная стена",
      "access": "public",
      "content": "Я покажу вам сегодня...",
      "rating": 0,
      "tags": null
    },
    {
      "id": "4735b21f-ab56-4ffd-bc2f-d8298771465d",
      "title": "Мои кирпичные стены",
      "access": "public",
      "content": "Расскажу об...",
      "rating": 0,
      "tags": null
    },
    {
      "id": "5c805f58-ba75-48d4-9278-762531e23fa5",
      "title": "Кирпичная дорогая стена",
      "access": "public",
      "content": "Хочу установить...",
      "rating": 0,
      "tags": null
    }
  ]
}
```

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 1735
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 16:41:49 GMT
etag: W/"6c7-5KEasvondQwif5+ov80a57Njts"
keep-alive: timeout=5
x-powered-by: Express
```



Таким образом, мы получили следующие посты:

```
{
  "id": "cfdfb3b2-6a16-4e02-a94b-c3d17a1d33dd",
  "title": "Кирпичная дорогая изящная стена",
  "access": "public",
  "content": "Я покажу вам сегодня...",
  "rating": 0,
  "tags": null
},
{
  "id": "4735b21f-ab56-4ffd-bc2f-d8298771465d",
  "title": "Мои кирпичные стены",
  "access": "public",
  "content": "Расскажу об...",
  "rating": 0,
  "tags": null
},
{
  "id": "5c805f58-ba75-48dd-9278-762531e23fa5",
  "title": "Кирпичная дорогая стена",
  "access": "public",
  "content": "Хочу установить..."
}
```

```
"rating": 0,
"tags": null
},
{
  "id": "43a6c97b-7a97-41a2-8076-77b97bce4d9f",
  "title": "NMIXX - очень перспективная группа",
  "access": "public",
  "content": "Я влюбился в девочек из NMIXX. Слушая их музыку, мне хочется обнять каждую...",
  "rating": 0,
  "tags": [
    "музыка",
    "кpop",
    "nmixx"
  ]
},
{
  "id": "f684a441-e88c-4cb8-8aff-5b452bd6dbe5",
  "title": "Новый Гарри Поттер",
  "access": "public",
  "content": "Сегодня я хочу обсудить один из самых обсуждаемых на сегодня слухов в индустрии кино. Это просто супер! Я говорю правду!",
  "rating": 0,
  "tags": [
    "фильмы",
    "Гарри Поттер"
  ]
},
{
  "id": "f01bb660-4145-4aef-ac66-c6b517daff35",
  "title": "Хэвон - супер",
  "access": "public",
  "content": "Хэвон - супер! Я долго наблюдал..."
```

```
"rating": 0,
"tags": null
},
{
  "id": "ebaa9d58-d5f7-4b1f-ad83-ef1c2cf93ae",
  "title": "Моя кирпичная стена",
  "access": "public",
  "content": "Ох уж это строительство...",
  "rating": 0,
  "tags": null
}
```

Именно на эти посты мы будем ориентироваться при фразовом поиске.

### **Фразовый поиск по title:**

Выполним фразовый поиск фразы ‘кирпичная стена’ по столбцу title. Исходя из существующих постов пользователя, нам должны вернуться следующие посты:

```
{
  "id": "4735b21f-ab56-4ffd-bc2f-d8298771465d",
  "title": "Мои кирпичные стены",
  "access": "public",
  "content": "Расскажу об...",
  "rating": 0,
  "tags": null
},
{
  "id": "5c805f58-ba75-48dd-9278-762531e23fa5",
  "title": "Кирпичная дорогая стена",
  "access": "public",
  "content": "Хочу установить...",
  "rating": 0,
```

```

    "tags": null
  },
{
  "id": "ebaa9d58-d5f7-4b1f-ad83-ef1c2cfd93ae",
  "title": "Моя кирпичная стена",
  "access": "public",
  "content": "Ох уж это строительство...",
  "rating": 0,
  "tags": null
}

```

Для этого выполним GET /api/posts/public, передав в качестве параметров – searchParam='title', searchString='кирпичная стена':

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** /api/posts/public
- Parameters:**
  - searchParam:** title (dropdown menu)
  - searchString:** кирпичная стена (text input)
- Buttons:** Execute, Clear
- Responses:** (Empty section)

The screenshot shows the response of the GET request to /api/posts/public. The response is a JSON object with the following structure:

```

{
  "status": 200,
  "data": [
    {
      "id": "ebaa9d58-d5f7-4b1f-ad83-ef1c2cfd93ae",
      "title": "Моя кирпичная стена",
      "content": "Ох уж это строительство..."
    },
    {
      "id": "4735b21f-ab56-4ffd-bc2f-d8298771465d",
      "title": "Мои кирпичные стены",
      "content": "Рассказы об..."
    },
    {
      "id": "5c805f58-ba75-48dd-9278-762531e23fa5",
      "title": "Кирпичная дорожная стена",
      "content": "Новую установить..."
    }
  ],
  "message": "List of all posts"
}

```

The response headers are also visible:

```

access-control-allow-credentials: true
access-control-allow-origin: *
content-length: 481
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 16:49:09 GMT
etag: W/"1e1-MlnTkcTzS/JaJmslzdIAPERMEa"
x-powered-by: Express

```



Метод правильно вернул все посты. Это значит, что индекс в базе данных успешно был создан, а также все посты были успешно добавлены в данный индекс. Также это означает, что фразовый поиск правильно учитывает расстояние между словами и правильно учитывает разные формы слова и регистр.

