Введение

Вы почти закончили обучение и дошли до завершающей части курса, большая часть теории и практики осталась позади. Мы гордимся вашим трудолюбием и упорством! Теперь осталось совсем немного — выполнить и защитить **дипломный проект.**

Некоторые считают диплом формальностью: «Я получил знания, а бумажка мне не важна». Но диплом — не просто свидетельство. Это кейс в ваше портфолио, который можно показывать работодателю. Это полезно, даже если прямо сейчас вы не ищете работу: например, для роста внутри компании и для закрепления ваших знаний, полученных в курсе.

Поэтому давайте соберёмся и сделаем это. Будет интересно!

Разработка локального поискового движка по сайту

Перед вами подробное описание будущего проекта, который станет отличным дополнением вашего портфолио. Здесь есть всё, что вам нужно, чтобы справиться с поставленной задачей.

Содержание

Описание задачи

Вы пришли работать в отдел разработки программного обеспечения недавно созданного информационно-новостного портала, на котором каждый день выходят новости о событиях в мире и статьи разных авторов. Руководитель поручил вам реализацию собственного поискового движка, который будет помогать посетителям сайта быстро находить нужную информацию, используя поле поиска.

Существующие поисковые движки ваши коллеги уже попробовали. Лучшим из них был давно существовавший движок Яндекс.Сервер, который можно было установить на своём сервере и использовать для поиска по своему сайту или нескольким своим сайтам. К сожалению, движок прекратил существовать как отдельный сервис. Ваше руководство решило реализовать собственный поиск, алгоритм и принципы работы которого при необходимости можно будет менять и развивать.

Задачи по реализации поискового движка будут выдаваться вам постепенно. Руководство хочет убедиться, что придуманный ими алгоритм работает верно, и не планирует перегружать вас сразу большим техническим заданием, к которому будет сложно подступиться.

Поисковый движок должен будет представлять из себя Spring-приложение (JAR-файл, запускаемый на любом сервере или компьютере), работающее с локально установленной базой данных MySQL, имеющее простой веб-интерфейс и API, через который им можно управлять и получать результаты поисковой выдачи по запросу. Принципы работы поискового движка должны быть следующими:

1. В конфигурационном файле перед запуском приложения задаются адреса сайтов, по которым движок должен будет осуществлять поиск.
2. Поисковый движок должен самостоятельно обходить все страницы заданных сайтов и индексировать их таким образом, чтобы потом по любому поисковому запросу находить наиболее релевантные (подходящие) страницы.
3. Пользователь присылает запрос через API движка. Запрос — это набор слов, по которым нужно найти страницы сайта.
4. Запрос определённым образом трансформируется в список слов, переведённых в базовую форму (например, для существительных — именительный падеж, единственное число).
5. В индексе ищутся те страницы, на которых встречаются все эти слова.
6. Результаты поиска ранжируются, сортируются и отдаются пользователю.

Ниже вы найдёте все технические подробности реализации поискового движка, которые помогут вам создать работающее приложение. Они разбиты на несколько небольших этапов. По каждому этапу подробно расписано, что необходимо сделать и как проверить конечный результат.

Если у вас появятся вопросы или вы найдёте ошибку в техническом задании, смело обращайтесь к вашему проверяющему преподавателю.

Этап 1. Подготовка

Цель

Подготовить всё необходимое программное обеспечение, чтобы начать программировать поисковый движок.

Что нужно сделать

1. Установите на свой компьютер JDK и среду разработки IntelliJ IDEA, если они ещё не установлены.

2. Установите на свой компьютер MySQL-сервер (если он ещё не установлен) и создайте в нём пустую базу данных search_engine. Для установки можете воспользоваться инструкциями. Создайте пользователя для подключения к этой базе данных. Это может быть пользователь root, который имеет доступ ко всем базам данных и который создаётся при установке MySQL-сервера, а может быть отдельный пользователь, имеющий доступ только к созданной вами базе данных, на ваше усмотрение.

Этап 2. Система обхода страниц сайта

Цель

Реализовать многопоточное приложение, которое обходит все страницы сайта, начиная с главной.

Что нужно сделать

Напишите программу, которая будет обходить заданный сайт, начиная с главной страницы, и добавлять адреса, статусы и содержимое всех страниц в базу данных:

1. Для перехода по очередной ссылке должен создаваться новый поток при помощи Fork-Join.
2. Этот поток будет при помощи JSOUP получать содержимое этой страницы и перечень ссылок, которые есть на этой странице (значений атрибутов href HTML-тегов <a>).

3. Ссылки должны фильтроваться таким образом, чтобы оставались только непросмотренные внутренние ссылки на страницы этого же сайта.
4. Просмотренные ссылки и содержимое страниц по этим ссылкам необходимо добавлять в отдельную таблицу в базе данных page, имеющую следующую структуру:

    - id INT NOT NULL AUTO_INCREMENT;
    - path TEXT NOT NULL — адрес страницы от корня сайта (должен начинаться со слеша, например: /news/372189/);
    - code INT NOT NULL — код ответа, полученный при запросе страницы (например, 200, 404, 500 или другие);
    - content MEDIUMTEXT NOT NULL — контент страницы (HTML-код).

