

Министерство образования и науки Российской Федерации  
Федеральное государственное бюджетное образовательное  
учреждение высшего образования  
«Иркутский государственный университет»  
(ФГБОУ ВО «ИГУ»)

Институт математики, экономики и информатики  
Кафедра алгебраических и информационных систем

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**  
**по направлению «09.03.03 Прикладная информатика»**  
**профиль подготовки «Информационная сфера»**

**РЕАЛИЗАЦИЯ ПОИСКОВОГО СЕРВИСА ДЛЯ ИОС «DOMIC»**

Студент 4 курса очного отделения  
Группа 02461–ДБ  
Левчук Максим Валерьевич

Руководитель:  
к.ф.-м.н., доцент  
\_\_\_\_\_ Зинченко А.С.

Допущена к защите  
Зав. кафедрой, д.ф.-м.н., профессор  
\_\_\_\_\_ Пантелеев В.И.

Иркутск 2017

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>4</b>
<b>Глава 1. Формирование требований к поисковому сервису</b>	<b>7</b>
1.1. Постановка задачи . . . . .	7
1.2. Обзор моделей информационного поиска . . . . .	8
1.3. Анализ инструментов осуществления поиска . . . . .	12
1.4. Требования к поисковому сервису . . . . .	14
<b>Глава 2. Технологии разработки</b>	<b>16</b>
2.1. Язык программирования Java . . . . .	16
2.2. Язык программирования Javascript . . . . .	18
2.3. Среда разработки IntelliJ IDEA . . . . .	19
2.4. Система управления базами данных MySQL . . . . .	20
2.5. Поисковая машина Apache Solr . . . . .	21
2.6. Библиотеки Apache Tika . . . . .	22
2.7. Фреймворк Spring . . . . .	23
2.8. Библиотека Hibernate . . . . .	23
<b>Глава 3. Описание разработанного программного решения</b>	<b>26</b>
3.1. Архитектура приложения . . . . .	26
3.2. Настройка Apache Solr . . . . .	28
3.3. Модуль извлечения данных . . . . .	35
3.4. Разработка клиентского приложения . . . . .	38
3.5. Инструкция по развёртыванию сервиса . . . . .	46
<b>ЗАКЛЮЧЕНИЕ</b>	<b>48</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>49</b>

<b>ПРИЛОЖЕНИЕ 1. Фрагмент ER-диаграммы базы данных ИОС «DOMIC»</b>	<b>52</b>
<b>ПРИЛОЖЕНИЕ 2. Структура классов модуля JDoX</b>	<b>53</b>

# ВВЕДЕНИЕ

В настоящее время, когда происходит стремительное развитие высоких технологий, постоянно создаются всё новые информационные системы. Среди них трудно отыскать такие, которые не зависели бы от той или иной текстовой информации, в связи с чем спрос на качественные средства обработки неструктурированных текстовых данных растёт экспоненциально.

Одной из таких систем является информационно-образовательная среда «DOMIC» (<http://domic.isu.ru>). Данная информационная система была разработана и запущена в 2005 году. С момента запуска и до настоящего времени преподаватели, использующие ИОС «DOMIC» в учебном процессе, создали в нём более 4000 учебных единиц. Каждая такая учебная единица может включать несколько текстовых документов, причём некоторые из них могут быть связаны между собой ссылками (в пределах одной учебной единицы). Однако возможностей для оперативного поиска и извлечения данных из этой коллекции документов нет.

Пользователи ИОС «DOMIC» могут получить доступ к документам лишь непосредственно из учебных единиц. В среде не реализована функциональность для поиска документов, поэтому задача поиска необходимого документа может оказаться довольно сложной.

Таким образом, **целью** выпускной квалификационной работы является разработка поискового сервиса для ИОС «DOMIC». Этот сервис должен предоставлять пользователям механизмы для извлечения документов из коллекции ИОС «DOMIC», их обработки и осуществления поиска по ним.

Исходя из данной цели, были поставлены следующие **задачи**:

- 1) разработать модуль для извлечения содержимого документов коллекции ИОС «DOMIC», который будет корректно обрабатывать текстовые документы, в том числе связанные между собой ссылками;

- 2) выбрать и настроить поисковую машину, способную быстро и качественно обрабатывать десятки тысяч документов;
- 3) реализовать клиентское приложение, с помощью которого пользователи смогут получить доступ к коллекции документов ИОС «DOMIC», осуществлять по ней поиск и получать результаты в удобном виде.

**Новизна** данной работы заключается в том, что разработанный программный продукт добавляет новую функциональность в ИОС «DOMIC», которую не смогло бы обеспечить в полном объёме ни одно готовое решение. Таким образом, становится очевидной необходимость разработки поискового сервиса для ИОС «DOMIC».

Реализованное программное решение предназначено в основном для преподавателей ИОС «DOMIC». А его **практическая значимость** состоит в том, что преподаватели получают новый (в рамках данной информационной системы) инструмент, благодаря которому они могут довольно просто находить результаты своих более ранних работ, а также знакомиться с наработками коллег. Перечисленные возможности могут положительно повлиять как на производительность труда самих преподавателей, так и на прогресс в обучении их студентов.

**Структура выпускной квалификационной работы.** Данная работа изложена на 53 страницах. Она состоит из введения, трёх глав, заключения, списка использованных источников и 3 приложений, в том числе одного — программного кода — на диске. Список литературы состоит из 21 наименования. Работа проиллюстрирована 8 рисунками.

Первая глава содержит описание поставленной задачи, обзор моделей информационного поиска, анализ существующих инструментов поиска и требования к разрабатываемой системе.

Во второй главе приводится описание всех основных технологий, на основе

которых разрабатывался поисковый сервис.

Третья глава содержит детали реализации сервиса, а также описания настройки некоторых его компонентов.

В заключении подводятся итоги проделанной работы, перечисляются полученные результаты.

# Глава 1. Формирование требований к поисковому сервису

## 1.1. Постановка задачи

Разрабатываемый программный продукт является функциональным дополнением для ИОС «DOMIC», поэтому рассмотрим компоненты, из которых состоит данная информационная система.

В основе ИОС «DOMIC» лежит база данных MySQL, в которой хранятся данные о различных сущностях, среди которых можно выделить следующие:

- учебные единицы;
- модули;
- дисциплины;
- курсы.

В Приложении 1 приведён фрагмент ER-диаграммы базы данных ИОС «DOMIC», в котором представлены сущности, использующиеся для построения поискового индекса.

Также имеется файловый репозиторий, в котором хранятся все документы, которые когда-либо были добавлены в учебные единицы преподавателями, работающими с ИОС «DOMIC».

База данных и файловый репозиторий связаны следующим образом: в таблице StudyEntity (учебная единица) имеется текстовое поле urlFirstPage, в котором хранятся ссылки на главные документы каждой учебной единицы, которые находятся в файловом репозитории; идентификатор (поле idStudyEntity) — это в то же время название папки, в которой находится данный файл (а также, возможно, другие файлы, связанные с главным документом).

Основная проблема заключается в том, что в ИОС «DOMIC» не существу-

ет механизмов для более оперативной работы с учебными единицами, отсутствует возможности поиска учебных единиц по контенту, либо кластеризации их по заданным критериям.

Исходя из потребности в данных механизмах было решено разработать поисковый сервис для ИОС «DOMIC», позволяющий осуществлять поиск по его коллекции документов.

## **1.2. Обзор моделей информационного поиска**

Областью исследований данной выпускной квалификационной работы является задача информационного поиска.

Информационный поиск (IR — Information Retrieval) — это процесс поиска в большой коллекции (хранящейся, как правило, в памяти компьютеров) некоего неструктурированного материала (обычно документа), удовлетворяющего некоторым информационным потребностям [21, 18].

К информационному поиску также относятся такие задачи, как навигация пользователей по коллекции документов и их фильтрация, а также дальнейшая обработка найденных документов.

Самой распространённой задачей информационного поиска является разработка систем, выполняющих поиск по произвольному запросу. Целью данных систем является поиск в коллекции документов, которые являются наиболее релевантными по отношению к любым информационным потребностям, которые сообщаются системе при помощи запросов пользователей.

Информационная потребность — это то, о чём пользователь хочет знать больше всего. Её необходимо отличать от запроса, то есть от того, что пользователь вводит в поисковую строку, пытаясь выразить свою информационную потребность. Документ называется релевантным, если, с точки зрения пользователя, он содержит ценную информацию, удовлетворяющую его информаци-



онную потребность.

В основе традиционных методов информационного поиска лежат три главных подхода — три модели [20]:

- булевская;
- вероятностная;
- векторная.

Рассмотрим каждую из этих моделей на основании трёх ключевых аспектов: документ, запрос и функция документа соответствия запросу.

1. Формат представления документа. Под документом понимается некоторый объект, содержащий информацию в зафиксированном виде. Документы могут содержать тексты на естественном или формализованном языке, изображения, звуковую информацию и так далее.
2. Формат представления запроса. Запрос — это формализованный способ выражения информационных потребностей пользователя системы. Для этого используется язык поисковых запросов, синтаксис которых варьируется от системы к системе.
3. Функция соответствия документа запросу. Степень соответствия запроса и найденного документа (релевантность) — субъективное понятие, поскольку результаты поиска, уместные для одного пользователя, могут быть неуместными для другого.