### Создание нового поста:

Теперь убедимся, что при создании нового поста – он будет добавлен не только в нашу базу данных, но и в базу данных Elasticsearch. Создадим такой пост:

```
{  
  "title": "Мои кирпичные дорогие стены",  
  "access": "public",  
  "content": "Я знаю один секрет...",  
  "tags": [  
    "строительство"  
  ]  
}
```

Для этого выполним метод POST /api/posts/public и зададим этот пост:

The screenshot shows a REST client interface with the following components:

- Header:** "posts" with a dropdown arrow.
- Method and URL:** "POST /api/posts/public" with a description "Create a new post".
- Parameters:** A section labeled "Parameters" with "No parameters" listed. It includes "Cancel" and "Reset" buttons.
- Request body:** A section labeled "Request body" with a "required" status and a dropdown menu set to "application/json".
- Examples:** A dropdown menu labeled "Examples:" with "[Modified value]" selected.
- JSON Body:** A text area containing the following JSON:

```
{  
  "title": "Мои кирпичные дорогие стены",  
  "access": "public",  
  "content": "Я знаю один секрет...",  
  "tags": [  
    "строительство"  
  ]  
}
```
- Execute:** A large blue button labeled "Execute".
- Responses:** A section labeled "Responses" at the bottom.

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:5000/api/posts/public' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "title": "Мои кирпичные дорожные стены",
    "access": "public",
    "content": "Я знаю один секрет...",
    "tags": [
      "строительство"
    ]
  }'
```

Request URL

http://localhost:5000/api/posts/public

Server response

Code Details

201

Response body

```
{
  "status": 201,
  "data": {
    "id": "e7e70286-e233-42e9-a575-bd78df9da39f",
    "title": "Мои кирпичные дорожные стены",
    "access": "public",
    "content": "Я знаю один секрет...",
    "rating": 0,
    "tags": [
      "строительство"
    ]
  },
  "message": "Post successfully created"
}
```

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 283
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 16:55:39 GMT
etag: W/"11b-3XA93yJdesNIP5js3Y1R7luNkMu"
keep-alive: timeout=5
location: /api/posts/e7e70286-e233-42e9-a575-bd78df9da39f
x-powered-by: Express
```

Новый пост успешно создан. Теперь проверим – был ли он добавлен в базу данных ElasticSearch. Для этого выполним метод GET /api/posts/public, передав параметры: searchParam='title', searchString='кирпичная стена':

GET /api/posts/public Get all user posts

Parameters

Cancel

Name	Description
searchParam	Search parameter that sets among whom the search substring will be searched - titles or contents
string	
(query)	<input type="text" value="title"/>
searchString	String that is searched among titles or contents
string	
(query)	<input type="text" value="кирпичная стена"/>

Execute Clear

Responses

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5000/api/posts/public?searchParam=title&searchString=ND0NBAND0NB8ND1X80ND0NB8FND0NB8ND1X87ND0NB8ND0NB8ND1X8FND0NB8ND1X81ND1X82ND0NB85ND0NB8ND0NB80' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/api/posts/public?searchParam=title&searchString=ND0NBAND0NB8ND1X80ND0NB8FND0NB8ND1X87ND0NB8ND0NB8ND1X8FND0NB8ND1X81ND1X82ND0NB85ND0NB8ND0NB80
```

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "status": 200,   "data": [     {       "id": "ebaa9658-d5f7-4b1f-ad83-ef1c2cf93ae",       "title": "Мои кирпичная стена",       "content": "Ох уж это строительство..."     },     {       "id": "4735b21f-ab56-4ffd-bc2f-d829871465d",       "title": "Мои кирпичные стены",       "content": "Расскажу об..."     },     {       "id": "5c805f58-ba75-48d4-9278-762531e23fa5",       "title": "Кирпичная дорогая стена",       "content": "Хочу установить..."     },     {       "id": "e7e70286-e233-42e9-a575-bd78df9da39f",       "title": "Мои кирпичные дорогие стены",       "content": "Я знаю один секрет..."     }   ],   "message": "List of all posts" }</pre></div><div><div>Download</div></div></div>

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 638
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 16:57:39 GMT
etag: W/"22e-a3/xyJprW0mkvsaocJC83R5PHQ"
keep-alive: timeout=5
x-powered-by: Express
```

Новый созданный пост был успешно найден. Это значит, что после создания поста – он не только записывается в нашу базу данных, но и записывается в базу данных ElasticSearch.

