Введение

Вы почти закончили прохождение курса и дошли до его завершающего блока, большая часть теории и практики осталась позади. Мы гордимся вашим трудолюбием и упорством! Теперь осталось совсем немного — выполнить и презентовать **итоговый проект**.

Итоговый проект нужен, чтобы вы закрепили все полученные на курсе знания и навыки, самостоятельно создав крупный и работоспособный продукт. Он станет отличным кейсом для вашего портфолио, который можно будет смело показывать потенциальным работодателям.

Опыт самостоятельного выполнения крупных комплексных проектов, таких как данный итоговый проект, даст вам значительные преимущества в поиске работы по профессии.

Желаем успехов в работе над проектом! Уверены, будет интересно! :)

Разработка локального поискового движка по сайту

Перед вами подробное описание будущего проекта. В этом техническом задании есть всё, что вам нужно, чтобы справиться с поставленной задачей.

Мы рассчитываем, что вы, приступая к выполнению этого проекта, прошли курс по языку Java, выполнили в нём практические работы, получили базовый уровень самостоятельности и обладаете следующими навыками:

- можете писать код на Java;
- владеете основными элементами синтаксиса языка Java;
- умеете писать понятный и поддерживаемый код;
- умеете работать с числами, строками, датами и коллекциями;
- умеете создавать собственные классы, абстрактные классы и интерфейсы;
- умеете работать с файлами разных форматов — создавать, читать и удалять при помощи программного кода файлы форматов CSV, HTML и JSON;
- можете разрабатывать многопоточные приложения;
- владеете механизмами работы с исключениями и логами;

- умеете создавать несложные приложения на Spring Boot;
- умеете находить решения проблем в интернете.

Описание проекта

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 помогает посетителям сайта быстро находить информацию, используя поле поиска.

Ваши коллеги уже попробовали существующие поисковые движки: лучшим из них оказался Яндекс Сервер, который можно установить на своём сервере и использовать для поиска по своим сайтам или по одному из них. К сожалению, движок прекратил существовать как отдельный сервис. Ваше руководство решило реализовать собственный поиск и алгоритм, принципы работы которого при необходимости можно менять и развивать.

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

Поисковый движок должен 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

1. В конфигурационном файле перед запуском приложения задаются адреса сайтов, по которым движок должен осуществлять поиск.
2. Поисковый движок должен самостоятельно обходить все страницы заданных сайтов и индексировать их (создавать так называемый индекс) так, чтобы потом находить наиболее релевантные страницы по любому поисковому запросу.
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 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.

На первом этапе нужно загрузить из репозитория заготовку проекта (простого приложения на Spring Boot). В этой заготовке реализованы полезные примеры и фрагменты будущего поискового движка, которые вам предстоит самостоятельно дополнить по настоящему техническому заданию. Мы также реализовали и включили в этот проект frontend-составляющую, чтобы вы могли полностью сконцентрироваться на backend.

Для вашего удобства техническая спецификация проекта собрана в отдельный документ, в котором содержатся:

- описание веб-интерфейса;
- структура таблиц базы данных;
- документация по командам API.

Рекомендации по работе над проектом

- Прочитайте это техническое задание и просмотрите техническую спецификацию проекта. Составьте представление о будущем проекте и его общей структуре.

- Выполнение этого проекта рассчитано суммарно на 40–60 часов работы. Распланируйте свой график таким образом, чтобы выполнить проект небольшими подходами по 2–3 часа и завершить его в течение одного месяца. Такой метод работы будет гораздо эффективнее, чем непрерывная работа в течение длительного времени. Не забывайте отдыхать, чтобы не выгореть и работать с интересом.

- Прежде чем отправлять готовый проект на проверку куратору (а в реальной работе — тестировщику), проверьте его самостоятельно.

Используйте для этого рекомендации по проверке, которые мы даём в конце каждого этапа.

- Попросите нескольких друзей или коллег протестировать проект вместе с вами, если у вас есть такая возможность. Обращайте внимание на все их замечания и комментарии. Возможно, вы захотите внести в продукт небольшие доработки, которые значительно улучшат качество и скорость его работы.

Рекомендации по технической реализации

1. Придерживайтесь принципов «чистого» кода, их соблюдение сделает ваш код более понятным и поддерживаемым:

    a. Избегайте повторов кода.

    b. Именуйте переменные, методы и классы в соответствии с правилами именования в Java, в частности, чтобы их имена отражали назначение.

    c. Используйте методы не длиннее 30 строк кода, не принимающие более трёх параметров (если требуется больше, то их нужно 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 этот 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 комментарии в коде методов. По коду должно быть понятно, что он делает. Если вы используете 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 стоит писать для блоков кода, которые не потребуют изменений в дальнейшем, чтобы и документация, и описание методов соответствовали поведению.