### **Булева модель информационного поиска**

Модель булева поиска — это модель информационного поиска, с помощью которой можно обрабатывать любой запрос, имеющий вид булева выражения, то есть выражения, в котором термы используются в сочетании с операциями AND, OR и NOT. В рамках этой модели документ рассматривается просто как множество слов.

Булева модель базируется на теории множеств и математической логике. Популярность этой модели связана, прежде всего, с простотой её реализации, которая позволяет индексировать и выполнять поиск в больших массивах документов.

В рамках булевой модели документы и запросы представляются в виде множества термов — ключевых слов и устойчивых словосочетаний. Каждый терм представлен как булева переменная: 0 (терм из запроса не присутствует в документе) или 1 (терм из запроса присутствует в документе).

Такая модель иногда используется во внутренних корпоративных системах поиска, базах данных. Основным недостатком этой модели является крайняя жёсткость и непригодность для ранжирования. Если слово, указанное в запросе, присутствует в документе, то этот документ считается найденным, в противном случае — не найденным. Не будет найден и документ, в котором встречаются только синонимы слова, указанного в запросе, в случае, когда само слово в документе не встречается.

### **Вероятностная модель информационного поиска**

В этой модели релевантность рассматривается как вероятность того, что данный документ может оказаться интересным пользователю. При этом подразумевается наличие уже существующего первоначального набора релевантных документов, выбранных пользователем или полученных автоматически при каком-нибудь упрощённом предположении. Вероятность оказаться релевантным для каждого следующего документа рассчитывается на основании соотношения встречаемости термов в релевантном наборе и в остальной части коллекции.

Документом в данной модели считается множество слов в виде обычного булевого вектора  $D = \{d_1, \dots, d_n\}$ , где  $n$  — количество всех термов, а  $d_i$  может принимать значения из множества  $\{0, 1\}$ . Запросом является множество слов.

Соответствие документа запросу строится следующим образом: пусть для каждого фиксированного запроса  $Q_k$  имеются распределения вероятностей на всех документах «быть релевантным» и «быть нерелевантным» запросу  $Q_k$ . Обозначается это соответственно как  $P(R|Q_k, D)$  и  $P(\bar{R}|Q_k, D)$ . Тогда функцией соответствия будет отношение двух величин:

$$\frac{P(R|Q_k, D)}{P(\bar{R}|Q_k, D)}.$$

Недостатками данной модели является низкая вычислительная масштабируемость (то есть резкое снижение эффективности при росте объёмов данных), а также необходимость постоянного обучения системы.

### **Векторная модель информационного поиска**

В рамках этой модели запросы представляют собой обычный текст, то есть одно или несколько слов. Нет необходимости использовать строгие языковые конструкции с операторами: система сама определяет какие документы лучше других удовлетворяют этим запросам. Также в тексте запроса допускается наличие одновременно двух и более одинаковых слов.

При обработке запроса он разбивается на отдельные термы, которые проходят несколько этапов препроцессинга: удаление стоп-слов (предлогов, союзов, часто употребляемых слов), добавление синонимов, стемминг (удаление окончаний и, возможно, суффиксов слов).

Каждый терм представляется в виде координаты в многомерном векторном пространстве, которая говорит о том, насколько «сильно» данный терм входит в документ. То есть таким образом, каждый документ — это множество из  $n$  действительных чисел.

Степень соответствия слова  $i$  документу  $j$  может быть задана в виде матрицы  $M$ , которая определяется по следующей формуле:

$$M_{ij} = TF_{ij} \cdot IDF_i,$$

где  $TF_{ij}$  (Term Frequency, частота терма) — относительная доля слова  $i$  в документе  $j$ ;  $IDF_i$  (Inversed Document Frequency, обратная документная частота) — величина, обратная количеству документов, которые содержат слово  $i$ .

Мера релевантности считается следующим образом. Запрос представляется в виде вектора с координатами 0 или 1:  $Q = \langle t_3 \text{ AND } t_5 \rangle = \{0, 0, 1, 0, 1, 0, \dots, 0\}$ . Каждый документ представляется в виде набора таких координат: много нулевых координат (это те термы, которые не встречаются в документе) и несколько ненулевых координат. Мера релевантности  $R(Q, D)$  — косинус угла  $\alpha$  между вектором запроса  $Q$  и документом  $D_j$ :

$$R(Q, D) = \cos \alpha = \frac{QD_j}{|Q||D_j|}.$$

Также необходима нормализация для того, чтобы уравнивать веса документов с разным количеством слов.

На основе рассмотренных моделей информационного поиска (преимущественно на основе векторной) реализовано большое количество библиотек поиска, поисковых серверов, благодаря чему обычным программистам не нужно самостоятельно реализовывать все алгоритмы, можно просто взять готовый инструмент, который позволит осуществлять индексирование (обработку содержимого и представление его в оптимальном для программы виде) и поиск по различным коллекциям документов.

### 1.3. Анализ инструментов осуществления поиска

Существует немало библиотек и серверов, предоставляющих функциональность для индексации и поиска в различных системах, у каждого такого

инструмента есть свои преимущества и недостатки. Нами был проведен сравнительный анализ по нескольким характеристикам трех таких инструментов, которые входят в десятку самых популярных — Apache Solr [2], Sphinx [10], Xapian [14]. Результаты сравнения представлены в таблице 1.

Таблица 1

Сравнительный анализ инструментов поиска

Характеристика	Solr	Sphinx	Xapian
Скорость индексации	2.75 МБ/с	4.5 МБ/с	1.36 МБ/с
Скорость поиска (ср. / макс.)	25 мс / 212 мс	7 мс / 75 мс	14 мс / 135 мс
Размер индекса	20 %	30 %	200 %
Реализация	Сервер	Сервер	Библиотека
Интерфейс	Веб-сервис	API, SQL	API
Стеммеры	31	15	15
Стоп-слова, синонимы	Да	Да	Да
Soundex	Да	Да	Нет
Расширяемость	Высокая	Средняя	Низкая

Скорость поиска измерялась при размере индекса в 1000 документов. Размер индекса представлен в процентах от размера исходных данных.

В итоге, при сравнении данных решений, были сделаны следующие выводы.

- Самая простая и быстрая поисковая машина — Sphinx. Однако его недостаток состоит в том, что он не очень хорошо работает с индексом: инструмент адекватно работает только, если из индекса ничего не удалять.
- Xapian — также довольно быстрая библиотека по части поиска, однако по скорости индексации несколько отстаёт от аналогов. К тому же у этого ре-

шения получается слишком большой индекс — целых 200 % относительно размера исходных данных.

- Solr — довольно быстрый, эффективный и очень расширяемый поисковый сервер. Поддерживает множество различных функций, по этому показателю значительно обходит своих конкурентов.

На основании данных выводов в качестве поисковой машины для разрабатываемого в данной работе поискового сервиса был выбран Solr. В пользу этого инструмента выступает и тот факт, что для него существует достаточно большое количество литературы и документации, в том числе и на русском языке. К тому же в случае необходимости для Solr можно написать свои компоненты на языке программирования Java и довольно просто их подключить.

## 1.4. Требования к поисковому сервису

Разрабатываемый поисковый сервис должен удовлетворять следующим требованиям:

1. Необходим программный модуль, который будет извлекать текстовое содержимое из документов различных форматов, при этом содержимое должно извлекаться корректно из файлов с различными кодировками.

В репозитории ИОС «DOMIC» представлены различные форматы документов, среди которых можно выделить следующие:

- веб-страницы (.html, .htm);
- документы MS Office (.doc, .docx, .ppt, .pptx, .xls, .xlsx);
- электронные документы (.pdf);
- файлы исходных кодов (.java, .cpp, .mw, .mws, .sql, .tex);
- текстовые файлы (.txt).

По поводу HTML-документов следует отметить, что они могут быть связаны между собой ссылками. Из базы данных ИОС «DOMIC» можно получить лишь путь к главному файлу. Программный модуль должен извлекать текст из главного файла и всех файлов, на которые ссылается главный, но при условии что эти файлы находятся в том же файловом репозитории (т.к. нет смысла извлекать данные из веб-страниц, которые находятся где-то в Интернете).

2. Необходимо настроить поисковую машину таким образом, чтобы она позволяла индексировать русскоязычные документы и осуществлять по ним поиск; в случае необходимости добавить дополнительные компоненты.
3. Приложение должно предоставлять клиентский интерфейс, посредством которого пользователи могли бы получить доступ к реализованной функциональности.
4. Программа должна предоставлять возможность для автоматической переиндексации коллекции документов по заданному расписанию. При этом документы, которые не были подвергнуты изменениям с момента предыдущей переиндексации, не должны вновь добавляться в индекс в виду того, что в коллекции документов находятся несколько тысяч документов, и операция по повторному добавлению их индекс является дорогостоящей.

## Глава 2. Технологии разработки

Разработка поискового сервиса для ИОС «DOMIC» велась на языке программирования Java 8-й версии. Для реализации клиентской части приложения использовались язык разметки гипертекстовых документов — HTML5, формальный язык описания внешнего вида документа — CSS, а также скриптовый язык сценариев для придания интерактивности веб-страницам — JavaScript.