**Изменение старого поста:**

Изменим существующий пост пользователя, чтобы убедиться, что после изменения поста в нашей базе данных – он также изменяется в базе данных ElasticSearch. Изменим заголовок, созданного на предыдущем этапе, поста следующим образом:

title: “Мои кирпичные дорогие изящные стены”

Изменим этот пост с помощью метода PATCH /api/posts/public/{postId}, указав id данного поста:

PATCH

/api/posts/public/{postId}

Update a post by id

Parameters

Cancel

Reset

Name	Description
postId * required	Post id. Example: 550e8400-e29b-41d4-a716-446655440000
string (path)	e7e70286-e233-42e9-a575-bd78df9da39f

Request body required

application/json

Examples:

[Modified value]

```
{
  "title": "Мои кирпичные дорожки издают стелы"
}
```

Execute

Responses

Curl

curl -X 'PATCH' \
 'http://localhost:5000/api/posts/public/e7e70286-e233-42e9-a575-bd78df9da39f' \
 -H 'accept: application/json' \
 -H 'Content-Type: application/json' \
 -d '{
 "title": "Мои кирпичные дорожки издают стелы"
 }'

Request URL

http://localhost:5000/api/posts/public/e7e70286-e233-42e9-a575-bd78df9da39f

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "status": 200,   "data": {     "id": "e7e70286-e233-42e9-a575-bd78df9da39f",     "title": "Мои кирпичные дорожки издают стелы",     "access": "public",     "content": "Я знаю один секрет...",     "rating": 0,     "tags": [       "строительство"     ],     "updatedAt": "2024-07-30T17:03:07.189Z"   },   "message": "Post successfully updated" }</pre></div><div>Download</div></div>

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 337
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 17:03:07 GMT
etag: W/"151-1204A2F7CB/1135y0vqndowFCY"
keep-alive: timeout=5
x-powered-by: Express
```

Данный пост был успешно изменен. Убедимся в этом, получив все посты пользователя через метод GET /api/posts/public, не передавая никаких параметров:

**posts**

GET /api/posts/public Get all user posts

Parameters

Name	Description
searchParam string (query)	Search parameter that sets among whom the search substring will be searched - titles or contents <input type="text" value="--"/>
searchString string (query)	String that is searched among titles or contents <input type="text" value="searchString"/>

Execute Clear

200

Response body

```
{
  "status": 200,
  "data": [
    {
      "id": "e7e70286-e233-42e9-a575-bd78df9da39f",
      "title": "Мои кирпичные дорогие изящные стены",
      "access": "public",
      "content": "Я знаю один секрет...",
      "rating": 0,
      "tags": [
        "строительство"
      ]
    }
  ],
}
```

Как можно заметить – действительно данный пост был обновлен в нашей базе данных. Теперь убедимся, что данный пост был обновлен в базе данных Elasticsearch. Для этого выполним этот же метод, передав параметры: searchParam='title', searchString='кирпичная стена'. Если данный пост действительно был обновлен в базе данных, то при таком фразовом поиске – мы не должны его получить, так как между запрашиваемыми словами 'кирпичная' и 'стена' в том посте находятся два слова: 'дорогие' и 'изящные', а по условию – расстояние не больше одного слова:

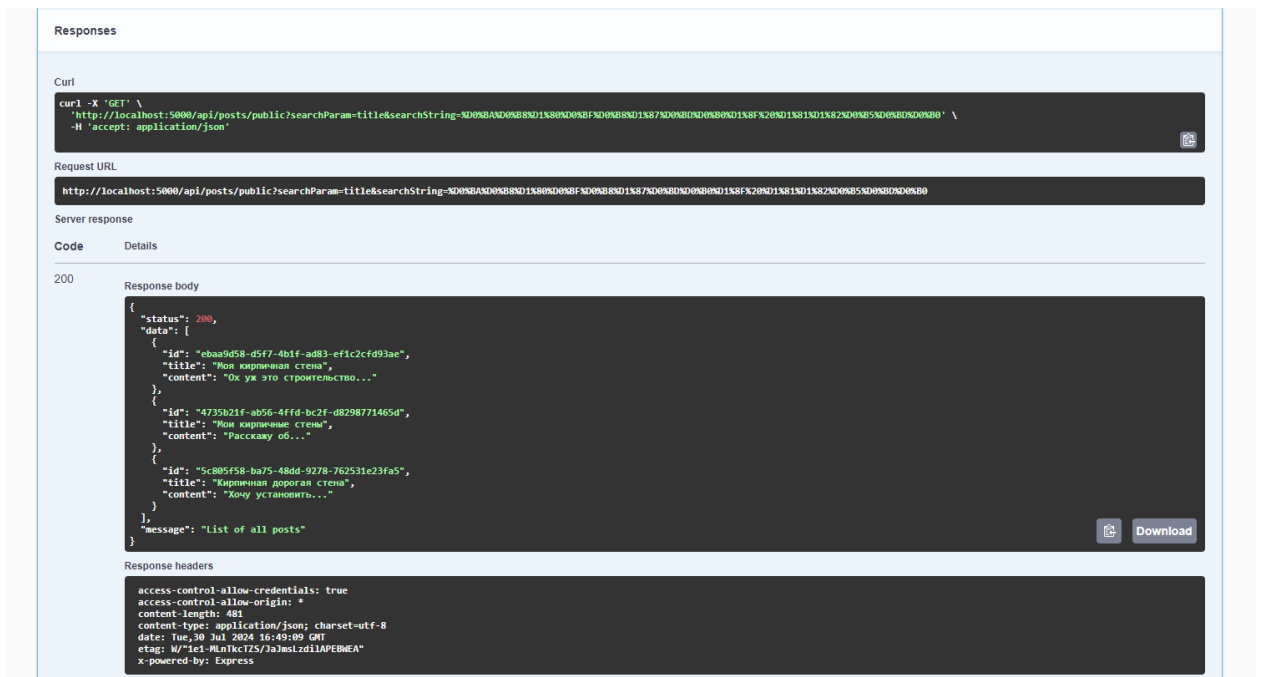
**posts**

GET /api/posts/public Get all user posts

Parameters

Name	Description
searchParam string (query)	Search parameter that sets among whom the search substring will be searched - titles or contents <input type="text" value="title"/>
searchString string (query)	String that is searched among titles or contents <input type="text" value="кирпичная стена"/>

Execute Clear



Как можно заметить – мы не получили данный пост. Значит он успешно был обновлен в базе данных Elasticsearch.

### Удаление существующего поста:

Теперь убедимся, что после удаления поста из нашей базы данных – он также удаляется и в базе данных Elasticsearch. Для этого удалим данный пост:

```
{
  "id": "ebaa9d58-d5f7-4b1f-ad83-ef1c2cfd93ae",
  "title": "Моя кирпичная стена",
  "access": "public",
  "content": "Ох уж это строительство...",
  "rating": 0,
  "tags": null
}
```

Воспользуемся методом DELETE /api/posts/public/{postId}, передав id данного поста:

DELETE

/api/posts/public/{postId}

Delete a post by id

Parameters

Cancel

Name	Description
postId	Post Id. Example: 550e8400-e29b-41d4-a716-446655440000
string	
(path)	ebaa9d58-d5f7-4b1f-ad83-ef1c2cf093ae

Execute

Responses

Curl

curl -X 'DELETE' \

'http://localhost:5000/api/posts/public/ebaa9d58-d5f7-4b1f-ad83-ef1c2cf093ae' \

-H 'accept: \*/\*'

Request URL

http://localhost:5000/api/posts/public/ebaa9d58-d5f7-4b1f-ad83-ef1c2cf093ae

Server response

Code	Details
200	<div>Response body</div> <div><pre>{   "status": 200,   "message": "Post successfully deleted" }</pre></div> <div>Download</div>

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 52
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 17:51:54 GMT
etag: W/"34-vdWQo+7dj29IdAImyKzuksAd678"
keep-alive: timeout=5
x-powered-by: Express
```

Пост был успешно удален.

Теперь нужно убедиться, что он был также удален из базы данных ElsticSearch. Для этого выполним метод GET /api/posts/public, передав туда параметры: searchParam='title', searchString='кирпичная стена'. Если пост был успешно удален из базы данных ElsticSearch, то мы не должны получить его в результатах:

posts

GET

/api/posts/public

Get all user posts

Parameters

Cancel

Name	Description
searchParam	Search parameter that sets among whom the search substring will be searched - titles or contents
string	
(query)	title
searchString	String that is searched among titles or contents
string	
(query)	кирпичная стена

Execute

Clear





```

"title": "Хэвон - супер",
"access": "public",
"content": "Хэвон - супер! Я долго наблюдал...",
"rating": 0,
"tags": null
}

```

Для этого выполним метод GET /api/posts/public, передав параметры: searchParam='content', searchString='супер':

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** /api/posts/public
- Parameters:**
  - searchParam:** content (selected from a dropdown menu)
  - searchString:** супер (entered in a text field)
- Buttons:** Execute, Clear, Cancel
- Responses:**
  - Code:** 200
  - Response body:**

```

{
  "status": 200,
  "data": [
    {
      "id": "f01bb668-4145-4aef-ac66-c0b517daff35",
      "title": "Хэвон - супер",
      "content": "Хэвон - супер! Я долго наблюдал..."
    },
    {
      "id": "f684a441-e88c-4cb8-8aff-5b452bd6be5",
      "title": "Новый Гарри Поттер",
      "content": "Сегодня я хочу обсудить один из самых обсуждаемых на сегодня слухов в индустрии кино. Это просто супер! Я говорю правду!"
    }
  ],
  "message": "List of all posts"
}

```
  - Response headers:**

```

access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 526
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 17:26:15 GMT
etag: W/"20c-k6Hk3Jb1/XIDPnToudIfpZqGdE"
keep-alive: timeout=5
x-powered-by: Express

```

Мы действительно получили правильные посты.