2. «Узким местом» в производительности приложений является работа с базой данных. Рекомендуем при работе с базой данных:

    a. Избегать запросов без ограничений, то есть не получать все ссылки или все слова из какой-либо таблицы и, тем более, из запроса с оператором JOIN. Получайте только те данные, которые вам необходимы. Фильтруйте, сортируйте, ограничивайте количество данных в ответе на уровне базы данных — в самих запросах к ней.

    b. Избегать многократных запросов в циклах. В таких случаях ищите варианты написания единого запроса.

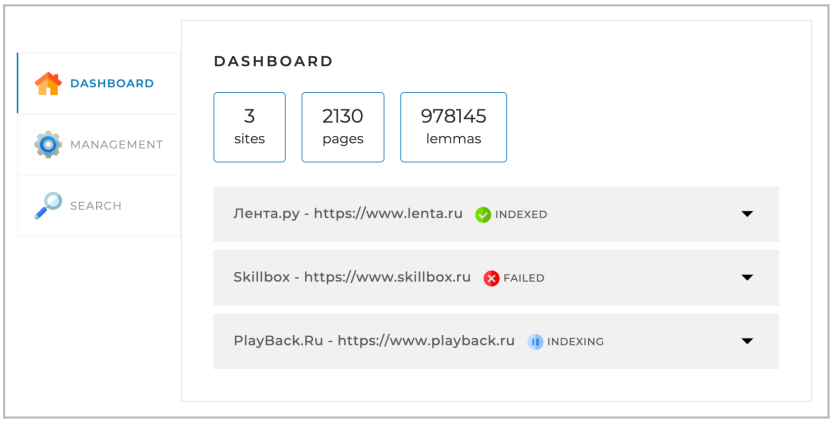
Этап 1. Подготовка

Цель

Скачать, запустить и изучить заготовку проекта, в котором вы будете реализовывать приложение, создать базу данных и подключить к ней проект.

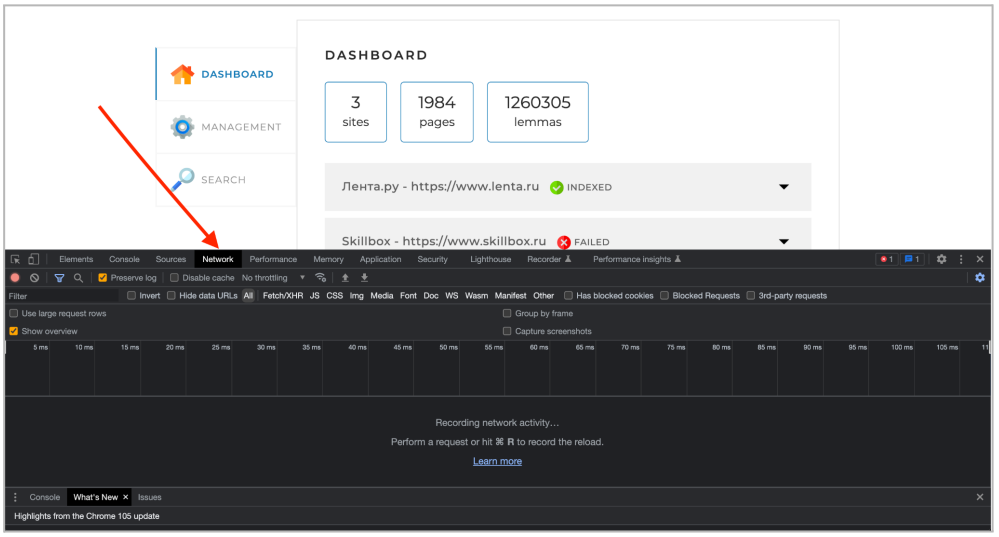
Что нужно сделать

- Установите на свой компьютер JDK и среду разработки IntelliJ IDEA, если они ещё не установлены.
- Загрузите проект-заготовку из Git-репозитория.
- Запустите приложение и откройте его через браузер по адресу: http://localhost:8080/. Вы должны увидеть страницу следующего вида:



- Внимательно изучите структуру этого проекта и то, как он работает.
    - Проект реализован на основе Spring Boot и сборщика Maven, поэтому в файле pom.xml заданы несколько зависимостей, подключающих сам Spring Boot, шаблонизатор Thymeleaf и библиотеку Lombok, предоставляющую возможности использования удобных аннотаций. Вы изучали эти темы в модулях «Разработка веб-приложений» и «Особенности ООП в Java».
    - Все классы находятся в папке **src/main/java/searchengine**. Это необходимо для правильной сборки и запуска приложения с помощью Maven.
    - Запуск приложения начинается с метода main, находящегося в классе **Application**, помеченном аннотацией **@SpringBootApplication**.
    - Поскольку Spring Boot включает в себя не только фреймворк Spring, но и веб-сервер Apache Tomcat, запущенное приложение сразу начинает «слушать» порт 8080 (по умолчанию) и при переходе в браузере по адресу http://localhost:8080/ начинает открываться главная страница приложения. Ниже мы детально разберём, как сделать, чтобы открывалась такая страница.
    - Сама веб-страница (файл **index.html**) размещена в папке **resources/templates**, поэтому её можно подключить в контроллере с помощью шаблонизатора Thymeleaf.
    - В папке **controllers** есть два контроллера: **DefaultController** и **ApiController**. Собственно, в DefaultController создан метод **index** с аннотацией **@RequestMapping("/")**, которая означает, что этот метод должен вызываться при запросе к главной странице приложения.
    - В самом методе **index** написан **return “index”**. Поскольку в проекте работает шаблонизатор Thymeleaf, такой код автоматически подключает и возвращает в качестве ответа код одноимённой веб-страницы (index.html), лежащей в папке **resources/templates**.

- Рассмотрим работу запроса к **ApiController**. Зайдите в браузер на главную страницу приложения, откройте WebInspector (нажатием на странице клавиши <F12>) и вкладку Network:

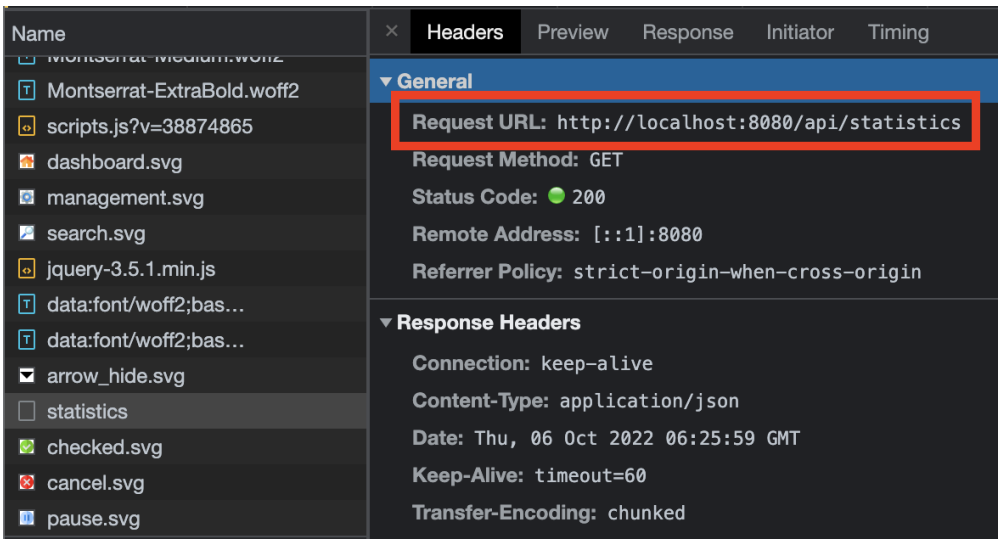


- Обновите веб-страницу. Вы увидите на этой вкладке, как она загружается:

Name	Method	Status	Type	Initiator	Size	Ti...	Waterfall
localhost	GET	200	document	Other	7.7 kB	2...	
Montserrat-SemiBold.woff2	GET	200	font	(index)	(memory ...	1 ...	
Montserrat-Light.woff2	GET	200	font	(index)	(memory ...	0 ...	
Montserrat-Medium.woff2	GET	200	font	(index)	(memory ...	0 ...	
Montserrat-ExtraBold.woff2	GET	200	font	(index)	(memory ...	0 ...	
scripts.js?v=38874865	GET	200	script	(index)	(memory ...	0 ...	
dashboard.svg	GET	200	svg+xml	:8080/:65	(memory ...	0 ...	
management.svg	GET	200	svg+xml	:8080/:65	(memory ...	0 ...	
search.svg	GET	200	svg+xml	:8080/:65	(memory ...	0 ...	
jquery-3.5.1.min.js	GET	200	script	:8080/:161	(memory ...	0 ...	
data:font/woff2;bas...	GET	200	font	fonts.css?v=8236...	(memory ...	0 ...	
data:font/woff2;bas...	GET	200	font	fonts.css?v=8236...	(memory ...	0 ...	
arrow_hide.svg	GET	200	svg+xml	basic.css?v=8236...	(memory ...	0 ...	
statistics	GET	200	xhr	jquery-3.5.1.min.js:2	861 B	6 ...	
checked.svg	GET	200	svg+xml	extra.css?v=8236...	(memory ...	0 ...	
cancel.svg	GET	200	svg+xml	extra.css?v=8236...	(memory ...	0 ...	
pause.svg	GET	200	svg+xml	extra.css?v=8236...	(memory ...	0 ...	