В качестве среды разработки использовалась IntelliJ IDEA версии 16 и 17, MySQL — в качестве системы управления базами данных.

Для организации и полнотекстового поиска по коллекции документов ИОС «DOMIC» использовался поисковый движок Apache Solr.

Также были использованы следующие библиотеки и фреймворки, написанные на Java. Среди основных из них можно выделить следующие:

- Apache Tika — библиотека для извлечения данных из текстовых документов большинства популярных форматов;
- Spring Framework — фреймворк для построения веб-приложения, основанного на HTTP и сервлетах (используется модуль Spring MVC);
- Hibernate — библиотека, обеспечивающая связь программных объектов с таблицами базы данных.

Далее будут более подробно описаны названные выше программные средства и решения.

### 2.1. Язык программирования Java

Java [6] — это сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (которую в 2009 году приобрела компания Oracle). Приложения на Java обычно транслируются в специальный байт-код, выполняемый виртуальной машиной Java (JVM) — про-



граммой, обрабатывающей байт-код и передающей инструкции оборудованию как интерпретатор. Дата официального выпуска языка — 23 мая 1995 года.

Одно из главных преимуществ языка Java — его независимость от платформы, на которой выполняются программы. Таким образом, один и тот же код можно запускать под управлением различных операционных систем: Windows, Linux, Solaris и др. Это становится очень важным, когда программы всё чаще загружаются посредством глобальной сети Интернет и используются на различных платформах.

Другим не менее важным преимуществом Java является большая схожесть с языком программирования C++. Поэтому тем программистам, которые знакомы с синтаксисом C и C++, будет просто освоить Java.

Кроме того, Java — полностью объектно-ориентированный язык, даже в большей степени, чем C++. Все сущности в языке Java являются объектами, за исключением немногих основных типов (primitive types), например числовых или булевских значений.

Важно и то, что разрабатывать на Java программы, которые не содержат ошибок, значительно легче, чем на C++. Всё дело в том, что разработчиками языка Java из компании Sun был проведён фундаментальный анализ программ на языке C++. Анализировались «узкие места» исходного кода, которые и приводят к появлению трудновывявимых ошибок. Поэтому было принято решение проектировать язык Java с учетом возможности создавать программы, в которых были бы скрыты наиболее распространённые ошибки.

Для этого было сделано следующее:

- Разработчики исключили возможность явного выделения и освобождения памяти. К примеру, память в Java освобождается автоматически с помощью механизма сбора мусора. Получается, что программист застрахован от ошибок, которые возникают от неправильного использования памяти.

- Введение истинных массивов и запрещение указателей. Теперь программисты не могут стереть данные из памяти по причине неправильного использования указателей.
- Была исключена возможность перепутать оператор присваивания с оператором сравнения на равенство. Как правило, проблема со знаком “=” очень часто приводит в С и С++ к логическим ошибкам, которые не так просто обнаружить, особенно в крупных программах.
- Полностью исключено множественное наследование. Оно было заменено новым понятием — интерфейс, идея которого была позаимствована из языка Objective C. Интерфейс даёт программисту практически всё, что тот может получить от множественного наследования, избегая при этом сложностей, которые возникают при управлении иерархиями классов.

Java — один из самых популярных языков программирования, простой в изучении и использовании. За всё время существования языка на нём было написано большое количество различных библиотек и фреймворков, являющиеся решениями почти любой проблемы, которая может встать перед разработчиком. Поэтому Java и был выбран в качестве основного языка для разработки поискового сервиса для ИОС «DOMIC».

## 2.2. Язык программирования Javascript

JavaScript (JS) [7] — скриптовый язык программирования, предназначенный для написания сценариев для придания интерактивности HTML-страницам.

JavaScript разработан фирмой Netscape Communication Corporation. Первоначальное название языка — LiveScript. После завоевания языком Java всемирной известности LiveScript из коммерческих соображений переименовали в JavaScript.

Важная особенность JavaScript — объектная ориентированность: программисту доступны многочисленные объекты, такие как документы, гиперссылки, формы, фреймы и т.д. Эти объекты характеризуются описательной информацией (свойствами), а также возможными действиями (функциями).

Из недостатков языка стоит отметить то, что общепринятые стандарты JavaScript по-разному поддерживаются разными браузерами. Однако данная проблема решается благодаря использованию различных библиотек, в частности, библиотеки jQuery [16].

## 2.3. Среда разработки IntelliJ IDEA

IntelliJ IDEA [4] — это интегрированная среда разработки программного обеспечения на многих языках программирования (в частности Java, JavaScript), разработанная компанией JetBrains. На данный момент является самой популярной средой разработки на Java.

Первая версия программы появилась в январе 2001 года и быстро приобрела популярность, как первая среда для Java с широким набором интегрированных инструментов для рефакторинга, которые позволяли программистам быстро реорганизовывать исходные тексты программ. Дизайн среды ориентирован на продуктивность работы программистов, позволяя сконцентрироваться на функциональных задачах, в то время как IntelliJ IDEA берёт на себя выполнение рутинных операций. Ни одна другая среда разработки не содержит такого многообразия различных функций, её возможности простираются далеко за пределы простой разработки и тестирования программ.

Начиная с шестой версии продукта IntelliJ IDEA предоставляет интегрированный инструментарий для разработки графического пользовательского интерфейса. Среди прочих возможностей, среда хорошо совместима со многими популярными свободными инструментами разработчиков, такими как CVS,

Subversion, Apache Ant, Maven и JUnit.

Начиная с версии 9.0, среда доступна в двух редакциях: Community Edition и Ultimate Edition. Community Edition является полностью свободной версией, доступной под лицензией Apache 2.0, в ней реализована полная поддержка Java SE, Groovy, Scala, а также интеграция с наиболее популярными системами управления версиями (Git, SVN). В редакции Ultimate Edition реализована поддержка Java EE, UML-диаграмм, подсчёт покрытия кода, а также поддержка других систем управления версиями, языков и фреймворков.

Список из наиболее популярных поддерживаемых языков программирования включает в себя: Java, JavaScript, Python, Groovy, Ruby, Scala, SQL, PHP, C, C++. Ряд языков поддерживаются посредством плагинов, разработанных сторонними разработчиками и добавленными в официальный репозиторий плагинов IntelliJ IDEA.

## **2.4. Система управления базами данных MySQL**

MySQL [9] — это свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ, LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в Distribu-

тив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

Из ограничений данной СУБД можно отметить то, что MySQL плохо подходит для использования в высоконагруженных приложениях, а также то, что MySQL не позволяет корректно применять регулярные выражения (операторы REGEXP и RLIKE) на строках в многобайтовых кодировках, например, для Юникода (UTF-8).

## 2.5. Поисковая машина Apache Solr

Apache Solr [2] (<http://lucene.apache.org/solr>) — это высокопроизводительный, потокобезопасный поисковый сервер промышленного качества, построенный на базе Apache Lucene. Сервер Solr был разработан компанией CNET, которая в начале 2006-го года безвозмездно передала его фонду Apache Software Foundation. С тех пор он не переставал развиваться — в него добавлено много новых возможностей, а вокруг сформировалось большое и активное сообщество, члены которого разрабатывают новые функции, исправляют ошибки и повышают производительность. Вот некоторые функции, которые ставят Solr на одно из лидирующих мест среди поисковых серверов:

- простые, основанные на HTTP протоколы индексирования и поиска, а также наличие клиентов, написанных на Java, PHP, Ruby и других языках;

- развитые механизмы кэширования и репликации с целью повышения производительности;
- простота настройки;
- фасетный поиск;
- выделение поисковых термов в найденных результатах;
- средства администрирования, протоколирования и отладки, позволяющие не гадать, что происходит в работающем сервере Solr;
- распределённый поиск;
- проверка орфографии;
- извлечение содержимого с помощью Apache Tika;
- качественная документация.

Одна из лучших черт Solr — использование Apache Lucene [1]. Как и вокруг Solr, вокруг Lucene образовалось активное сообщество, и у неё сложилась репутация надёжной поисковой библиотеки (Solr же, напротив, является поисковым сервером), выдающей высококачественные результаты с великолепной производительностью. Apache Lucene, первоначально написанная Дугом Каттингом (Doug Cutting), превратилась в быструю и мощную библиотеку для полнотекстового поиска [19].

## 2.6. Библиотеки Apache Tika

Apache Tika [3] — это набор библиотек, написанных на языке Java, предназначенный для извлечения, анализа, выделения мета-данных и структурированного текстового контента из файлов различных форматов. Всего поддерживается более 2000 форматов, включая HTML, XML, DOC, OLE2, OOXML, RTF, ePub, OpenDocument, PDF, различные форматы изображений, мультимедиа,

архивов и пакетов программ.

Первоначально Apache Tika была разработана как часть Apache Lucene, но позднее переросла в самостоятельный проект.

Кроме библиотек, в состав Tika входит консольная утилита и GUI-приложение для удобного извлечения данных из различных файлов.

## 2.7. Фреймворк Spring

Spring Framework (или просто Spring) [11] — это универсальный фреймворк с открытым исходным кодом для Java-платформы. Также существует форк для платформы .NET Framework, названный Spring.NET.