### Проверка на сохранение документов в индексе при перезапуске Docker:

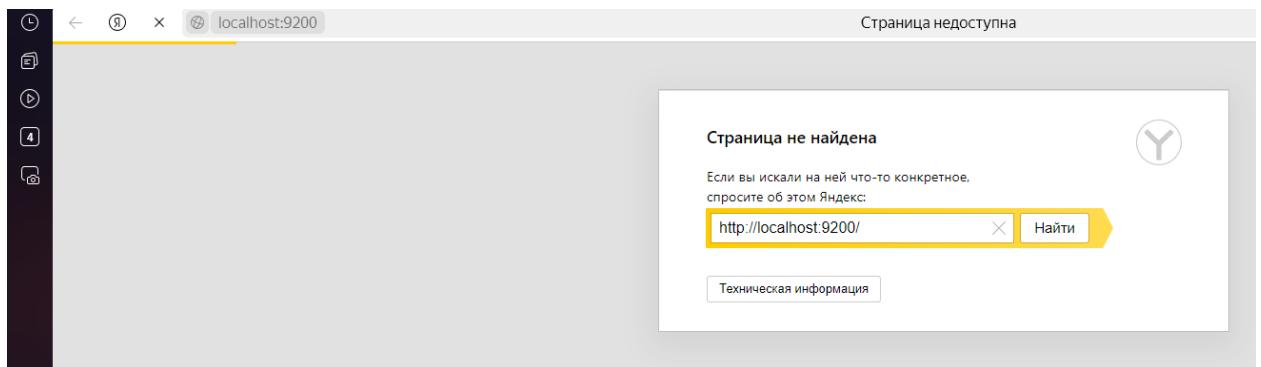
Убедимся, что данные действительно сохраняются, если перезапустить Docker. Для этого остановим работу Docker:

```

2926ad}{cmHfNz6KQfOqpqvjAFwnXw}} is selected as the current health node.", "ecs.version": "1.2.0", "service.name": "
ES_ECS", "event.dataset": "elasticsearch.server", "process.thread.name": "elasticsearch[572f822926ad] [management] [T#3]
", "log.logger": "org.elasticsearch.health.node.selection.HealthNodeTaskExecutor", "elasticsearch.cluster.uuid": "vxWA
mI3wRT2WL95vQceXWQ", "elasticsearch.node.id": "cmHfNz6KQfOqpqvjAFwnXw", "elasticsearch.node.name": "572f822926ad", "ela
sticsearch.cluster.name": "docker-cluster"}
elasticsearch_container | {"@timestamp": "2024-07-30T16:27:32.118Z", "log.level": "INFO", "current.health": "YELLO
W", "message": "Cluster health status changed from [RED] to [YELLOW] (reason: [shards started [[post_idx][0]])", "p
revious.health": "RED", "reason": "shards started [[post_idx][0]]", "ecs.version": "1.2.0", "service.name": "ES_ECS", "
event.dataset": "elasticsearch.server", "process.thread.name": "elasticsearch[572f822926ad] [masterService#updateTask]
[T#1]", "log.logger": "org.elasticsearch.cluster.routing.allocation.AllocationService", "elasticsearch.cluster.uuid":
"vxWAmI3wRT2WL95vQceXWQ", "elasticsearch.node.id": "cmHfNz6KQfOqpqvjAFwnXw", "elasticsearch.node.name": "572f822926ad"
, "elasticsearch.cluster.name": "docker-cluster"}
Gracefully stopping... (press Ctrl+C again to force)
[+] Stopping 1/1
✓ Container elasticsearch_container Stopped 2.6s
^C^CЗавершить выполнение пакетного файла [Y(да)/N(нет)]? y
PS C:\Users\Богдан\Desktop\Консорцеум Кодекс\Стажировка\dacord>

```

Перейдем по пути <http://localhost:9200>, чтобы убедиться, что порт 9200 перестал отвечать:



Порт 9200 перестал отвечать. Теперь остановим также наше приложение, а после перезапустим Docker и приложение таким же способом, как в начале тестов:

```

time="2024-07-30T20:47:07+03:00" level=warning msg="C:\\Users\\Богдан\\Desktop\\Консорцеум Кодекс\\Стажировка\\dac
cord\\docker-compose.yaml: 'version' is obsolete"
[+] Running 1/0
✓ Container elasticsearch_container Created 0.0s
Attaching to elasticsearch_container
elasticsearch_container | Jul 30, 2024 5:47:10 PM sun.util.locale.provider.LocaleProviderAdapter <clinit>
elasticsearch_container | WARNING: COMPAT locale provider will be removed in a future release
elasticsearch_container | {"@timestamp": "2024-07-30T17:47:11.470Z", "log.level": "INFO", "message": "Using [jdk] n
ative provider and native methods for [Linux]", "ecs.version": "1.2.0", "service.name": "ES_ECS", "event.dataset": "el
asticsearch.server", "process.thread.name": "main", "log.logger": "org.elasticsearch.nativeaccess.NativeAccess", "elast
icsearch.node.name": "572f822926ad", "elasticsearch.cluster.name": "docker-cluster"}
elasticsearch_container | {"@timestamp": "2024-07-30T17:47:11.847Z", "log.level": "INFO", "message": "Java vector i

```

```

QfOqpqvjAFwnXw", "elasticsearch.node.name": "572f822926ad", "elasticsearch.cluster.name": "docker-cluster"}
elasticsearch_container | {"@timestamp": "2024-07-30T17:47:23.268Z", "log.level": "INFO", "message": "Node [{572f82
2926ad}{cmHfNz6KQfOqpqvjAFwnXw}} is selected as the current health node.", "ecs.version": "1.2.0", "service.name": "
ES_ECS", "event.dataset": "elasticsearch.server", "process.thread.name": "elasticsearch[572f822926ad] [management] [T#3]
", "log.logger": "org.elasticsearch.health.node.selection.HealthNodeTaskExecutor", "elasticsearch.cluster.uuid": "vxWA
mI3wRT2WL95vQceXWQ", "elasticsearch.node.id": "cmHfNz6KQfOqpqvjAFwnXw", "elasticsearch.node.name": "572f822926ad", "ela
sticsearch.cluster.name": "docker-cluster"}
elasticsearch_container | {"@timestamp": "2024-07-30T17:47:23.298Z", "log.level": "INFO", "current.health": "YELLO
W", "message": "Cluster health status changed from [RED] to [YELLOW] (reason: [shards started [[post_idx][0]])", "p
revious.health": "RED", "reason": "shards started [[post_idx][0]]", "ecs.version": "1.2.0", "service.name": "ES_ECS", "
event.dataset": "elasticsearch.server", "process.thread.name": "elasticsearch[572f822926ad] [masterService#updateTask]
[T#1]", "log.logger": "org.elasticsearch.cluster.routing.allocation.AllocationService", "elasticsearch.cluster.uuid":
"vxWAmI3wRT2WL95vQceXWQ", "elasticsearch.node.id": "cmHfNz6KQfOqpqvjAFwnXw", "elasticsearch.node.name": "572f822926ad"
, "elasticsearch.cluster.name": "docker-cluster"}