5. По полю path должен быть установлен индекс, чтобы поиск по нему был быстрым, когда в нём будет много ссылок.
6. Код ответа необходимо научиться определять самостоятельно, воспользовавшись документацией к библиотеке JSOUP.
7. По итогу работы программы в таблице должны оказаться ссылки на все страницы этого сайта без повторов.
8. Чтобы внешний сайт считал запросы от вашего приложения обычными посещениями пользователей, можно, например, обращаться к нему таким образом (установив фейковый User-Agent и фейковый referrer):

```
doc = Jsoup.connect("https://www.facebook.com/")
        .userAgent("Mozilla/5.0 (Windows; U; WindowsNT
5.1; en-US; rv1.8.1.6) Gecko/20070725 Firefox/2.0.0.6")
        .referrer("http://www.google.com")
        .get();
```

    Рекомендуется установить корректное значение User-Agent, например HeliontSearchBot (поисковый бот Heliont), где Heliont — пример названия вашего поискового движка, которое вы можете дать ему самостоятельно.

Некоторые сайты могут быть защищены от слишком частых запросов, и вам надо обходить их аккуратно, с задержками в 0,5–5 секунд между запросами.

Как проверить работу программы

Запустите программу, подставив в качестве адреса сайта любой из следующего списка:

1. http://www.playback.ru/
2. https://volochek.life/
3. http://radiomv.ru/
4. https://ipfran.ru/
5. https://dimonvideo.ru/
6. https://nikoartgallery.com/
7. https://et-cetera.ru/mobile/
8. https://www.lutherancathedral.ru/
9. https://dombulgakova.ru/
10. https://www.svetlovka.ru/

По итогам работы программы в базе данных должен оказаться список страниц сайта, который был передан программе. Например, для сайта http://www.playback.ru/ должен быть список такого вида:

id	path	code	content
1	/	200	<!DOCTYPE html><html><head><title>Инт...
2	/dostavka.html	200	<!DOCTYPE html><html><head><title>Инт...
3	/pickup.html	200	<!DOCTYPE html><html><head><title>Инт...
4	/payment.html	200	<!DOCTYPE html><html><head><title>Инт...
...	...	...	...

В поле content должны быть коды соответствующих страниц сайта.

Этап 3. Лемматизатор

Описание

Лемматизатор — это специальная библиотека, которая позволяет получать леммы слов — их исходные формы (для существительных, например, это слово в именительном падеже и единственном числе). Лемматизацию удобно использовать в поисковых движках, поскольку она позволяет искать нужную информацию с учётом морфологии. Например, на странице встречается слово «лошадей», а вы вводите поисковый запрос «лошадь». При простом поиске наличия слова в тексте данная страница не найдётся, а если

все слова привести к «лошадь», то найдутся все страницы, на которых это слово присутствует в исходном или изменённом варианте.

Существует множество лемматизаторов. Мы рекомендуем воспользоваться лемматизатором, который используется в поисковом движке Apache Solr: https://github.com/akuznetsov/russianmorphology.

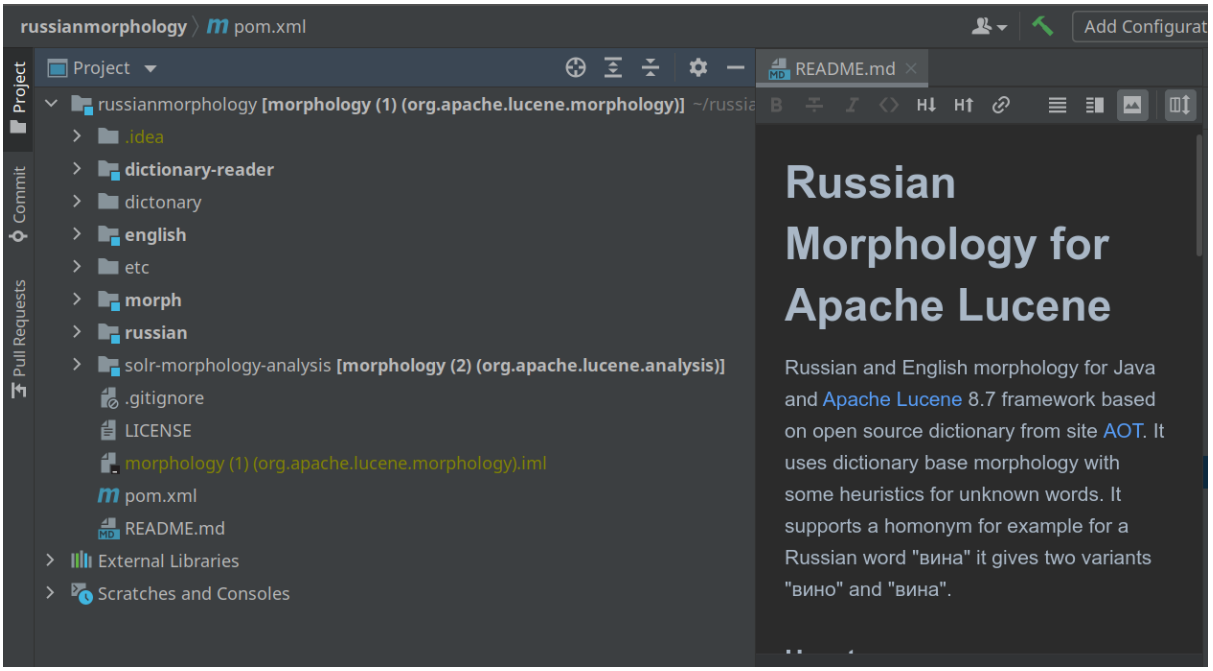
Цель

Подключить и опробовать библиотеку лемматизации слов.