Несмотря на то, что Spring не обеспечивает какую-либо конкретную модель программирования, он стал широко распространённым в Java-сообществе главным образом как альтернатива и замена модели Enterprise JavaBeans. Spring предоставляет большую свободу Java-разработчикам в проектировании; кроме того, он предоставляет хорошо документированные и лёгкие в использовании средства решения проблем, возникающих при создании приложений корпоративного масштаба.

Между тем, особенности ядра Spring применимы в любом Java-приложении, и существует множество расширений и усовершенствований для построения веб-приложений на Java Enterprise платформе. По этим причинам Spring приобрёл большую популярность и признаётся разработчиками как стратегически важный фреймворк.

## 2.8. Библиотека Hibernate

Hibernate [5] — это библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения (ORM — Object-Relational Mapping).

Целью Hibernate является освобождение разработчика от значительного объёма сравнительно низкоуровневого программирования при работе с реляционными базами данных в объектно-ориентированных языках программирования. Разработчик может использовать Hibernate как в процессе проектирования системы классов и таблиц «с нуля», так и для работы с уже существующей базой данных.

Библиотека не только решает задачу связи классов Java с таблицами базы данных (и типов данных Java с типами данных SQL), но также предоставляет средства для автоматической генерации и обновления набора таблиц, построения запросов и обработки полученных данных. Hibernate может значительно уменьшить время разработки, которое обычно тратится на ручное написание SQL- и JDBC-кода. Hibernate автоматизирует генерацию SQL-запросов и освобождает разработчика от ручной обработки результирующего набора данных и преобразования объектов, максимально облегчая перенос (портирование) приложения на любые базы данных SQL.

Hibernate обеспечивает прозрачную поддержку сохранности данных (persistence) для «POJO» (то есть для стандартных Java-объектов); единственное строгое требование для сохраняемого класса — наличие конструктора по умолчанию (без параметров).

Mapping (сопоставление, проецирование) Java-классов с таблицами базы данных осуществляется с помощью конфигурационных XML-файлов или Java-аннотаций. При использовании файла XML Hibernate может генерировать скелет исходного кода для классов длительного хранения. В этом нет необходимости, если используются аннотации. Hibernate может использовать файл XML или аннотации для поддержки схемы базы данных.

Обеспечиваются возможности по организации отношения между классами «один-ко-многим» и «многие-ко-многим». В дополнение к управлению связями между объектами Hibernate также может управлять рефлексивными отноше-



ями, где объект имеет связь «один-ко-многим» с другими экземплярами своего собственного типа данных.

Нibernate поддерживает отображение пользовательских типов значений. Это делает возможными такие сценарии:

- переопределение типа по умолчанию SQL, Нibernate выбирает при отображении столбца свойства;
- проецирование перечисляемого типа Java на поле БД, будто они являются обычными свойствами;
- проецирование одного свойства в несколько колонок.

Нibernate может использоваться как в самостоятельных приложениях Java, так и в программах Java ЕЕ, выполняемых на сервере (например, сервлет или компоненты ЕJB). Также он может включаться как дополнительная возможность к другим языкам программирования.

## Глава 3. Описание разработанного программного решения

### 3.1. Архитектура приложения

В основе программного проекта, разработанного в рамках данной выпускной квалификационной работы, лежат база данных MySQL и файловый репозиторий информационно-образовательной среды «DOMIC». Также имеется сконфигурированный поисковый сервер Solr, обеспечивающий функциональность для индексирования документов и поиска по ним. И конечно же, реализованный программный продукт, который состоит из нескольких взаимосвязанных модулей. Вначале кратко опишем, что представляют собой эти модули и как они взаимодействуют с другими компонентами, а затем рассмотрим каждый из них подробно.

1. Модуль DoX (сокращение от «DOmic eXtension» — расширение для ИОС «DOMIC»). В этом модуле содержатся механизмы для извлечения данных из документов, а также для взаимодействия с поисковым сервером Solr.

Для извлечения содержимого документов была использована библиотека Apache Tika, в дополнение к которой были реализованы несколько классов, предоставляющих функциональность для извлечения содержимого из файлов, связанных ссылками.

2. Модуль Jerdox (Java sERver DOmic eXtension). Этот модуль объединяет между собой все реализованные в ходе данной работы компоненты. Он является реализацией веб-сервера на основе Spring MVC, который предоставляет функциональность для взаимодействия с базой данных и файловым репозиторием ИОС «DOMIC», а также веб-интерфейс для взаимодействия пользователей с системой.

Взаимодействие с базой данных осуществляется посредством Java Persistence API (JPA) — интерфейса Java, который реализует концепцию ORM (Object-Relational Mapping, объектно-реляционное отображение), представляющую собой технологию для связи таблиц базы данных с объектами языка программирования.

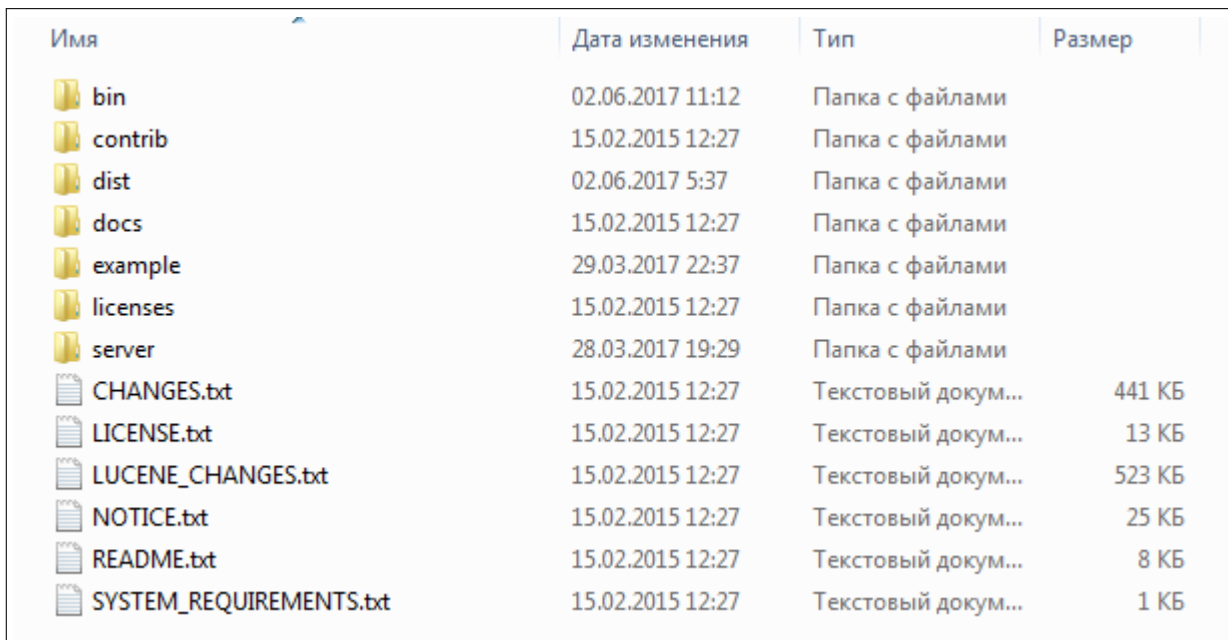
В качестве конкретной реализации этого интерфейса использована библиотека Hibernate. При получении записей из базы данных Hibernate автоматически преобразует их в специальные Entity-объекты, свойства которых как раз и являются атрибутами таблиц БД. С этими объектами уже можно осуществлять различные действия: чтение и изменение их свойств, добавление новых объектов или удаление уже существующих. Причём все изменения могут быть сохранены в базе данных без особых проблем — эти обязанности также берёт на себя Hibernate.

Что касается веб-интерфейса, то при запросе клиента сервер возвращает клиентскую страницу и все необходимые для неё ресурсы (скрипты, таблицы стилей, изображения и т. д.). С этой клиентской страницы можно отправлять запросы как на главный сервер, так и на поисковый сервер Solr.

3. Модуль Jax, представляющий собой набор классов с утилитными методами, основными задачами которых являются инкапсуляция часто повторяющихся фрагментов кода и реализация некоторой полезной функциональности, которая отсутствует в стандартном API Java.
4. Spring-Utils — набор нескольких классов для облегчения конфигурирования программ, написанных с использованием фреймворка Spring.

## 3.2. Настройка Apache Solr

Поисковая машина Solr версии 5.0.0 была скачана с официального сайта Lucene — <http://lucene.apache.org/solr/downloads> в виде zip-архива. Извлечённая из скачаного архива папка Solr представлена на рисунке 1.