```

View in Docker Desktop View Config Enable Watch

```
← ↻ ↺ localhost:9200
Автоформатировать ☐
{
  "name" : "572f822926ad",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "vxWAmI3wRT2WL95vQceXWQ",
  "version" : {
    "number" : "8.14.1",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "93a57a1a76f556d8aee6a90d1a95b06187501310",
    "build_date" : "2024-06-10T23:35:17.114581191Z",
    "build_snapshot" : false,
    "lucene_version" : "9.10.0",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}

{
  event: 'reverted',
  name: '20240719151942-add_text_tsv_column_to_post_table.js',
  durationSeconds: 0.022
}
Executing (default): SELECT "id", "name", "role", "email", "password", "isActivated", "refreshToken", "verifToken", "createdAt", "updatedAt", "rating" FROM "Users" AS "User" WHERE "User"."isActivated" = false;
Executing (default): SELECT "id", "name", "role", "email", "password", "isActivated", "refreshToken", "verifToken", "createdAt", "updatedAt", "rating" FROM "Users" AS "User" WHERE "User"."isActivated" = true AND "User"."verifToken" IS NOT NULL;
Server running on http://localhost:5000 - development
```

Все успешно перезапустилось.

После перезапуска, перейдем в Swagger и выполним опять метод GET /api/posts/public, передав параметры: searchParam='title', searchString='кирпичная стена'. Если данные сохранились, то данный запрос должен вернуть нам все посты из предыдущих тестов:

posts

GET /api/posts/public

Get all user posts

Parameters

Cancel

Name	Description
searchParam string (query)	Search parameter that sets among whom the search substring will be searched - titles or contents <div>title</div>
searchString string (query)	String that is searched among titles or contents <div>кирпичная стена</div>

Execute

Clear

Responses

Curl

curl -X 'GET' \
'http://localhost:5000/api/posts/public?searchParam=title&searchString=ND0NBAND0NB8ND1X80ND0NBFXD0NB8ND1X87ND0NB0ND0NB0ND1X8FX20ND1X81ND1X82ND0NB5ND0NB0ND0NB0' \
-H 'accept: application/json'

Request URL

http://localhost:5000/api/posts/public?searchParam=title&searchString=ND0NBAND0NB8ND1X80ND0NBFXD0NB8ND1X87ND0NB0ND0NB0ND1X8FX20ND1X81ND1X82ND0NB5ND0NB0ND0NB0

Server response

Code

Details

200

Response body

{
 "status": 200,
 "data": [
 {
 "id": "4735b21f-ab56-4ffd-bc2f-d8298771465d",
 "title": "Мои кирпичные стены",
 "content": "Расскажу об..."
 },
 {
 "id": "5c805f58-ba75-48dd-9278-762531e23fa5",
 "title": "Кирпичная дорогая стена",
 "content": "Хочу установить..."
 }
 ],
 "message": "List of all posts"
}

Download

Response headers

access-control-allow-credentials: true
access-control-allow-origin: \*
connection: keep-alive
content-length: 329
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 17:52:30 GMT
etag: W/"149-1Agg0L97gEne/tP0MgzmlkV8s8"
keep-alive: timeout=5
x-powered-by: Express

Мы действительно при фразовом поиске получили все посты из предыдущих тестов. Это значит, что данные в индексе сохраняются, даже при перезапуске Docker.

**Проверка на получение постов других пользователей:**

Теперь проверим, что другой пользователь не может получать чужие посты при фразовом поиске. Для этого выйдем из системы:

GET

/auth/logout Logout of account

Parameters

No parameters

Execute

Clear

Responses

Curl

curl -X 'GET' \
'http://localhost:5000/auth/logout' \
-H 'accept: \*/\*'