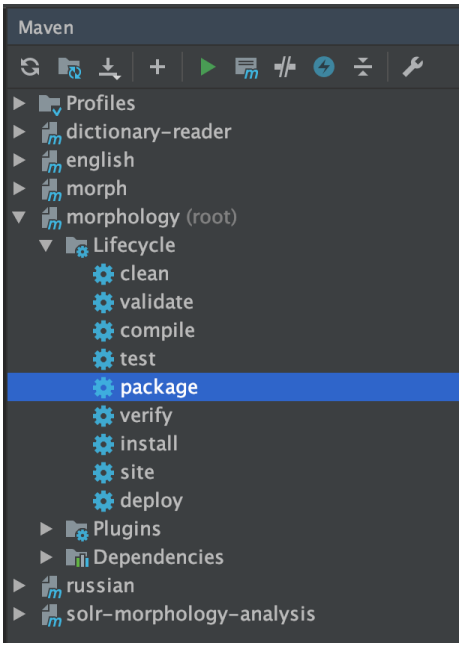
Что нужно сделать

1. Создайте папку, в которую вы склонируете исходники лемматизатора и выполните команду `git clone https://github.com/akuznetsov/russianmorphology`

2. Откройте проект директорию russianmorphology (File → Open…) и откройте проект:



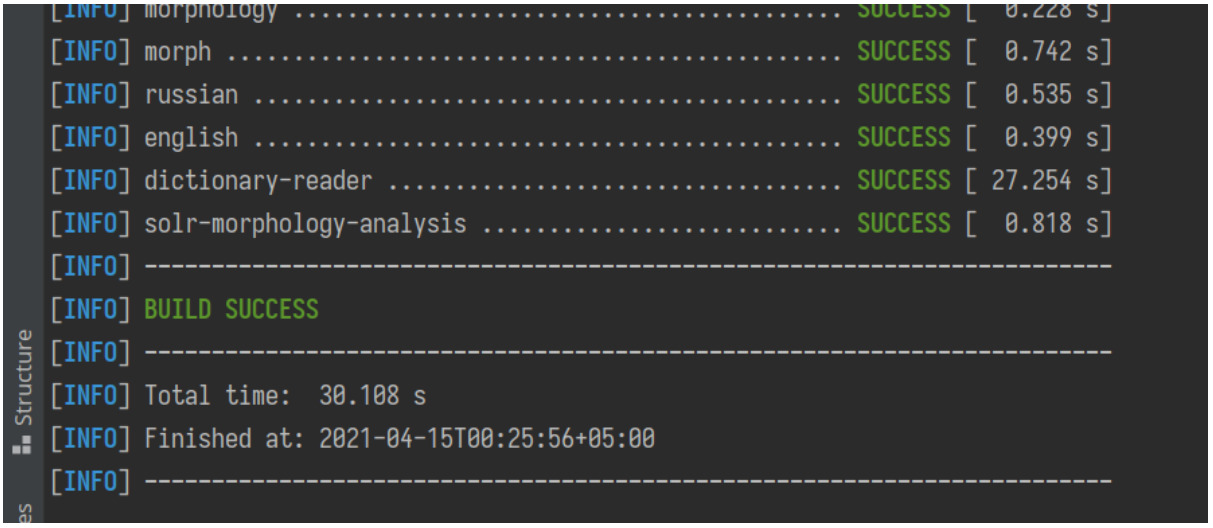
3. Запустите команду package у корневого Maven-проекта morphology:



Или выполните команду в командной строке в корне проекта:

```
mvn package
```

Убедитесь, что сборка прошла успешно:



4. Если вы работаете в операционной системе Linux или macOS, в корне созданного проекта в командной строке выполните команду:

   ```
   find . -name "*.jar"
   ```

   Эта команда выведет список всех JAR-файлов, которые сформировались в проекте и которые вам необходимо скопировать в свой проект и подключить. На момент написания текущего технического задания формируются следующие файлы:

- ./morph/target/morph-1.5.jar
- ./russian/target/russian-1.5.jar
- ./english/target/english-1.5.jar
- ./dictionary-reader/target/dictionary-reader-1.5.jar
- ./solr-morphology-analysis/target/morphology-1.5.jar

Если вы работаете в операционной системе Windows, убедитесь, что эти файлы присутствуют в соответствующих папках.

5. Подключите эти JAR-файлы к новому проекту, в котором вы будете экспериментировать с лемматизатором, и создайте отдельный класс с методом main, чтобы проверить, как работает лемматизация. Код лемматизатора работает следующим образом:

```
LuceneMorphology luceneMorph =
    new RussianLuceneMorphology();
List<String> wordBaseForms =
    luceneMorph.getNormalForms("леса");
wordBaseForms.forEach(System.out::println);
```

Данный код выведет следующие строки (две возможные исходные формы слова «леса»):

```
лес
леса
```

6. Создайте класс с методом, который будет принимать в качестве параметра текст, а выдавать перечень лемм для каждого слова в этом тексте (за исключением междометий, союзов, предлогов и частиц — см. ниже) и количество вхождений данной леммы в текст.

Пример текста на входе:

```
Повторное появление леопарда в Осетии позволяет предположить, что
леопард постоянно обитает в некоторых районах Северного Кавказа.
```

Ожидаемый результат:

```
повторный — 1
появление — 1
постоянно — 1
позволять — 1
```

```
предположить — 1
северный — 1
район — 1
кавказ — 1
осетия — 1
леопард — 2
обитать — 1
```

7. Передаваемые в метод тексты необходимо очищать от служебных частей речи. Часть речи можно определить так:

```
LuceneMorphology luceneMorph =
    new RussianLuceneMorphology();
List<String> wordBaseForms =
    luceneMorph.getMorphInfo("или");
wordBaseForms.forEach(System.out::println);
```

Такой код выдаст следующую информацию:

```
или|n СОЮЗ
```

В данном случае «СОЮЗ» означает, что слово является союзом. Другие примеры:

```
и|o МЕЖД
копал|A C мр,ед,им
копать|a Г дст,прш,мр,ед
хитро|j Н
хитрый|Y КР_ПРИЛ ср,ед,од,но
синий|Y П мр,ед,вн,но
```

Как проверить работу программы

Попробуйте передать на вход программы несколько разных текстов и проверьте, верно ли выдаётся список лемм с количествами.