- В начале загружается сама страница (первая строка localhost), затем загружаются её компоненты: шрифты (файлы *.woff2), изображения (файлы *.svg) и скрипты (файлы *.js). В проекте они все размещены в папках внутри папки **resources/static/assets**.

- Кроме того, происходит запрос **statistics**. Нажмите на него, и вы увидите, по какому пути происходит этот запрос:



- Нажмите справа на вкладку Preview, и вы увидите ответ, который приходит от приложения на веб-страницу. Разверните его и внимательно изучите. Он полностью соответствует формату ответа команды API **statistics**, описанному в технической спецификации.

- Когда вы будете реализовывать остальные команды API, мы рекомендуем вам также использовать WebInspector для просмотра запросов веб-страницы к приложению и содержимого получаемых ответов. Это очень удобно.

- Посмотрим, как работает ApiController и как он формирует ответ на запрос **/api/statistics**.

- Обратите внимание, что контроллер помечен двумя аннотациями: @RestController и @RequestMapping("/api"). Первая означает, что этот контроллер будет работать по стандарту REST и, в частности, возвращать ответы в формате JSON. Вторая устанавливает префикс в пути запроса: все запросы, начинающиеся с **/api**, будут направляться на методы этого контроллера.

- В контроллере также создан объект класса StatisticsService (на самом деле это — интерфейс, см. ниже) и в конструкторе ему присваивается передаваемое значение. Это сервис, который отвечает за формирование ответа на запрос **/api/statistics**. Ниже мы рассмотрим его детально.

- В методе контроллера, который помечен аннотацией @GetMapping("/statistics") и, следовательно, отвечает на соответствующий GET-запрос, формируется успешный ответ:

```
return ResponseEntity.ok(...);
```

Это — короткая запись, формирующая ответ в формате JSON с HTTP-кодом 200. В качестве параметра в метод **ok** передаётся результат выполнения метода **getStatistics**, вызываемого у объекта **statisticsService**.

- Теперь давайте посмотрим на сервис. Все сервисы в приложениях на основе Spring Boot принято размещать в папке **services**. В ней размещён интерфейс StatisticsService и класс, который его имплементирует. Но перед этим — немного теории.

- Приложения на основе фреймворка Spring обычно содержат три слоя реализации:

    - **Presentation**. В этом слое находятся контроллеры. Слой «общается» с пользователями, в нашем случае — принимает запросы по API и отдаёт ответы.
    - **Business**. В этом слое реализуется бизнес-логика приложения. Она как раз содержится в классах-сервисах. Этот слой ничего «не знает» о слое Presentation и никак от него не зависит. Если мы изменим что-то в слое Presentation, это никак не повлияет на слой Business.
    - **Data Access**. Этот слой отвечает за хранение данных и, в частности, за подключение к БД. Он также ничего «не знает» о других слоях и не зависит от них.

  Каждый слой занимается только своими задачами и реализация одного слоя не должна зависеть от реализации другого. Все «обязанности» должны быть чётко разделены между соответствующими слоями. Например, контроллер должен получать данные от пользователя и вызывать нужный сервис, не более. Все расчёты и проверки должны происходить в классах-сервисах.

  Если посмотреть на слои данного приложения, то увидим не только сами классы, но ещё и интерфейсы между ними.

  Интерфейсы нужны, чтобы слои приложения не зависели от реализации классов. Это значит, что сервисы и контроллеры могут меняться независимо, заменяться, и это не будет влиять на другие слои.

- Теперь вернёмся к нашему приложению. В классе **StatisticsServiceImpl** сначала происходит подключение данных из файла конфигурации. Файл конфигурации (**application.yaml**) лежит в папке **resources** приложения и содержит список сайтов с их названиями и адресами. Откройте его и посмотрите, какие параметры в нём написаны.

- Чтобы данные из файла конфигурации попали в сервис, в приложении сделано следующее:

    - Реализованы классы **Site** и **SiteList**. Они находятся в папке **config**. У них есть lombok-аннотации **@Setter** и **@Getter**, которые добавляют в классы сеттеры и геттеры для всех полей.

    - Класс **SitesList** помечен аннотациями **@Component** и **@ConfigurationProperties(prefix = "indexing-settings")**. Обратите внимание на значение prefix — это название ключа конфигурации, внутри которого лежит список сайтов. Аннотации приводят к автоматической инициализации объекта этого класса данными из файла **application.yaml**.

    - Класс сервиса **StatisticsServiceImpl** помечен lombok-аннотацией **@RequiredArgsConstructor**, которая добавляет в него конструктор с аргументами, соответствующими неинициализированным final-полям класса. В этом классе только одно такое поле — **sitesList**, поэтому будет добавлен конструктор только с этим аргументом. При создании объекта класса **StatisticsServiceImpl** в конструктор будет передан объект класса SitesList, который, как было сказано выше, автоматически инициализируется на основе данных конфигурации.