Request URL

http://localhost:5000/auth/logout

Server response

Code

Details

200

Response body

{
 "status": 200,
 "message": "Logged out successfully"
}

Download

Response headers

access-control-allow-credentials: true
access-control-allow-origin: \*
content-length: 50
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 17:57:30 GMT
etag: W/"32-uPVLuQe00IsuQx91d+XLBQ9CTlDQ"
x-powered-by: Express

auth

GET /auth/payload Get a user payload

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:5000/auth/payload' \
-H 'accept: application/json'
```

Request URL

http://localhost:5000/auth/payload

Server response

Code	Details
401 <small>Undocumented</small>	<p>Error: Unauthorized</p> <p>Response body</p> <p>Unauthorized - you are not signed in</p> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-origin: * connection: keep-alive content-length: 36 content-type: text/html; charset=utf-8 date: Tue, 30 Jul 2024 17:58:04 GMT etag: W/"24-1pnbilyb3ip60XZvllipj14ybA" keep-alive: timeout=5 x-powered-by: Express</pre>

Мы успешно вышли из системы.

Теперь авторизируемся как другой пользователь:

auth

GET /auth/payload Get a user payload

POST /auth/signin Login to account

Parameters

No parameters

Request body required

application/json

Examples:

[Modified value]

```
{
  "email": "nozdryakovbogdan3112@mail.ru",
  "password": "bgdghsH748LkKd???"
}
```

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:5000/auth/signin' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "email": "nozdryakovbogdan3112@mail.ru",
    "password": "bGdghsH748Lkdd??"
  }'
```

Request URL

http://localhost:5000/auth/signin

Server response

Code	Details
200	<p>Response body</p> <pre>{   "status": 200,   "data": {     "name": "ingaale",     "email": "nozdryakovbogdan3112@mail.ru"   },   "message": "Authorization was successful" }</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-origin: * connection: keep-alive content-length: 121 content-type: application/json; charset=utf-8 date: Tue, 30 Jul 2024 17:59:30 GMT etag: W/"79-507Lfw4kzmPN1prA4NV/4bch80" keep-alive: timeout=5 x-powered-by: Express</pre>

GET /auth/payload Get a user payload

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5000/auth/payload' \
  -H 'accept: application/json'
```

Request URL

http://localhost:5000/auth/payload

Server response

Code	Details
200	<p>Response body</p> <pre>{   "status": 200,   "data": {     "name": "ingaale",     "role": "user",     "email": "nozdryakovbogdan3112@mail.ru"   },   "message": "User details" }</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-origin: * connection: keep-alive content-length: 119 content-type: application/json; charset=utf-8 date: Tue, 30 Jul 2024 18:00:47 GMT etag: W/"77-IV+rDh8MDjk1OCoinQasxHCBg" keep-alive: timeout=5 x-powered-by: Express</pre>

Мы успешно авторизовались как другой пользователь.

Теперь выполним метод GET /api/posts/public без передачи параметров и посмотрим на посты данного пользователя:

## posts

GET

/api/posts/public Get all user posts



### Parameters

Cancel

Name Description

searchParam Search parameter that sets among whom the search substring will be searched - titles or contents

string  
(query) --

searchString String that is searched among titles or contents

string  
(query) searchString

Execute

Clear

### Responses

#### Curl

```
curl -X 'GET' \
  'http://localhost:5000/api/posts/public' \
  -H 'accept: application/json'
```

#### Request URL

http://localhost:5000/api/posts/public

#### Server response

Code Details

200

#### Response body

```
{
  "status": 200,
  "data": [
    {
      "id": "1edfc759-927a-4bba-8337-efe74706531a",
      "title": "Властелин колец - книги против фильмов",
      "access": "public",
      "content": "Я прочитал все книги, а также посмотрел все фильмы про Властелин Колец...",
      "rating": 0,
      "tags": [
        "фильмы",
        "книги",
        "Властелин Колец"
      ]
    },
    {
      "id": "81caa256-1b2a-4082-ab6f-2063cdf6e9d",
      "title": "Я влюбился в IVE",
      "access": "public",
      "content": "Я влюбился в IVE! Они очень классные...",
      "rating": 0,
      "tags": null
    },
    {
      "id": "6e016d12-e994-4089-8eff-00fb520b8acf",
      "title": "Мой дом - моя крепость",
      "access": "public",
      "content": "Я всегда считал, что стена моего дома..."
    }
  ]
}
```

#### Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 1866
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 18:02:59 GMT
etag: W/"74a-511Ve2r9u5Qg+MSLr9YjMcGokXI"
keep-alive: timeout=5
x-powered-by: Express
```

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5000/api/posts/public' \
  -H 'accept: application/json'
```

Request URL

http://localhost:5000/api/posts/public

Server response

Code Details

200

Response body

```
{
  "id": "6e816d12-e994-4089-8eff-00fb520b8acf",
  "title": "Мой дом - моя крепость",
  "access": "public",
  "content": "Я всегда считал, что стена моего дома...",
  "rating": 0,
  "tags": null
},
{
  "id": "7bdf15f7-65eb-4c2d-bf68-8edaf63d21d",
  "title": "Самая надежная кирпичная стена",
  "access": "public",
  "content": "В далеке я увидел свет. Стена моего дома была разрушена...",
  "rating": 0,
  "tags": null
},
{
  "id": "7a7780ef-c2b9-4f54-9c95-072b4772d33e",
  "title": "Новая стена",
  "access": "public",
  "content": "Я построил новый дом. Моя новая стена...",
  "rating": 0,
  "tags": null
},
{
  "id": "7ae3e293-7482-4982-8515-684e2b339b61",
  "title": "Ремонт дома",
  "access": "public"
}
```

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 1866
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 18:02:59 GMT
etag: W/"74a-51fYe2r9wsQg+MSLr9YJMcGokXI"
keep-alive: timeout=5
x-powered-by: Express
```