Этап 4. Система индексации веб-страниц

Описание

Индексация — это процесс формирования поискового индекса по некоторому объёму информации. Поисковый индекс — это специальным образом организованная база данных (в нашем случае — база данных MySQL), позволяющая быстро и удобно осуществлять поиск по этой информации.

Скорость поиска по поисковому индексу в любых поисковых системах, как правило, занимает очень короткое время (обычно доли секунды) по сравнению с обычным поиском перебором по всему массиву информации (вспомните разницу в скорости поиска перебором в простом массиве и бинарного поиска в отсортированном).

Удобство хранения информации в индексе достигается за счёт правильно организованной структуры хранения — базы данных. В частности, по обычному тексту или набору текстов вы не сможете искать информацию с учётом морфологии русского языка и одновременно оценивать релевантность результатов, если у вас не будет специально организованного поискового индекса.

Цель

Реализовать систему индексации страниц сайта — систему, которая позволит подсчитывать встречающиеся на страницах сайта слова (точнее, их леммы) и в дальнейшем по поисковому запросу определять наиболее релевантные (соответствующие поисковому запросу) страницы.

Что нужно сделать

На втором этапе вы уже реализовали систему, которая умеет обходить страницы сайта по ссылкам в многопоточном режиме, а на третьем этапе написали программу, которая извлекает слова из текста, преобразует их в леммы и считает количество вхождений каждой леммы в текст.

Теперь необходимо сделать так, чтобы ваша программа не только обходила весь сайт, но и сохраняла информацию о содержащихся леммах в базу данных. Для этого расширьте имеющуюся базу данных таким образом, чтобы она имела следующую структуру:

field — поля на страницах сайтов

- id INT NOT NULL AUTO_INCREMENT;
- name VARCHAR(255) NOT NULL — имя поля;

- selector VARCHAR(255) NOT NULL — CSS-выражение, позволяющее получить содержимое конкретного поля;
- weight FLOAT NOT NULL — релевантность (вес) поля от 0 до 1.

Значения, которыми изначально должна быть заполнена таблица:

name	selector	weight
title	title	1,0
body	body	0,8

page — проиндексированная страница

- id INT NOT NULL AUTO_INCREMENT;
- path TEXT NOT NULL — адрес страницы от корня сайта (должен начинаться со слеша, например: /news/372189/);
- code INT NOT NULL — код ответа, полученный при запросе страницы (например, 200, 404, 500 или другие);
- content MEDIUMTEXT NOT NULL — контент страницы (HTML-код).

lemma — леммы, встречающиеся в текстах (см. справочно: лемматизация)

- id INT NOT NULL AUTO_INCREMENT;
- lemma VARCHAR(255) NOT NULL — нормальная форма слова;
- frequency INT NOT NULL — количество страниц, на которых слово встречается хотя бы один раз. Максимальное значение не может превышать общее количество слов на сайте.

index — поисковый индекс

- id INT NOT NULL AUTO_INCREMENT;
- page_id INT NOT NULL — идентификатор страницы;
- lemma_id INT NOT NULL — идентификатор леммы;
- rank FLOAT NOT NULL — ранг леммы на этой страницы (см. ниже алгоритм расчёта).

После создания структуры базы данных допишите систему обхода сайта таким образом, чтобы она выполняла следующие операции:

1. Извлекала из кода очередной страницы (сначала — главной) блоки информации, соответствующие HTML-элементам, указанным в таблице field базы данных. Это можно делать с помощью библиотеки JSOUP.

2. Полученные фрагменты страницы очищала от HTML-тегов и разбивала на отдельные слова, а каждое слово преобразовывала в лемму (исходную форму) с помощью написанного вами лемматизатора.
3. Собирала все уникальные леммы для данной страницы и считала их количества.
4. Если лемма отсутствует в базе, добавляла её в таблицу lemma со значением frequency равным 1. Если присутствует, увеличивала число frequency на 1. Число frequency должно соответствовать количеству страниц, на которых лемма встречается хотя бы один раз.
5. Добавляла связку леммы и страницы, на которой она встречается, в таблицу _index со значением rank, которое рассчитывается как сумма произведений количества данной леммы в поле страницы (_field) и веса данного поля. К примеру, слово «лошадь» встречается один раз в title и четыре раза в body некой страницы. В этом случае rank должен быть рассчитан следующим образом:

$$R = 1,0 + 4 * 0,8 = 4,2$$

6. Некоторые ссылки могут вести на несуществующие страницы (код ответа — 404) или страницы, содержащие ошибки (код ответа — 500). Такие страницы также необходимо добавлять в таблицу page, но не индексировать.

Как проверить работу программы

На данном этапе достаточно убедиться в том, что после индексации сайта (из списка выше, см. этап 2) данные в базе данных, в том числе в таблицах lemma и index выглядят правдоподобно. Индексация при этом завершается без ошибок и для всех страниц есть записи в таблице _index. Проверить корректность индексации можно будет после реализации системы поиска (см. следующий этап).

Этап 5. Система поиска

Цель

Реализовать систему поиска информации с использованием созданного поискового индекса.