- В методе сервиса **getStatistics** происходит постепенная сборка объекта класса **StatisticsResponse** из данных о сайтах, а также некоторой случайной информации и двух заданных в начале метода массивов.

- Обратите внимание, что все классы, на основе которых сервисом собирается итоговый объект, размещены в папке **dto** и подпапке **statistics**. Аббревиатура DTO расшифровывается как Data Transfer Object, что в переводе с английского означает «Объект передачи данных». Все объекты запросов и ответов, которые вы будете

использовать в приложении, а также объектов, из которых они будут построены, необходимо хранить в папке **dto**.

- Придерживайтесь заданной нами структуры проекта: папок, интерфейсов и классов, а также принципов их именования. Это поможет вам быстрее создать итоговый проект и легко в нём ориентироваться, а нам — разобраться в написанном вами коде.

- 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 базе данных. Это может быть пользователь root, который имеет доступ ко всем базам данных (создаётся при установке MySQL-сервера), а может быть отдельный пользователь, имеющий доступ только к созданной вами базе данных, на ваше усмотрение.

- Подключите зависимости для работы с базой данных:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
   <scope>runtime</scope>
</dependency>
```

- Создайте конфиг application.yaml в корне проекта и пропишите в нём в явном виде порт, по которому будет доступно веб-приложение, а также данные доступа к MySQL-серверу:

```
server:
 port: 8080

spring:
 datasource:
   username: user
   password: pass
   url:
jdbc:mysql://localhost:3306/search_engine?useSSL=false&requir
eSSL=false&allowPublicKeyRetrieval=true
 jpa:
   properties:
     hibernate:
```

```
    dialect: org.hibernate.dialect.MySQL8Dialect
  hibernate:
    ddl-auto: update
  show-sql: true
```

- В проекте уже создан конфигурационный файл **application.yaml** в папке **resources**. Скопируйте его содержимое (перечень сайтов) в свой конфиг, а сам файл удалите из папки **resources**.

- Запустите (или перезапустите) приложение и убедитесь, что оно запускается без ошибок, и главная страница по-прежнему открывается в браузере.

Этап 2. Система обхода веб-страниц

Цель

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

Что нужно сделать

- На этом этапе — систему, которая будет по команде обходить веб-страницы сайтов и сохранять их содержимое в таблицу в базе данных.

- Создайте в проекте папку model и в ней — классы, которые будут соответствовать таблицам **site** и **page** в базе данных. Структура таблиц описана в технической спецификации. Создайте их по правилам, которые вы изучали в модуле курса «Работа с MySQL в Java». В частности, не забудьте про аннотации @Entity, @Id, @GeneratedValue, @ManyToOne и @JoinColumn. Для enum-поля создайте отдельный Enum (его можно поместить в ту же папку model). Различные типы текстовых полей обозначайте аннотацией @Column:

```
@Column(columnDefinition = "ENUM('INDEXING', 'INDEXED')")
@Column(columnDefinition = "VARCHAR(255)")
@Column(columnDefinition = "TEXT")
@Column(columnDefinition = "MEDIUMTEXT")
```

- После создания классов для таблиц базы данных запустите приложение и убедитесь, что в базе данных появились соответствующие таблицы и что они имеют верную структуру.

- Обратите внимание, что по полю **path** в таблице **page** должен быть установлен индекс, чтобы поиск по нему был быстрым, когда в нём много ссылок. Узнать об индексах можно в курсе Язык запросов SQL.

- В качестве примера того, как создаются и используются классы для работы с базой данных, мы подготовили для вас специальный проект. Можете загрузить его по ссылке и ознакомиться с его структурой. Для запуска проекта прочитайте инструкции, приведённые в файле README.md, который лежит в корне.

- Создайте в контроллере метод запуска индексации **startIndexing** в соответствии с технической спецификацией. В этом методе пропишите запуск сервиса индексации сайтов.

- В сервисе индексации сайтов пропишите код, который будет брать из конфигурации приложения список сайтов и по каждому сайту:

    - удалять все имеющиеся данные по этому сайту (записи из таблиц **site** и **page**);

    - создавать в таблице **site** новую запись со статусом **INDEXING**;

    - обходить все страницы, начиная с главной, добавлять их адреса, статусы и содержимое в базу данных в таблицу **page**;

    - в процессе обхода постоянно обновлять дату и время в поле **status_time** таблицы **site** на текущее;

    - по завершении обхода изменять статус (поле status) на **INDEXED**;

    - если произошла ошибка и обход завершить не удалось, изменять статус на **FAILED** и вносить в поле **last_error** понятную информацию о произошедшей ошибке.

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

- **ОБРАТИТЕ ВНИМАНИЕ!** Задачу по обходу веб-страниц сайта вы уже решали в практической работе после модуля «Многопоточность». Рекомендуем использовать уже написанный вами код.

- Обход каждого из сайтов, перечисленных в конфигурационном файле, должен запускаться в отдельном потоке.

- Поскольку на страницах сайта могут находиться повторяющиеся ссылки, сервис должен проверять запросом к базе данных, заходил ли он по каждой очередной ссылке или нет.

- Код ответа необходимо научиться определять самостоятельно, воспользовавшись документацией к библиотеке JSOUP.

- После работы программы в таблице должны оказаться ссылки на все страницы заданных в конфиге сайтов без повторов.

- Чтобы индексируемый сайт считал запросы от вашего приложения обычными посещениями пользователей, можно, например, обращаться к нему при помощи фейкового User-Agent и фейкового referrer так:

```
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 самостоятельно.

  Также рекомендуем вынести значения User-Agent и referer в конфигурацию вашего приложения и считывать их оттуда.

- Некоторые сайты могут быть защищены от слишком частых запросов: их необходимо обходить аккуратно, с задержками в 0,5–5 секунд между запросами.

- Реализуйте также функцию остановки обхода сайтов — команду API **stopIndexing** в соответствии с технической спецификацией. Она должна останавливать все потоки и записывать в базу данных для всех сайтов, страницы которых ещё не удалось обойти, состояние **FAILED** и текст ошибки «Индексация остановлена пользователем».

Как проверить работу программы

Пропишите в конфигурации приложения хотя бы два сайта 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 этих сайтов и их страниц. Например, для сайта http://www.playback.ru/ в таблицах должны быть данные следующего вида:

site

id	status	status_time	last_error	url	name
1	INDEXED	2022-09-25 10:15:34		http://www.playback.ru/	PlayBack.Ru

page

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

*В поле content должны быть коды соответствующих страниц сайта.

Этап 3. 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 короткое время (обычно доли секунды) по сравнению с обычным поиском перебором по всему массиву информации. Вспомните разницу в скорости между поиском перебором в простом массиве и бинарным поиском 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.

В рамках этого этапа вам будет необходимо подключить так называемый лемматизатор — библиотеку, которая позволяет получать леммы слов — их исходные формы. Например, для существительных — 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 будем использовать лемматизатор (ссылка приведена для справки, использовать коды из данного репозитория вам не нужно), который используется в широко используемом поисковом движке Apache Solr (он, в частности, используется в качестве поискового движка Википедии).

Цель

Реализовать систему индексации страниц сайта, которая позволит подсчитывать встречающиеся на страницах сайта слова (точнее, их леммы) и в

дальнейшем по поисковому запросу определять наиболее релевантные страницы.

Что нужно сделать

На втором этапе вы уже реализовали систему, которая умеет обходить страницы сайтов по ссылкам в многопоточном режиме. Теперь вам предстоит написать код, который будет извлекать из текстов страниц слова, преобразовывать их в леммы, считать количество вхождений каждой леммы в текст и сохранять эту информацию в базу данных.

- Создайте классы для ещё двух таблиц базы данных (**lemma** и **index**) в соответствии с технической спецификацией.

- Мы разработали и опубликовали для вас репозиторий с исходными кодами и сгенерированными JAR-библиотеками, предназначенными для лемматизации слов — получения из каждого слова его исходной формы. Ниже описаны шаги по подключению этих библиотек к вашему проекту.

- Добавьте в файл **pom.xml** блок с указанием пути к созданному нами репозиторию (из него будут подгружаться зависимости):

```
<repositories>
  <repository>
    <id>skillbox-gitlab</id>

<url>https://gitlab.skillbox.ru/api/v4/projects/263574/packages/maven</url>
  </repository>
</repositories>
```

- Добавьте также в файл **pom.xml** в раздел с зависимостями все библиотеки из данного репозитория:

```
<dependency>
  <groupId>org.apache.lucene.morphology</groupId>
  <artifactId>morph</artifactId>
  <version>1.5</version>
</dependency>
<dependency>
  <groupId>org.apache.lucene.analysis</groupId>
  <artifactId>morphology</artifactId>
  <version>1.5</version>
</dependency>
```

```
<dependency>
  <groupId>org.apache.lucene.morphology</groupId>
  <artifactId>dictionary-reader</artifactId>
  <version>1.5</version>
</dependency>
<dependency>
  <groupId>org.apache.lucene.morphology</groupId>
  <artifactId>english</artifactId>
  <version>1.5</version>
</dependency>
<dependency>
  <groupId>org.apache.lucene.morphology</groupId>
  <artifactId>russian</artifactId>
  <version>1.5</version>
</dependency>
```

- Теперь вам необходимо указать токен для доступа к данному Maven-репозиторию, поскольку GitLab запрещает публичный доступ к библиотекам. Для указания токена найдите или создайте файл settings.xml.