Имя	Дата изменения	Тип	Размер
bin	02.06.2017 11:12	Папка с файлами	
contrib	15.02.2015 12:27	Папка с файлами	
dist	02.06.2017 5:37	Папка с файлами	
docs	15.02.2015 12:27	Папка с файлами	
example	29.03.2017 22:37	Папка с файлами	
licenses	15.02.2015 12:27	Папка с файлами	
server	28.03.2017 19:29	Папка с файлами	
CHANGES.txt	15.02.2015 12:27	Текстовый докум...	441 КБ
LICENSE.txt	15.02.2015 12:27	Текстовый докум...	13 КБ
LUCENE_CHANGES.txt	15.02.2015 12:27	Текстовый докум...	523 КБ
NOTICE.txt	15.02.2015 12:27	Текстовый докум...	25 КБ
README.txt	15.02.2015 12:27	Текстовый докум...	8 КБ
SYSTEM_REQUIREMENTS.txt	15.02.2015 12:27	Текстовый докум...	1 КБ

Рисунок 1. Структура корневого каталога Solr

Папка bin содержит скрипты для запуска и остановки сервера. Папка example содержит несколько файлов с примерами документов, настроек Solr, а также необходимые ресурсы. Папка solr внутри папки server содержит все созданные поисковые ядра (core), для каждого из которых определены собственные файлы конфигурации и файлы данных.

Поисковое ядро (core) — это коллекция документов, представленных в Solr в определённом формате, для которой формируется отдельный индекс и в пределах которой можно осуществлять поиск.

Solr представляет собой веб-сервер, развёрнутый внутри контейнера сервлетов Jetty.

Для того, чтобы сконфигурировать поисковую машину, были выполнены следующие действия:

1. Произведён запуск Solr из командной строки с помощью следующей команды `solr start`.

Скрипт `solr.cmd` (для Windows) расположен в папке `bin`, также имеются скрипты для запуска Solr в других операционных системах.

Теперь Solr становится доступным локально по URL-адресу `http://localhost:8983/solr`.

2. Было создано поисковое ядро (core) «domic» с настройками по умолчанию для документов ИОС «DOMIC» при помощи команды `solr create -c domic -d basic_configs`.

Параметры команды `solr create`:

- `-c <name>` — название создаваемого core (обязательный параметр);
- `-d <confdir>` — директория, из которой копируются файлы конфигурации и дополнительные служебные файлы;
- `-n <configName>` — название конфигурации, по умолчанию такое же, как у core;
- `-p <port>` — порт запущенного локального экземпляра Solr, по умолчанию скрипт пытается определить порт путём поиска запущенных экземпляров Solr;
- `-s <shards>` — количество частей, на которые core будет разбит, по умолчанию 1;
- `-rf <replicas>` — количество копий каждого документа в core, по умолчанию 1.

Теперь созданному поисковому ядру можно посылать HTTP-запросы по URL-адресу `http://localhost:8983/solr/domic`. На рисунке 2 представлен интерфейс администратора Solr.

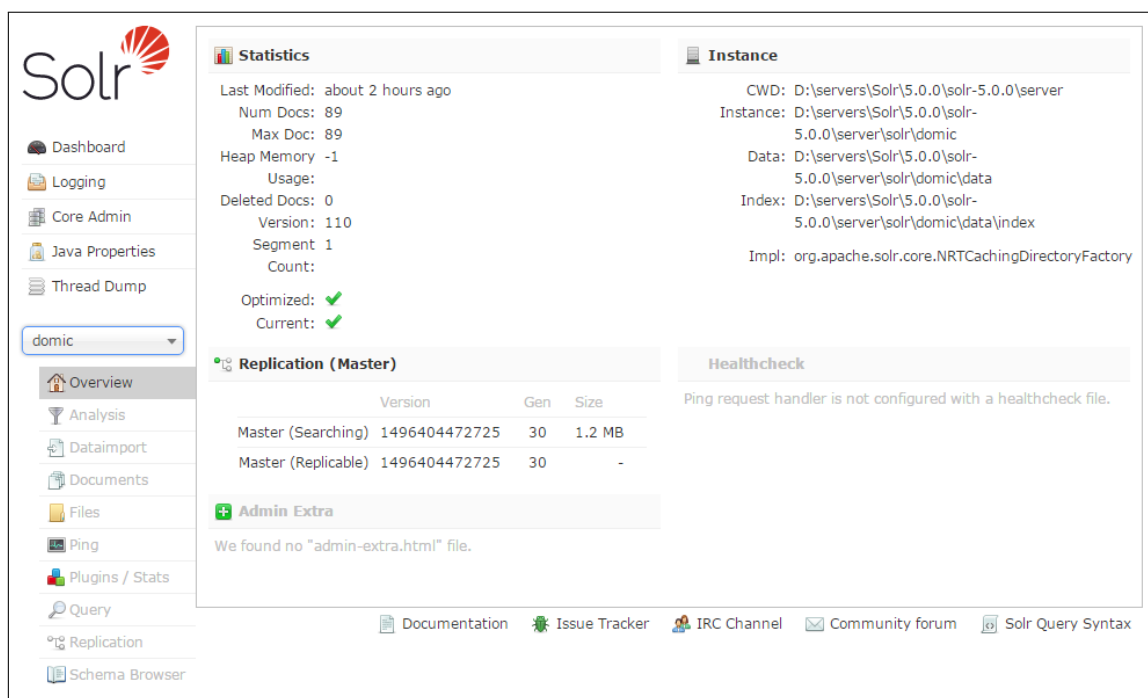


Рисунок 2. Интерфейс администратора Solr

3. Были добавлены новые типы полей документов, поля, а также их обработчики в файл конфигурации `schema.xml` созданного поискового ядра.

В Solr каждый документ представляется в виде набора полей — свойств, которые описывают этот документ. Поля используются Solr'ом для хранения и индексирования некоторого содержимого документов. Эти поля бывают различных типов: текстовые, числовые, булевские, бинарные и другие.

Чтобы Solr мог индексировать тексты на русском языке, в файл `schema.xml` был добавлен тип поля под названием «`text_ru`», который представлен на рисунке 3.

Для этого поля указаны два анализатора: один описывает этапы предобработки содержимого во время его индексирования, другой — во время выполнения запросов.

Анализатор для индексируемого содержимого содержит следующие стан-

```

<fieldType name="text_ru" class="solr.TextField"
  positionIncrementGap="100" omitNorms="true">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true"
      words="lang/stopwords_ru.txt"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.KeywordMarkerFilterFactory"
      protected="lang/protwords_ru.txt"/>
    <filter class="solr.RussianLightStemFilterFactory"/>
    <filter class="solr.PorterStemFilterFactory"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.SynonymFilterFactory"
      synonyms="lang/synonyms_ru.txt"
      ignoreCase="true" expand="true"/>
    <filter class="solr.StopFilterFactory"
      ignoreCase="true" words="lang/stopwords_ru.txt"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.KeywordMarkerFilterFactory"
      protected="lang/protwords_ru.txt"/>
    <filter class="solr.RussianLightStemFilterFactory"/>
    <filter class="solr.PorterStemFilterFactory"/>
  </analyzer>
</fieldType>

```

Рисунок 3. Тип поля text\_ru в файле schema.xml

дартные обработчики, которые есть внутри Solr (следует отметить, что в качестве обработчиков указываются специальные фабричные классы, которые создают необходимые обработчики):

- стандартный токенизатор (StandardTokenizerFactory), разбивающий текст на отдельные токены по специально описанному алгоритму;
- фильтр стоп-слов (StopFilterFactory), удаляющий из списка полученных токенов все слова, указанные в файле «lang/stopwords\_ru.txt», который находится в папке созданного ядра «domic» и который можно дополнить вручную;
- фильтр LowerCaseFilterFactory, преобразующий все токены к нижнему регистру (предыдущая операция — удаление стоп-слов — в Solr является регистронезависимой, поэтому вполне логично было указать

данный фильтр следующим);

- фильтр `KeywordMarkerFilterFactory`, который в отличие от фильтра стоп-слов, наоборот помечает некоторые слова как ключевые, то есть запрещённые для удаления и изменения. Все такие слова указаны в файле «`lang/protwords_ru.txt`», который так же находится в папке поискового ядра «`domic`» и который так же можно редактировать вручную;
- облегчённый стеммер русского языка (`RussianLightStemFilterFactory`), который отбрасывает от слов их окончания;
- стеммер Портера (`PorterStemFilterFactory`) для английского языка, который добавлен для отбрасывания окончаний английских слов.

Анализатор для текста запросов отличается от предыдущего лишь тем, что после токенизатора в нём указан фильтр `SynonymFilterFactory`, который расширяет слова запросов синонимами. Синонимы указываются в текстовом файле «`lang/synonyms_ru.txt`», который тоже можно найти в папке с ядром «`domic`».

В качестве ещё одного типа поля был добавлен тип «`prefix`», который предназначен для выполнения префиксных запросов. Этот тип показан на рисунке 4.

Поле с таким типом используется в клиентском приложении для запросов с автодополнением: пользователь вводит запрос, программа после каждого введённого символа отправляет запрос на сервер Solr с введённым текстом, в ответ на который Solr возвращает возможные варианты продолжения, которые тут же показываются пользователю, из них он может выбрать подходящий.

Из неописанных ранее компонентов анализаторов здесь присутствует лишь



```

<fieldType name="prefix" class="solr.TextField"
  stored="false" indexed="true">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.EdgeNGramFilterFactory"
      minGramSize="2" maxGramSize="10"/>
  </analyzer>
  <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
  </analyzer>
</fieldType>

```

Рисунок 4. Тип поля prefix в файле schema.xml

фильтр `EdgeNGramFilterFactory` для вычисления префиксов индексируемых слов и добавления их в множество генерируемых лексем. В нём используется разбиение лексем на n-граммы. Указаны минимальная и максимальная длины n-граммы: 2 и 10 соответственно. Для анализатора текста запросов этот фильтр указывать нет необходимости, так как запрос и так уже является префиксом.