Что нужно сделать

Необходимо дописать в программе код, который будет по поисковому запросу (строке текста) выдавать результаты поиска в виде списка объектов с их свойствами. Этот код должен выполнять следующий алгоритм:

1. Разбивать поисковый запрос на отдельные слова.
2. Формировать из этих слов список уникальных лемм. Исключать из списка слов междометия, союзы, предлоги и частицы. Кроме того, рекомендуется исключать леммы, которые встречаются на слишком большом количестве страниц (определите этот процент самостоятельно).
3. Сортировать леммы в порядке увеличения частоты встречаемости (по возрастанию значения поля frequency) — от самых редких до самых частых.
4. По первой, самой редкой лемме из списка находить все страницы, на которых лемма встречается. Далее искать соответствия следующей леммы и этого списка страниц, и так по каждой следующей лемме. Список страниц при этом на каждой итерации должен уменьшаться.
5. Если в итоге не осталось ни одной страницы, выводить пустой список.
6. Если страницы найдены, рассчитывать по каждой из них релевантность (и выводить её потом, см. ниже). Для этого для каждой страницы рассчитывать абсолютную релевантность — сумму всех rank всех найденных на странице лемм (из таблицы _index), которая делится на максимальное значение этой абсолютной релевантности для всех найденных страниц. Пример расчёта:

Страница	**Rank слова «лошадь»**	**Rank слова «бегает»**	**Абсолютная релевантность**	**Относительная релевантность**
1	4,2	3,1	7,3	0,7
2	1,0	1,5	2,5	0,24
3	9,9	0,4	10,3	1

Пример расчёта абсолютной релевантности для первой страницы:

$$R_{abs} = 4,2 + 3,1 = 7,3$$

Относительную релевантность можно получить делением абсолютной для конкретной страницы на максимальную абсолютную релевантность среди всех страниц данной поисковой выдачи:

$$R_{rel} = 7{,}3 / 10{,}3 = 0{,}70873786$$

7. Сортировать страницы по убыванию релевантности (от большей к меньшей) и выдавать в виде списка объектов со следующими полями:

    - uri — путь к странице вида /path/to/page/6784;
    - title — заголовок страницы;
    - snippet — фрагмент текста, в котором найдены совпадения, выделенные жирным, в формате HTML (выделение жирным — тег <b>);
    - relevance — релевантность страницы (см. выше формулу расчёта).

Как проверить работу программы

Проиндексируйте несколько сайтов из списка выше; запустите поиск по фразам, которые встречаются на определённых страницах каждого сайта и не встречаются на этих сайтах; сравните результат с ожидаемым. В случае запроса фразы, которая есть на какой-либо странице сайта, эта страница должна быть найдена и должна иметь высокую релевантность, а если запросить поиск фразы, которой нет на сайте, должен быть выдан пустой список.

Этап 6. Веб-приложение и многосайтовый режим

Цель

Создать из получившейся программы веб-приложение, которое будет индексировать несколько сайтов.

Что нужно сделать

1. Создайте веб-приложение на Spring Boot (аналогично тому, как это делалось в модуле по созданию веб-приложений).

2. Добавьте в базу данных таблицу site, имеющую следующую структуру:

- id INT NOT NULL AUTO_INCREMENT;
- status ENUM('INDEXING', 'INDEXED', 'FAILED') NOT NULL — текущий статус полной индексации сайта, отражающий готовность поискового движка осуществлять поиск по сайту — индексация или переиндексация в процессе, сайт полностью проиндексирован (готов к поиску) или не удалось проиндексировать (сайт не готов к поиску и не будет до устранения ошибок и перезапуска индексации);
- status_time DATETIME NOT NULL — дата и время статуса (в случае статуса INDEXING дата и время должны обновляться регулярно при добавлении каждой новой страницы в индекс);
- last_error TEXT — текст ошибки индексации или NULL, если её не было;
- url VARCHAR(255) NOT NULL — адрес главной страницы сайта;
- name VARCHAR(255) NOT NULL — имя сайта.

3. Добавьте в таблицы page и lemma поле site_id, которое позволит индексировать сайты независимо.

4. Создайте конфигурационный файл приложения в формате YAML (application.yaml) и разместите его в корне проекта. В будущем при сборке приложения в единый JAR-файл размещайте конфигурационный файл за пределами приложения рядом с этим JAR-файлом. В YAML-файле должны содержаться:

    - **Данные доступа к локальной базе данных MySQL: хост, логин, пароль, имя базы.**
    - **Перечень сайтов, которые необходимо индексировать.** Это должен быть массив объектов, содержащих адрес и имя сайта. В перечне могут находиться как один, так и несколько сайтов. Адреса всех сайтов должны быть полные и не должны содержать слеш в конце. Например:

```
sites:
     - url: https://www.lenta.ru
       name: Лента.ру
     - url: https://www.skillbox.ru
       name: Skillbox
```

    - **Имя User-Agent, который необходимо подставлять при запросах страниц сайтов.** Это необходимо для того, чтобы корректно представлять свой поисковый движок сайту. Чтобы сайт считал запросы от вашего поискового движка обычными посещениями пользователей, можно, например, обращаться к

нему таким образом (установив фейковый User-Agent и фейковый referrer):

```
doc = Jsoup.connect("https://www.facebook.com/")
        .userAgent("Mozilla/5.0 (Windows; U;
WindowsNT 5.1; en-US; rv1.8.1.6) Gecko/20070725
Firefox/2.0.0.6")
        .referrer("http://www.google.com")
        .get();
```

Рекомендуется установить корректное значение User-Agent, например HeliontSearchBot (поисковый бот Heliont), где Heliont — пример названия вашего поискового движка, которое вы можете ему дать.