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5000/api/posts/public' \
  -H 'accept: application/json'
```

Request URL

http://localhost:5000/api/posts/public

Server response

Code Details

200

Response body

```
{
  "id": "7a7780ef-c2b9-4f54-9c95-072b4772d33e",
  "title": "Новая стена",
  "access": "public",
  "content": "Я построил новый дом. Моя новая стена...",
  "rating": 0,
  "tags": null
},
{
  "id": "7ae3e293-7482-4982-8515-684e2b339b61",
  "title": "Ремонт дома",
  "access": "public",
  "content": "Кирпичная стена данного дома имеет приличную толщину",
  "rating": 0,
  "tags": null
},
{
  "id": "d052ac89-12d3-49ce-8fe6-72034e5c3751",
  "title": "Строительство - тяжелый труд",
  "access": "public",
  "content": "Данная кирпичная постройка была в ужасном состоянии. Стена была полностью разрушена...",
  "rating": 0,
  "tags": null
}
},
{
  "message": "List of all posts"
}
```

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 1866
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 18:02:59 GMT
etag: W/"74a-51fYe2r9wsQg+MSLr9YJMcGokXI"
keep-alive: timeout=5
x-powered-by: Express
```

То есть данный пользователь имеет следующие посты:

{

"id": "1edfc759-927a-4bba-8337-efe74706531a",

"title": "Властелин колец - книги против фильмов",



```
"access": "public",
"content": "Я прочитал все книги, а также посмотрел все фильмы про Властелин
Колец...",
"rating": 0,
"tags": [
  "фильмы",
  "книги",
  "Властелин Колец"
]
},
{
  "id": "81caa256-1b2a-4082-ab6f-2063cdaf6e9d",
  "title": "Я влюбился в IVE",
  "access": "public",
  "content": "Я влюбился в IVE! Они очень классные...",
  "rating": 0,
  "tags": null
},
{
  "id": "6e016d12-e994-4089-8eff-00fb520b8acf",
  "title": "Мой дом - моя крепость",
  "access": "public",
  "content": "Я всегда считал, что стена моего дома...",
  "rating": 0,
  "tags": null
},
{
  "id": "7bdf15f7-65eb-4c2d-bf68-9eda4e3dc21d",
  "title": "Самая надежная кирпичная стена",
  "access": "public",
  "content": "В далеке я увидел свет. Стена моего дома была разрушена...",
  "rating": 0,
  "tags": null
}
```

```
},
{
  "id": "7a7780ef-c2b9-4f54-9c95-072b4772d33e",
  "title": "Новая стена",
  "access": "public",
  "content": "Я построил новый дом. Моя новая стена...",
  "rating": 0,
  "tags": null
},
{
  "id": "7ae3e293-7482-4982-8515-684e2b339b61",
  "title": "Ремонт дома",
  "access": "public",
  "content": "Кирпичная стена данного дома имеет приличную толщину",
  "rating": 0,
  "tags": null
},
{
  "id": "d052ac89-12d3-49ce-8fe6-72034e5c3751",
  "title": "Строительство - тяжелый труд",
  "access": "public",
  "content": "Данная кирпичная постройка была в ужасном состоянии. Стена была полностью разрушена...",
  "rating": 0,
  "tags": null
}
```

Теперь выполним метод GET /api/posts/public, передав параметры: searchParam='title', searchString='кирпичная стена'. Данный пользователь имеет только один пост, где есть указанная строка в title. Данный метод не должен возвращать посты предыдущего пользователя, где в title также есть указанная строка:

posts

GET

/api/posts/public

Get all user posts

Parameters

Name

Description

searchParam

Search parameter that sets among whom the search substring will be searched - titles or contents

string (query)

title

searchString

String that is searched among titles or contents

string (query)

кирпичная стена

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:5000/api/posts/public?searchParam=title&searchString=%D0%BA%D0%BB%D1%80%D0%BF%D0%BE%D1%87%D0%BD%D0%BD%D1%8F%20%D1%81%D1%82%D0%BE%D0%A0' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:5000/api/posts/public?searchParam=title&searchString=%D0%BA%D0%BB%D1%80%D0%BF%D0%BE%D1%87%D0%BD%D0%BD%D1%8F%20%D1%81%D1%82%D0%BE%D0%A0
```

Server response

Code

Details

200

Response body

```
{
  "status": 200,
  "data": [
    {
      "id": "7bdf15f7-65eb-4c2d-bf68-9eda4e3dc21d",
      "title": "Самая надежная кирпичная стена",
      "content": "В далеке я увидел свет. Стена моего дома была разрушена..."
    }
  ],
  "message": "List of all posts"
}
```

Download

Response headers

```
access-control-allow-credentials: true
access-control-allow-origin: *
connection: keep-alive
content-length: 283
content-type: application/json; charset=utf-8
date: Tue, 30 Jul 2024 18:07:39 GMT
etag: W/"11b-S2ox2XG/64a4GEY5MA86OCINxb8"
keep-alive: timeout=5
x-powered-by: Express
```

Пользователь получил именно свой пост и не получил посты предыдущего пользователя. Значит все работает правильно – фразовый поиск позволяет искать пользователям только свои посты.