Этот тип поля позволяет обрабатывать запросы с несколькими словами и допускает совпадения, начинающиеся на границе слова.

Также в файле `schema.xml` описаны все поля, которые будут описывать индексируемые документы. Эти поля представлены на рисунке 5.

- `id` — идентификатор документа в коллекции (обязательное поле с уникальным значением);
- `title` — название документа. В качестве типа для этого поля был указан добавленный ранее «`text_ru`»;
- `resourceName` — название файла (используется для ссылки на файл в клиентском приложении);
- `creationDate` — дата создания документа (если возможно получить);

```

<uniqueKey>id</uniqueKey>
<field name="title" type="text_ru"
      indexed="true" stored="true"/>
<field name="resourceName" type="string"
      indexed="true" stored="true"/>
<field name="creationDate" type="date"
      indexed="true" stored="true"/>
<field name="contentType" type="string"
      indexed="true" stored="true"/>
<field name="author" type="text_ru"
      indexed="true" stored="true"/>
<field name="module" type="string"
      indexed="true" stored="true"/>
<field name="studyEntityType" type="string"
      indexed="true" stored="true"/>
<field name="text" type="text_ru"
      indexed="true" stored="true"
      multiValued="false"/>
<field name="textPrefix" type="prefix"/>
<copyField source="title" dest="textPrefix"/>

```

Рисунок 5. Поля документов в файле schema.xml

- contentType — тип документа (расширение файла);
- author — автор документа (при наличии);
- module — модуль ИОС «DOMIC», к которому относится документ;
- studyEntityType — тип учебной единицы ИОС «DOMIC», к которой относится документ;
- text — текстовое содержимое документа. В качестве типа также указан «text\_ru». Это основное поле, по которому производится индексирование и поиск;
- textPrefix — поле добавленного типа «prefix», содержимое которого берётся из поля title и которое используется в клиентском приложении для префиксных запросов.

После всех этих действий Solr полностью сконфигурирован для решения задачи индексирования коллекции документов ИОС «DOMIC» и поиска по ней.

### 3.3. Модуль извлечения данных

Модуль для извлечения текстовых данных из документов является составной частью проектного модуля DoX. Этот модуль состоит из нескольких Java-классов, его общая структура представлена на рисунке 6.

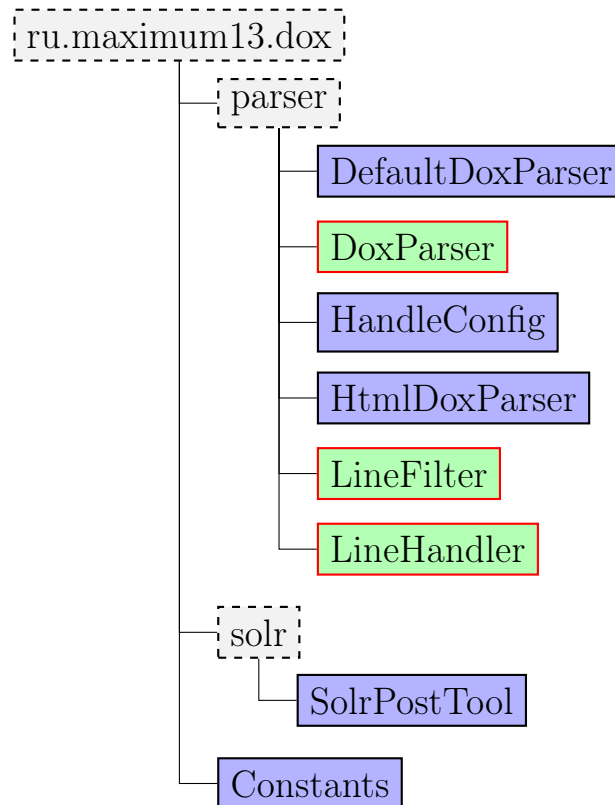


Рисунок 6. Структура классов модуля DoX

Модуль является обёрткой (с дополнительным функционалом) библиотеки Apache Tika, которая содержит механизмы по определению кодировок и извлечению содержимого и метаданных из файлов более чем 2000 форматов.

Основным интерфейсным классом модуля является класс DoxParser из пакета ru.maximum13.dox.parser. В этом интерфейсе есть два абстрактных метода:

- Parser getTikaParser() — возвращает необходимый парсер из библиотеки Tika.
- String parse(...) — возвращает извлечённое текстовое содержимое файла.

Метод принимает три аргумента: файл, из которого нужно извлечь текст;

объект `metadata` класса `Metadata`, в который будут добавлены все полученные из файла метаданные; объект класса `HandleConfig`, содержащий обработчики извлечённого содержимого.

В дополнение к двум абстрактным методам интерфейс содержит метод с реализацией (что стало возможным с появлением Java 8) `default void parse(...)`, который так же принимает три аргумента, за исключением того, что последним параметром является массив переменной длины из объектов классов, наследуемых от интерфейса `org.xml.sax.ContentHandler`, который используется библиотекой `Tika` для определения способа обработки и хранения данных.

Также `DoxParser` содержит несколько статических методов:

- `static String handle(...)`, принимающий два аргумента: объект класса `Reader`, из которого метод принимает данные, и объект класса `HandleConfig`, определяющий способ обработки этих данных, например, удаление пустых строк или лишних пробелов.
- Два перегруженных метода `static DoxParser of(...)`. Один из них принимает в качестве аргументов файл, второй — расширение файла. Методы `of(...)` возвращают соответствующий расширению файла парсер. В частности, для расширений `html`, `htm` возвращает объект класса `HtmlDoxParser`, для всех остальных — объект класса `DefaultDoxParser`.

Для интерфейса `DoxParser` написано две конкретных реализации:

- `DefaultDoxParser` — является по сути простой обёрткой над библиотекой `Tika` с указанным парсером по умолчанию `AutoDetectParser`, который автоматически определяет типы файлов.
- `HtmlDoxParser` — класс с переопределённым методом `public String parse(File file, Metadata metadata, HandleConfig config)`. Этот метод получает на входе HTML-файл, получает из него содержимое, затем извлекает из файла все ссылки на другие документы, и извлекает содержимое из тех,

которые находятся в том же файловом репозитории (той же машине). Объединённое содержимое всех этих файлов возвращается методом `parse()` в виде строки.

Класс `HandleConfig` содержит в себе список фильтров и обработчиков строк текста. Фильтры являются объектами, создаваемыми в качестве реализаций функционального интерфейса `LineFilter`, который является наследником стандартного функционального интерфейса `Predicate`. Определяют логическим условием, какие строки текста необходимо отфильтровать. Обработчики — это объекты функционального интерфейса `LineHandler` (наследника `UnaryOperator`). Определяют способ преобразования строк файлов (например, удаление начальных или конечных пробельных символов).

Помимо вышеперечисленных классов, в проектный модуль `DoX` добавлен класс `SolrPostTool` для отправки запросов серверу `Solr`. Этот класс является обёрткой над классом `HttpSolrClient` из библиотеки `SolrJ`.

В классе `SolrPostTool` задаются настройки по умолчанию для соединения с `Solr` (такие как время ожидания подключения и выполнения запроса, парсер ответов сервера, максимальное количество соединений и др.), а также реализованы несколько методов-обёрток:

- `SolrPingResponse ping()` — позволяет узнать доступность поискового сервера.
- `UpdateResponse post(...)`, который позволяет отправлять `Solr`'у коллекцию документов (объектов класса `SolrInputDocument` из библиотеки `SolrJ`) на индексацию. Также реализован перегруженный метод, который в качестве параметра принимает массив переменной длины, состоящий из объектов типа `SolrInputDocument`. В случае успешной отправки документов метод отправляет запрос на фиксацию изменений в `Solr`, иначе предыдущая операция не будет иметь никакого эффекта.

- `QueryResponse query(String q)` — отправляет поисковые запросы серверу.
- `UpdateResponse deleteByQuery(String q)` — позволяет по заданному запросу удалить документы из индекса Solr.

Все эти методы класса `SolrPostTool` возвращают объекты классов, наследованных от базового класса `SolrResponse`, которые представляют собой ответы на запросы (на каждый тип запроса — соответствующий тип ответа).

В SolrJ эта функциональность реализована за счёт отправки POST- и GET-запросов серверу посредством Apache HTTP-клиента.

### 3.4. Разработка клиентского приложения

Клиентское приложение, как уже было сказано ранее, решено было разрабатывать с применением Spring Framework. На основе модуля MVC этого фреймворка было реализовано веб-приложение, общая структура которого представлена в Приложении 2.

#### Конфигурирование веб-приложения

Обычно для конфигурирования таких веб-приложений в Spring MVC используются файлы `application-context.xml`, `mvc-dispatcher-servlet.xml` и `web.xml`, в которых описываются все необходимые сущности и бины, а также их настройки и параметры.