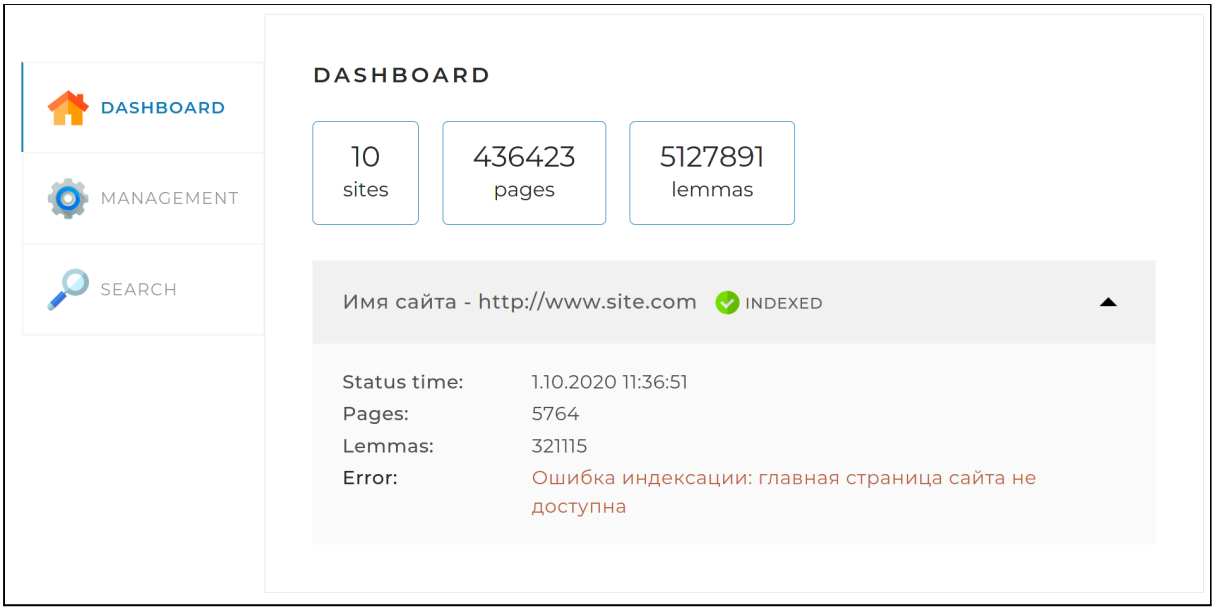
- Путь к веб-интерфейсу (по умолчанию — /admin).

5. Подключите frontend-составляющую к веб-приложению (положите в папку resources аналогично тому, как было показано в модуле по созданию веб-приложений).

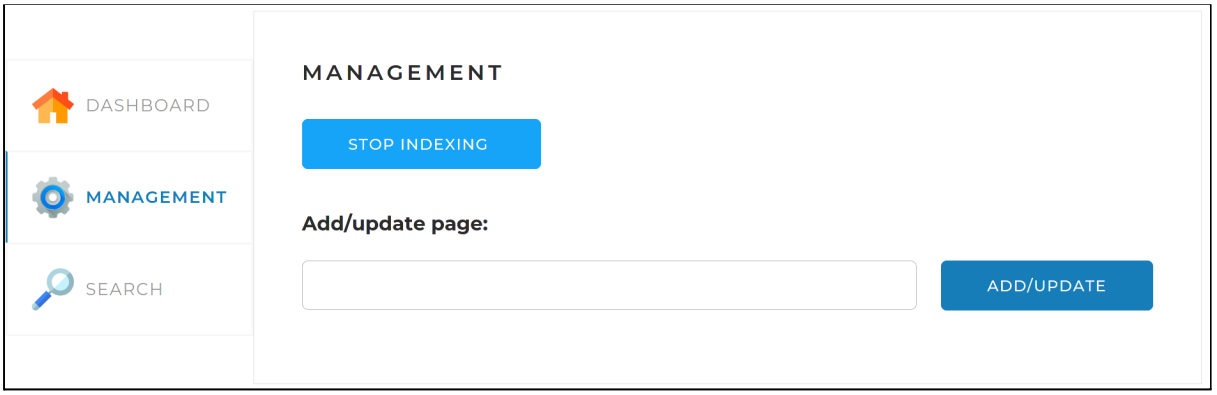
   Frontend-составляющая представляет собой веб-страницу, которая должна отображаться в приложении при входе по адресу, установленному в конфигурационном файле (по умолчанию — /admin).

   На этой веб-странице находятся три вкладки:

   - **Dashboard.** Эта вкладка открывается по умолчанию. На ней отображается общая статистика по всем сайтам, а также детальная статистика и статус по каждому из сайтов (статистика, получаемая по запросу /api/statistics).



- **Management.** На этой вкладке находятся инструменты управления поисковым движком — запуск и остановка полной индексации (переиндексации), а также возможность добавить (обновить) отдельную страницу по ссылке.



- **Search.** Эта страница предназначена для тестирования поискового движка. На ней находится поле поиска, выпадающий список с выбором сайта, по которому искать, а при нажатии на кнопку «Найти» выводятся результаты поиска (по API-запросу /api/search).

Вся информация подгружается на вкладки путём запросов страницы к API. Веб-страница уже разработана и находится репозитории. Ваша задача — разархивировать страницу и дополнительные файлы и добавить их в правильную папку в своём проекте.

6. Измените в коде страницы “index.html” значение переменной “backendApiUrl” на путь к хосту, на котором у вас размещено веб-приложение. В файле “index.html” есть следующие строки кода:

```
<script>
// замените значение переменной на ваш бекенд,
// например http://localhost:8080
var backendApiUrl =
     'https://virtserver.swaggerhub.com/lunpully/adminSea
rch/1.0.0';
</script>
```

В них нужно заменить только значение переменной “backendApiUrl”, указанное в одинарных кавычках.