    - В Windows он располагается в директории `C:/Users/<Имя вашего пользователя>/.m2`
    - В Linux — в директории `/home/<Имя вашего пользователя>/.m2`
    - В macOs — по адресу `/Users/<Имя вашего пользователя>/.m2`

Если файла **settings.xml** нет, создайте его и вставьте в него код:

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 https://maven.apache.org/xsd/settings-1.0.0.xsd">

<servers>
    <server>
      <id>skillbox-gitlab</id>
      <configuration>
        <httpHeaders>
          <property>
            <name>Private-Token</name>
```

```
            <value>wtb5axJDFX9Vm_W1Lexg</value>
          </property>
        </httpHeaders>
      </configuration>
    </server>
  </servers>
</settings>
```

Если файл уже есть, но в нём нет блока `<servers>`, то добавьте в него только этот блок. Если этот блок в файле есть, добавьте внутрь него блок `<server>` из фрагмента кода, приведённого выше.

В блоке `<value>` находится уникальный токен доступа. Если у вас возникнет «401 Ошибка Авторизации» при попытке получения зависимостей, возьмите актуальный токен доступа из документа по ссылке. Если он не подойдёт, запросите у куратора актуальный токен доступа, пришлите куратору файл settings.xml и логи обновления зависимостей.

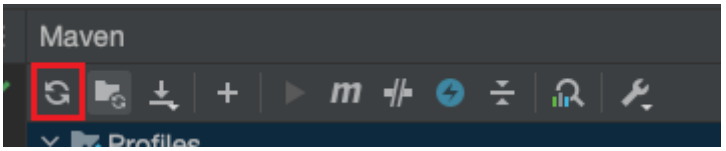
Обязательно почистите кэш maven. Самый надёжный способ — удалить директорию:

- Windows: `C:\Users\<user_name>\.m2\repository`
- macOs: `/Users/<user_name>/.m2/repository`
- Linux: `/home/<user_name>/.m2/repository`

`<user_name>` — имя пользователя, под которым вы работаете.

После этого снова попробуйте обновить данный из pom.xml.

Затем обновите зависимости в проекте при помощи <Ctrl+Shift+O> или <⌘⇧I>:



Пример проекта с подключенными библиотеками, настроенным pom.xml и примером settings.xml — в отдельном репозитории.

- После подключения зависимостей 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
```

- Создайте класс с методом, который будет принимать в качестве параметра текст и возвращать перечень лемм для каждого слова в этом тексте (за исключением междометий, союзов, предлогов и частиц — см. ниже) и количество упоминаний каждой такой леммы в переданном тексте. Ниже описаны детали реализации данного метода.

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
```

- **ОБРАТИТЕ ВНИМАНИЕ!** Разделение текстов на слова вы уже реализовывали в одном из заданий к модулю «Строки» курса

«Java-разработчик с нуля». Рекомендуем использовать в этом проекте уже написанный вами ранее код.

- Метод должен возвращать HashMap<String, Integer>, в котором ключами будут леммы, а значениями — их количества в переданном тексте.

- 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
```

- Если с реализацией кода возникают сложности, посмотрите наш пример такого кода.

- Также можете посмотреть пример реализации приложения с использованием данных классов и библиотек в проекте, который мы вам рекомендовали ранее. Можете загрузить его к себе на компьютер и собрать в соответствии с инструкциями, приведёнными в файле README.md (файл лежит в корне проекта).

- В этом же классе реализуйте метод, который будет очищать код веб-страниц от HTML-тегов.

- Реализуйте функцию индексации отдельной веб-страницы. Для этого сначала реализуйте метод контроллера для команды API **indexPage** в соответствии с технической спецификацией.

- Напишите код, который будет получать HTML-код переданной веб-страницы, сохранять его в базу данных в таблицу **page**, преобразовывать в набор лемм и их количеств, а затем сохранять эту информацию в таблицы **lemma** и **index** базы данных следующим образом:

    - Леммы должны добавляться в таблицу **lemma**. Если леммы в таблице ещё нет, она должна туда добавляться со значением **frequency**, равным 1. Если же лемма в таблице уже есть, число **frequency** необходимо увеличить на 1. Число **frequency** у каждой леммы в итоге должно соответствовать количеству страниц, на которых эта лемма встречается хотя бы один раз.

    - Связки лемм и страниц должны добавляться в таблицу **index**. Для каждой пары «лемма-страница» в этой таблице должна создаваться одна запись с указанием количества данной леммы на страницы в поле rank.

- Проверьте работу индексации на отдельной странице, указав путь к ней в веб-интерфейсе вашего приложения и запустив её индексацию. Не забудьте, что при добавлении страницы в базу данных она должна привязываться к записи в таблице **site**, которая либо уже должна там находиться, либо должна быть создана на основе одного из пунктов списка сайтов в конфигурации вашего приложения.