Однако от такого подхода решено было отказаться в пользу Java-конфигураций [12], то есть конфигураций, описанных в виде Java-классов. Этот подход имеет ряд преимуществ, из которых основными, пожалуй, являются следующие:

- использование синтаксиса Java, что очень удобно;
- многие ошибки конфигураций выявляются уже на стадии компиляции исходных кодов, ошибки же синтаксиса отлавливаются средой разработки.

В качестве основного класса конфигурации веб-приложения был реализован класс `WebConfig`, наследуемый от класса `WebMvcConfigurerAdapter` (стандартного Spring’овского класса). У класса были указаны следующие аннотации:

1. `@Configuration` — помечаем данный класс как класс конфигурации;
2. `@EnableWebMvc` — включаем поддержку Spring MVC (без данной аннотации веб-приложение работать не будет);
3. `@EnableScheduling` — включаем поддержку планировщика задач (выполнения задач по расписанию);
4. `@ComponentScan(«ru.maximum13.jerdox»)` — указываем, что Spring должен сканировать классы в этом пакете и во всех его подпакетах на наличие Spring’овских компонентов.

В классе были переопределены следующие методы:

- `addResourceHandlers(...)` — в методе были добавлены отображения url-адресов файлов ресурсов на пути файлов на сервере;
- `configureViewResolvers(...)` — в этом методе в качестве веб-страниц для отображения по умолчанию были указаны jsp-страницы, путь к которым на сервере был также указан;
- `configureContentNegotiation(...)` — в данном методе были установлены основные типы данных, возвращаемые сервером: JSON и HTML;
- `configureMessageConverters(...)` — в методе были добавлены конвертеры, преобразующие Java-классы в JSON (и обратно) для передачи их по HTTP.

Ещё один класс конфигурации — `AppConfig`, который унаследован от класса `AbstractAnnotationConfigDispatcherServletInitializer`. В этом классе были реализованы абстрактные методы:

- `Class<?>[] getRootConfigClasses()` — метод возвращает класс `WebConfig`, который является основным классом конфигурации приложения;

- `Class<?>[] getServletConfigClasses()` — данный метод также возвращает класс `WebConfig`, который является файлом конфигурации сервлета (веб-приложения);
- `String[] getServletMappings()` — метод возвращает путь по умолчанию для текущего сервлета ("/");

Метод `DispatcherServlet createDispatcherServlet(...)` также переопределён в данном классе. В этом методе было указано, что создаваемый диспетчер сервлетов должен выбрасывать исключение в случае, если для запрошенного с сервера url не будет найден соответствующий обработчик. Это необходимо для того, чтобы в случае, если пользователь обратится к несуществующему url данного сервера, ему вернулась заготовленная страница с заголовком «Данной страницы не существует» вместо стандартной страницы с ошибкой сервлета.

### **Создание объектов сущностей базы данных**

В пакете `ru.maximom13.jerdox.model` было реализовано пять классов, представляющих собой объектное отображение таблиц реляционной базы данных `domic`. Так как для этого была использована спецификация JPA, то все эти классы были помечены аннотацией `@Entity`, указывающей на то, что эти классы являются сущностями БД. Также каждому классу была добавлена аннотация `@Table`, которая указывает, отображением какой конкретной таблицы данный класс является.

У каждого класса сущности объявлены поля, которые являются отображениями столбцов соответствующих таблиц базы данных (для каждого поля класса установлена аннотация `@Column` для указания конкретного столбца таблицы). Также в классы были добавлены конструктор по умолчанию, геттеры и сеттеры, а одно из полей было помечено аннотацией `@Id` (для указания поля-идентификатора). Всё это является требованиями к классам сущностей в JPA.

Перечислим эти Entity-классы:



- Course — класс-отображение таблицы `domic.course`;
- Discipline — класс-отображение таблицы `domic.discipline`;
- Module — класс-отображение таблицы `domic.module`;
- StudyEntity — класс-отображение таблицы `domic.studyentity`;
- StudyEntityType — класс-отображение таблицы `domic.studyentitytype`.

### **Реализация объектов доступа к данным**

Объекты доступа к данным, или по-другому DAO-объекты (Data Access Object) — это объекты языка программирования, позволяющие получить данные из БД (либо в виде объектов-сущностей, либо в виде каких-либо значений). Обычно каждый такой объект отвечает за получение данных из какой-то одной конкретной таблицы. В Spring данные классы помечаются аннотацией `@Repository`.

В пакете `ru.maximum13.jerdex.dao` реализованы три класса: `ModuleDao`, `StudyEntityDao`, `StudyEntityTypeDao`.

Рассмотрим детальнее реализацию этих DAO-классов.

1. `ModuleDao`. В данном классе реализован один метод — `List<Module> getAll()`, который получает из базы данных все сущности `Module` и возвращает их в виде списка.
2. `StudyEntityDao`. Этот класс содержит реализацию нескольких методов, среди которых:
  - `List<StudyEntity> getAll()` — метод получает из базы данных и возвращает список всех сущностей `studyentity`;
  - `long getCount()` — метод возвращает количество записей в таблице `studyentity`;
  - `Set<String> getFirstPageURLs()` — данный метод получает из таблицы `studyentity` множество всех значений столбца `urlFirstPage`.

3. `StudyEntityTypeDao`. Этот класс содержит реализацию двух методов: `List<StudyEntity> getAll()` и `long getCount()`, которые являются аналогами методов с такими же названиями из класса `StudyEntityDao`.

### **Реализация сервисных классов**

В пакете `ru.maximum13.jerdox.service` модуля `Jerdox` реализован один сервисный класс — `IndexService`. В этом классе используются объекты DAO из пакета

`ru.maximum13.jerdox.dao`, а также объект класса `SolrPostTool`. Все эти объекты внедряются в данный сервис при помощи стандартных механизмов внедрения зависимостей `Spring`.

В классе реализованы следующие методы:

- `long getStudyEntitiesCount()` — метод возвращает количество учебных единиц из таблицы `StudyEntity` посредством вызова метода `getCount()` класса `StudyEntityDao`;
- `Set<String> getFileExtensions()` — метод возвращает неповторяющееся множество всех расширений файлов, на которые ссылаются значения столбца `urlFirstPage` таблицы `StudyEntity`;
- `void reindexDatabase()` — метод производит переиндексацию документов `Solr`: при вызове этого метода все документы из коллекции ИОС «DOMIC» отправляются поисковой машине.

### **Реализация классов-контроллеров**

В модуле `Jerdox` было реализовано четыре класса-контроллера, которые расположены в пакете `ru.maximum13.jerdox.controller`:

1. `SearchController` — базовый контроллер приложения. В момент создания объекта данного класса он инициализирует все ресурсы, необходимые для работы приложения: вызывает метод `init()` класса `Resources` из пакета

ru.maximum13.jerdox, который загружает все ресурсы из файла config.xml, расположенного в каталоге resources данного модуля.

В этом классе реализован метод `String loadSearchPage(...)` — обработчик запросов url по умолчанию, обрабатываемый данным контроллером. Возвращает название клиентской страницы поискового сервиса («search.jsp»), которую должен вернуть сервер в ответ на запрос, а также добавляет в модель этой страницы параметр с названием «solr\_domic\_url», в котором представлен url-адрес поисковой машины Solr.

2. `RepositoryController` — контроллер, у которого реализован метода `void loadFile(...)`, который в ответ запроса записывает запрошенный файл из файлового репозитория ИОС «DOMIC».
3. `IndexController` — данный контроллер зависит от `IndexService` (последний внедряется в данный класс как зависимость). В этом контроллере реализован метод `void reindexDatabase()`, который запускается раз в сутки в одно и то же время (благодаря указанной аннотации `@Scheduled`) и делегирует `IndexService`’у задачу по переиндексации коллекции документов ИОС «DOMIC».
4. `ExceptionHandlerController` — данный контроллер обрабатывает ошибки запросов. В частности, в этом классе реализован метод `handleException()`, который в случае, если не будет найден обработчик для запрошенного url, будет возвращать в ответ на запрос страницу с ошибкой.

## **Реализация пользовательского интерфейса**

В качестве пользовательского интерфейса поискового сервиса была реализована веб-страница `search.jsp`, которую возвращает сервер. Данная страница представлена на рисунке 7.