7. Реализуйте команды API для управления индексацией, а также команду получения статистической информации.

О том, как получать данные из параметров запроса в Spring, читайте статью — https://www.baeldung.com/spring-request-param

Все команды должны соответствовать следующей спецификации:

Запуск полной индексации — GET /api/startIndexing

Метод запускает полную индексацию всех сайтов или полную переиндексацию, если они уже проиндексированы.

Если в настоящий момент индексация или переиндексация уже запущена, метод возвращает соответствующее сообщение об ошибке.

Параметры:

Метод без параметров

Формат ответа в случае успеха:

```
{
     'result': true
}
```

Формат ответа в случае ошибки:

```
{
     'result': false,
     'error': "Индексация уже запущена"
}
```

Остановка текущей индексации — GET /api/stopIndexing

Метод останавливает текущий процесс индексации (переиндексации). Если в настоящий момент индексация или переиндексация не происходит, метод возвращает соответствующее сообщение об ошибке.

Параметры:

Метод без параметров.

Формат ответа в случае успеха:

```
{
     'result': true
}
```

Формат ответа в случае ошибки:

```
{
     'result': false,
     'error': "Индексация не запущена"
```

```
}
```

Добавление или обновление отдельной страницы — POST /api/indexPage

Метод добавляет в индекс или обновляет отдельную страницу, адрес которой передан в параметре.

Если адрес страницы передан неверно, метод должен вернуть соответствующую ошибку.

Параметры:

- url — адрес страницы, которую нужно переиндексировать.

Формат ответа в случае успеха:

```
{
     'result': true
}
```

Формат ответа в случае ошибки:

```
{
     'result': false,
     'error': "Данная страница находится за пределами
сайтов,
                указанных в конфигурационном файле"
}
```

Статистика — GET /api/statistics

Метод возвращает статистику и другую служебную информацию о состоянии поисковых индексов и самого движка.

Параметры:

Метод без параметров.

Формат ответа:

```
{
```

```
        'result': true,
        'statistics': {
             "total": {
                  "sites": 10,
                  "pages": 436423,
                  "lemmas": 5127891,
                  "isIndexing": true
             },
             "detailed": [
                  {
                       "url": "http://www.site.com",
                       "name": "Имя сайта",
                       "status": "INDEXED",
                       "statusTime": 1600160357,
                       "error": "Ошибка индексации: главная
                                 страница сайта недоступна",
                       "pages": 5764,
                       "lemmas": 321115
                  },
                  ...
             ]
}
```

7. При запуске индексации индексация каждого из сайтов, перечисленных в конфигурационном файле, должна запускаться в отдельных потоках.

8. В начале индексации каждого из сайтов необходимо добавлять или обновлять имеющуюся запись в таблице site: изменять статус на INDEXING, а также удалять все имеющиеся данные по этому сайту. В процессе индексации обновлять дату и время в поле status_time таблицы site на текущее. По завершении индексации изменять статус (поле status) на INDEXED. Если произошла ошибка и индексацию завершить не удалось, изменять статус на FAILED и вносить в поле last_error понятную информацию об ошибке.

Как проверить работу программы

С помощью веб-интерфейса запустите и остановите индексацию сайтов (см. список в этапе 2), а также обновите или добавьте отдельные страницы на этих сайтах и убедитесь, что в базе произошли соответствующие изменения. К примеру, можно проиндексировать какой-то сайт, удалить из базы данные по одной из его страниц (из таблиц page и index), запустить добавление этой

страницы через веб-интерфейс и убедиться, что страница снова добавилась в таблицу page и для неё появились записи в таблице index.

Проверить качество индексации можно на следующем этапе после реализации API поиска.

Этап 7. API поиска и корректные ответы в случае ошибок

Цель

Реализовать API для выполнения поискового запроса, чтобы и веб-интерфейс стал полностью рабочим, и поисковым движком можно было пользоваться.

Что нужно сделать

Необходимо реализовать API-команду поискового запроса в соответствии со следующей спецификацией:

Получение данных по поисковому запросу — GET /api/search

Метод осуществляет поиск страниц по переданному поисковому запросу (параметр query).

Чтобы выводить результаты порционно, также можно задать параметры offset (сдвиг от начала списка результатов) и limit (количество результатов, которое необходимо вывести.

В ответе выводится общее количество результатов (count), не зависящее от значений параметров offset и limit, и массив data с результатами поиска. Каждый результат — это объект, содержащий свойства результата поиска (см. ниже структуру и описание каждого свойства).

Если поисковый запрос не задан или ещё нет готового индекса (сайт, по которому ищем, или все сайты сразу не проиндексированы), метод должен вернуть соответствующую ошибку (см. ниже пример). Тексты ошибок должны быть понятными и должны отражать суть ошибок.

Параметры:

- query — поисковый запрос;

- site — сайт, по которому осуществлять поиск (если не задан, поиск должен происходить по всем проиндексированным сайтам); задаётся в формате адреса, например: `http://www.site.com` (без слэша в конце);
- offset — сдвиг от 0 для постраничного вывода (параметр необязательный, если не установлен, то значение по умолчанию равно нулю);
- limit — количество результатов, которое необходимо вывести (параметр необязательный, если не установлен, то значение по умолчанию равно 20-ти).