- В случае попытки индексации страницы с какого-то другого сайта команда API должна выдавать ошибку в соответствии с технической спецификацией. Убедитесь в этом в веб-интерфейсе вашего приложения.

- В случае, если переданная страница уже была проиндексирована, перед её индексацией необходимо удалить всю информацию о ней из таблиц **page**, **lemma** и **index**.

- Допишите код обхода веб-страниц, который вы создавали на втором этапе таким образом, чтобы каждая полученная веб-страница преобразовывалась в набор лемм и их количеств, и эта информация также сохранялась в таблицы **lemma** и **index** базы данных по описанному выше алгоритму.

- Коды страниц, при получении которых HTTP-ответ был ошибочным (с кодами 4xx или 5xx), индексировать не нужно.

- Запустите индексацию пары сайтов из списка выше и убедитесь, что всё работает.

- Перепишите также метод API **statistics**. Он должен рассчитывать статистику и возвращать её в формате, описанном в технической спецификации. Формировать статистику рекомендуем, используя классы, уже созданные в пакете **dto.statistics**. Статистика должна отображаться на главной странице сервиса.

Как проверить работу программы

С помощью веб-интерфейса запустите и остановите индексацию сайтов (см. список в этапе 2) и убедитесь в том, что после индексации хотя бы пары сайтов (из списка выше, см. этап 2) данные в базе, в том числе в таблицах **lemma** и **index**, выглядят правдоподобно, а сама индексация при этом завершается без ошибок и для всех страниц с неошибочными HTTP-кодами (не 4xx и не 5xx) есть записи в таблице **index**. Также на главной странице вашего сервиса должна отображаться правдоподобная статистика.

Обновите или добавьте отдельные страницы на проиндексированных 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 появилась в таблице **page** и для неё появились записи в таблице **index**.

Проверить корректность индексации можно только после реализации системы поиска (см. следующий этап).

Этап 4. 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.

- Создайте метод для команды API **search** в соответствии с технической спецификацией. Метод должен выполнять следующий алгоритм:

    - Разбивать поисковый запрос на отдельные слова и формировать из этих слов список уникальных лемм, исключая междометия, союзы, предлоги и частицы. Используйте для этого код, который вы уже писали в предыдущем этапе.

    - Исключать из полученного списка леммы, которые встречаются на слишком большом количестве страниц. Поэкспериментируйте и определите этот процент самостоятельно.

    - Сортировать леммы в порядке увеличения частоты встречаемости (по возрастанию значения поля frequency) — от самых редких до самых частых.

    - По первой, самой редкой лемме из списка, находить все страницы, на которых она встречается. Далее искать соответствия следующей леммы из этого списка страниц, а затем повторять операцию по каждой следующей лемме. Список страниц при этом на каждой итерации должен уменьшаться.

    - Если в итоге не осталось ни одной страницы, то выводить пустой список.

    - Если страницы найдены, рассчитывать по каждой из них релевантность (и выводить её потом, см. ниже) и возвращать.

    - 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 релевантности для конкретной страницы на максимальную абсолютную релевантность среди всех страниц данной поисковой выдачи:

$$R_{rel} = 7,3 / 10,3 = 0,7087$$

- 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 (см. ниже);
  - relevance — релевантность страницы (см. выше формулу расчёта).

- Сниппеты — фрагменты текстов, в которых найдены совпадения, для всех страниц должны быть примерно одинаковой длины — такие, чтобы на странице с результатами поиска они занимали примерно три строки. В них необходимо выделять жирным совпадения с исходным поисковым запросом. Выделение должно происходить в формате HTML при помощи тега <b>. Алгоритм получения сниппета из веб-страницы реализуйте самостоятельно.

- Обратите внимание, что метод поиска должен учитывать, по каким сайтам происходит этот поиск — по всем или по тому, который выбран в веб-интерфейсе в выпадающем списке.

Как проверить работу программы

Проиндексируйте несколько сайтов из списка выше (см. этап 2) и запустите поиск по фразам, которые встречаются на определённых страницах каждого сайта и не встречаются на других сайтах, а затем сравните результат с ожидаемым.

При запросе фразы, существующей на странице сайта, эта страница должна быть найдена и должна иметь высокую релевантность, а при запросе несуществующей на сайте фразы должен быть выдан пустой список.

Этап 5. Публикация проекта на GitHub

Цель

Разместить разработанный проект в публичном доступе для итоговой презентации кураторам и демонстрации своим потенциальным работодателям.

Что нужно сделать

- Разместите исходные коды вашего приложения в публичном доступе в своём GitHub. Создайте в корне репозитория файл **README.md**, в котором:
    - опишите проект;
    - стэк используемых технологий;
    - инструкцию по локальному запуску проекта — последовательность команд и действий.
- В написании красивого и понятного файла **README.md** вам поможет наше руководство «Как написать красивый и информативный README.md».