Формат ответа в случае успеха:

```
{
     'result': true,
     'count': 574,
     'data': [
          {
               "site": "http://www.site.com",
               "siteName": "Имя сайта",
               "uri": "/path/to/page/6784",
               "title": "Заголовок страницы,
                         которую выводим",
               "snippet": "Фрагмент текста,
                          в котором найдены
                          совпадения, <b>выделенные
                          жирным</b>, в формате HTML",
               "relevance": 0.93362
          },
          ...
     ]
}
```

Формат ответа в случае ошибки:

```
{
     'result': false,
     'error': "Задан пустой поисковый запрос"
}
```

Кроме того, во всех командах API необходимо реализовать корректные ответы в случае возникновения ошибок. Любой метод API может возвращать ошибку, если она произошла. В этом случае ответ должен выглядеть стандартным образом:

```
{
    'result': false,
    'error': "Указанная страница не найдена"
}
```

Такие ответы должны сопровождаться соответствующими статус-кодами. Желательно ограничиться использованием кодов 400, 401, 403, 404, 405 и 500 при возникновении соответствующих им типов ошибок.

Как проверить работу программы

Проиндексировать несколько сайтов из списка, выполнить несколько поисковых запросов по ним и убедиться, что выдаваемые результаты соответствуют ожидаемым.

Этап 8. Размещение на Heroku и в GitHub

Цель

Научиться размещать проекты в публичном доступе для демонстрации на защите дипломов и своим потенциальным работодателям.

Что нужно сделать

1. Разместить созданное приложение на сервисе Heroku и залить на него заполненную базу данных. Для этого сначала на своём компьютере запустите индексацию нескольких сайтов и, когда она завершится, выгрузите дамп базы. Затем разместите ваше приложение на сервисе Heroku (инструкция) и залейте в базу данных ту базу, которую вы создали и заполнили данными о проиндексированных сайтах локально.

2. Разместите исходные коды вашего приложения в публичном доступе в своём GitHub. Создайте в корне репозитория файл “README.md”, в котором:

a. опишите проект;
b. стек используемых технологий;
c. краткую инструкцию по тому, как запустить проект локально (команды, действия, какие переменные среды обязательно требуется задать);
d. ссылку на рабочий проект (в сервисе Heroku).

Для написания красивого и понятного README.md вам поможет руководство **Как написать красивый и информативный README.md**

Дополнительные задачи

Если выполнение основного технического задания вам покажется простым, предлагаем вам дополнительные задачи. Вы можете выбирать любые из них и самостоятельно реализовать, сообщив об этом вашему проверяющему преподавателю:

1. Реализовать вывод результатов поиска в случаях, когда не все леммы найдены. В этом случае выводить дополнительно сообщение наподобие «По запросу **ИСХОДНЫЙ_ЗАПРОС** ничего не найдено. Скорректированный запрос: **ЗАПРОС**».
2. Реализовать поисковый движок с использованием СУБД PostgreSQL вместо MySQL.
3. Реализовать вывод перечня ошибочных страниц на отдельной странице.
4. Реализовать возможность отдельной переиндексации отдельных сайтов.
5. Реализовать возможность переиндексации сайтов без стирания предыдущего индекса на период переиндексации, чтобы поиском можно было пользоваться непрерывно в случае запуска полной переиндексации. Для этого можно использовать временные таблицы в процессе новой индексации и по её завершении заменять старый индекс на новый.
6. Выделить в отдельное поле заголовок h1 на HTML-страницах с весом 0,9 и не учитывать его содержимое в body.
7. Покрыть unit-тестами написанные вами методы, в которых реализованы различные алгоритмы.
8. Авторизация в веб-интерфейсе. Логин и пароль при входе устанавливаются в конфиге (пример простой авторизации).

Рекомендации

1. При работе с базой данных избегайте запросов без ограничений. То есть не получайте все ссылки, все слова. Получайте только то, что вам надо, фильтруйте, сортируйте, ограничивайте количество данных в ответе на уровне базы данных.
2. Избегайте запросов в циклах, при необходимости найдите вариант написания одного запроса.
3. Придерживайтесь принципов «чистого» кода:

    a. Избегайте повторов кода.

    b. Именуйте переменные, методы и классы в соответствии с правилами именования в Java, а также таким образом, чтобы их имена отражали назначение.

    c. Используйте методы не длиннее 30 строк кода и не принимающие более трёх параметров (если требуется больше, то пора их объединять в класс, коллекцию).

    d. Избегайте слишком высокой вложенности кода: старайтесь писать код с вложенностью не более двух уровней:

```
for(int i = 0; i < data.size(); i++) {
     if(data.getItem(i).equals(VALUE)) {
          doSomething(VALUE);
     } else {
          doSomeAnotherAction();
     }
}
```

    e. По возможности упрощайте код. Например, код:

```
public boolean isFinalResult() {
     if(isFinished && result < 10) {
          return true;
     }
     return false;
}
```

можно упростить до:

```
public boolean isFinalResult() {
     return isFinished && result < 10;
}
```

f. Не забывайте для упрощения кода использовать тернарный оператор. Например, вместо кода:

```
public String getCondition(int value) {
     if(value > 10) {
          return “WHERE value = 10”;
     }
     return “WHERE value = ” + value;
}
```

можно написать:

```
public String getCondition(int value) {
     return "WHERE value = " + value > 10 ? 10 :
          value);
}
```

g. Если идёт сравнение, то часто min/max — отличный выбор:

```
public String getCondition(int value) {
     return "WHERE value = " + Math.min(10, value);
          }
```

h. По возможности не пишите комментариев в коде методов. По коду должно быть понятно, что он делает. Если в коде у вас сложное регулярное выражение, назовите переменную так, чтобы было понятно, что это регулярное выражение описывает. Если метод сложный, то декомпозировать на более мелкие части лучше, чем писать многострочные комментарии. Также нежелательно писать JavaDoc в начале проекта, их имеет смысл писать для тех блоков кода, которые не потребуют в дальнейшем изменений, чтобы и документация, и описание методов соответствовали поведению.