При реализации этого компонента были использованы язык программиро-

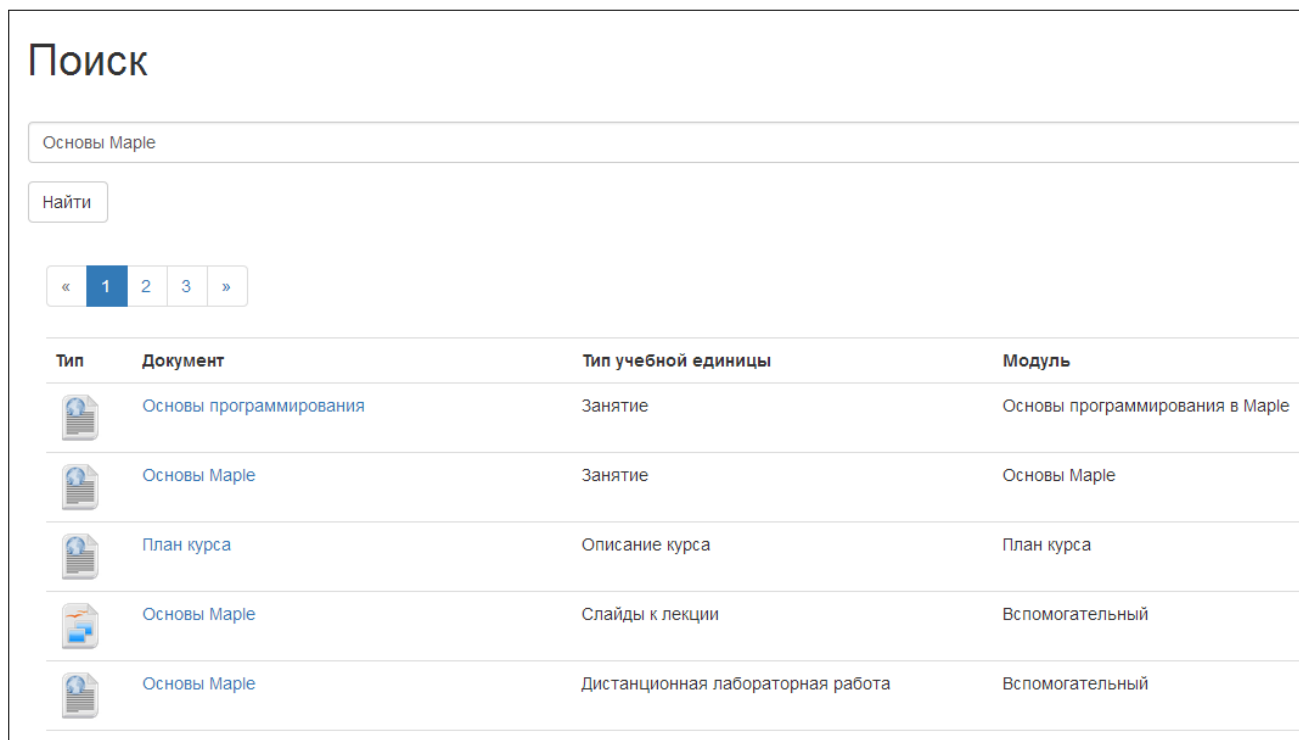


Рисунок 7. Веб-интерфейс поискового сервиса

вания JavaScript [7], а также его библиотеки: jQuery [16], jQuery-ui [17], jQuery-bootpag [15], Bootstrap [8] и Bootstrap-select [13].

На веб-странице расположены следующие компоненты:

- Поисковая строка, в которую необходимо ввести поисковый запрос. При вводе символов в данное поле серверу Solr с помощью AJAX-запроса (который выполняется без необходимости перезагрузки страницы) посылается поисковый запрос по полю «textPrefix». В ответ на этот запрос Solr возвращает несколько возможных вариантов продолжения запроса, которые отображаются пользователю в виде выпадающего списка.

Таким образом реализована функциональность запросов по префиксу, компоненты для которой были настроены в Solr и описаны ранее.

Пример такого запроса и предложенные пользователю варианты продемонстрированы на рисунке 8. Список вариантов упорядочен по их релевантности запросу.

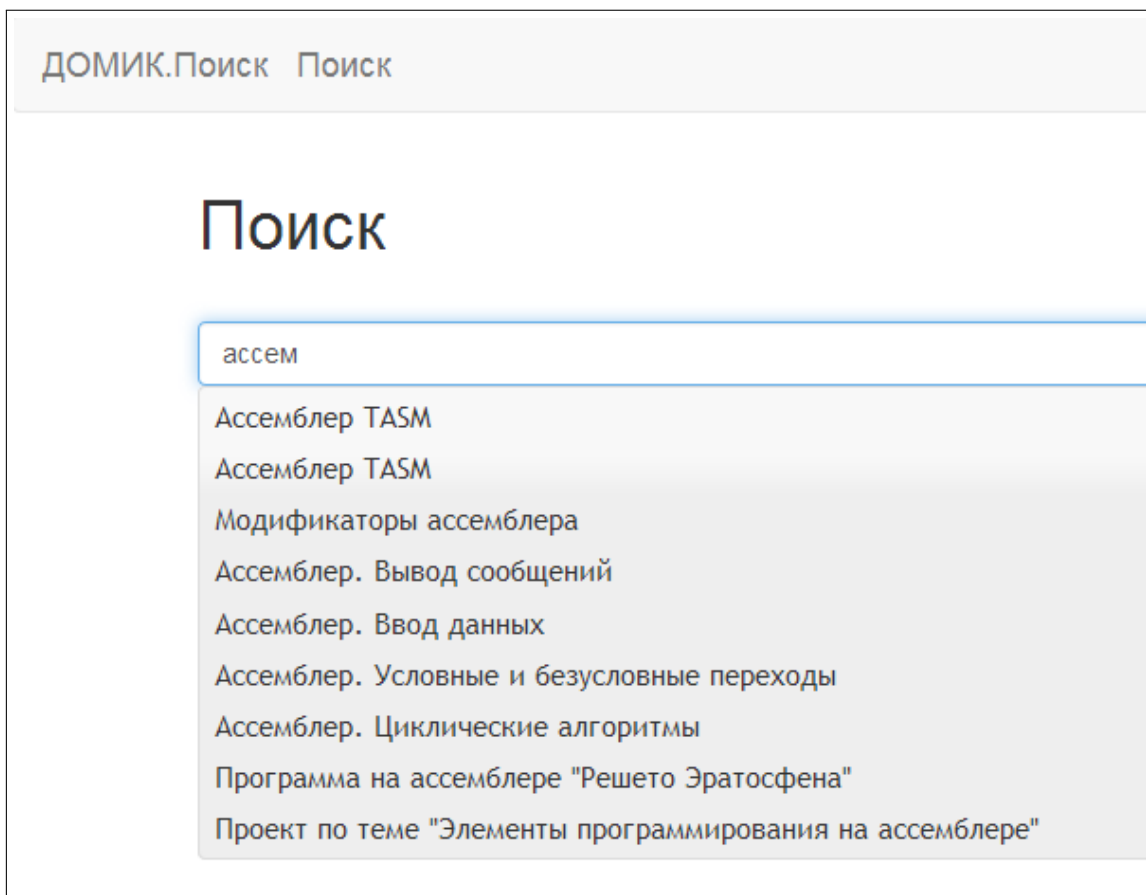


Рисунок 8. Пример префиксного запроса

Как видно из рисунка, пользователю предложены не только варианты запроса, которые завершают введённую строку, но также несколько вариантов, в которых введённые символы встречаются не в начале.

- Кнопка «Найти», при нажатии на которую на сервер Solr отправляется AJAX-запрос с введённым в поисковую строку текстом. В ответ на этот запрос Solr возвращает список всех найденных документов, которые будут отображены в списке результатов.
- Список результатов. В данном списке отображаются сведения о документах, которые вернул Solr в ответ на запрос. Отображаются следующие сведения: название документа и ссылка, по которой этот документ можно открыть для просмотра (либо скачать на свой компьютер, если в браузере документ открыть невозможно); тип учебной единицы и модуль, в

которых находится каждый конкретный документ.

- Кнопки для отображения разных страниц поисковой выдачи. Так как список документов, которые возвращает Solr, может быть достаточно большим, то в таблице отображается максимум 10 документов из данного списка. С помощью же кнопок можно отображать другие 10 элементов из списка. Для удобства пользователей данные кнопки расположены перед и после списка результатов.

### 3.5. Инструкция по развёртыванию сервиса

В Приложении, которое представлено на диске, находится три основных каталога:

- «JDoX» — проект с исходными кодами и собранным приложением разработанного поискового сервиса, а также скриптами для его сборки (для ОС Windows);
- «solr-5.0.0» — папка с поисковой машиной Solr;
- «apache-tomcat-8.5.11\_jdoh» — папка Apache Tomcat с собранным внутри него и готовым к запуску приложением.

Также в корневом каталоге диска расположен bat-файл «start\_server.bat» для запуска поискового сервиса на компьютере с операционной системой Windows.

Что касаето требований для запуска компонентов приложения, на компьютере должна быть установлена Java версии не ниже 8 (как минимум JRE, JDK не обязателен).

Последовательность действий для запуска сервиса:

1. запустить поисковую машину Solr. Для этого нужно перейти в каталог «solr-5.0.0\bin\» и запустить файл «solr\_start.bat»;

2. развернуть поисковый сервис. Для данной операции следует запустить файл «start\_server.bat», который находится в корневом каталоге Приложения, представленного на диске.

## ЗАКЛЮЧЕНИЕ

В результате выпускной квалификационной работы были решены следующие задачи:

- разработан модуль для извлечения содержимого из документов коллекции ИОС «DOMIC», который корректно обрабатывает различные текстовые документы, в том числе связанные между собой ссылками;
- настроена поисковая машина Apache Solr, способная быстро и качественно обрабатывать несколько десятков тысяч (и более) документов;
- реализовано клиентское приложение, с помощью которого пользователи могут получить доступ к коллекции документов ИОС «DOMIC», осуществлять поиск по ней и получать результаты в удобном виде.

Таким образом, поставленная цель была достигнута — реализован поисковый сервис для информационно-образовательной среды «DOMIC».



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

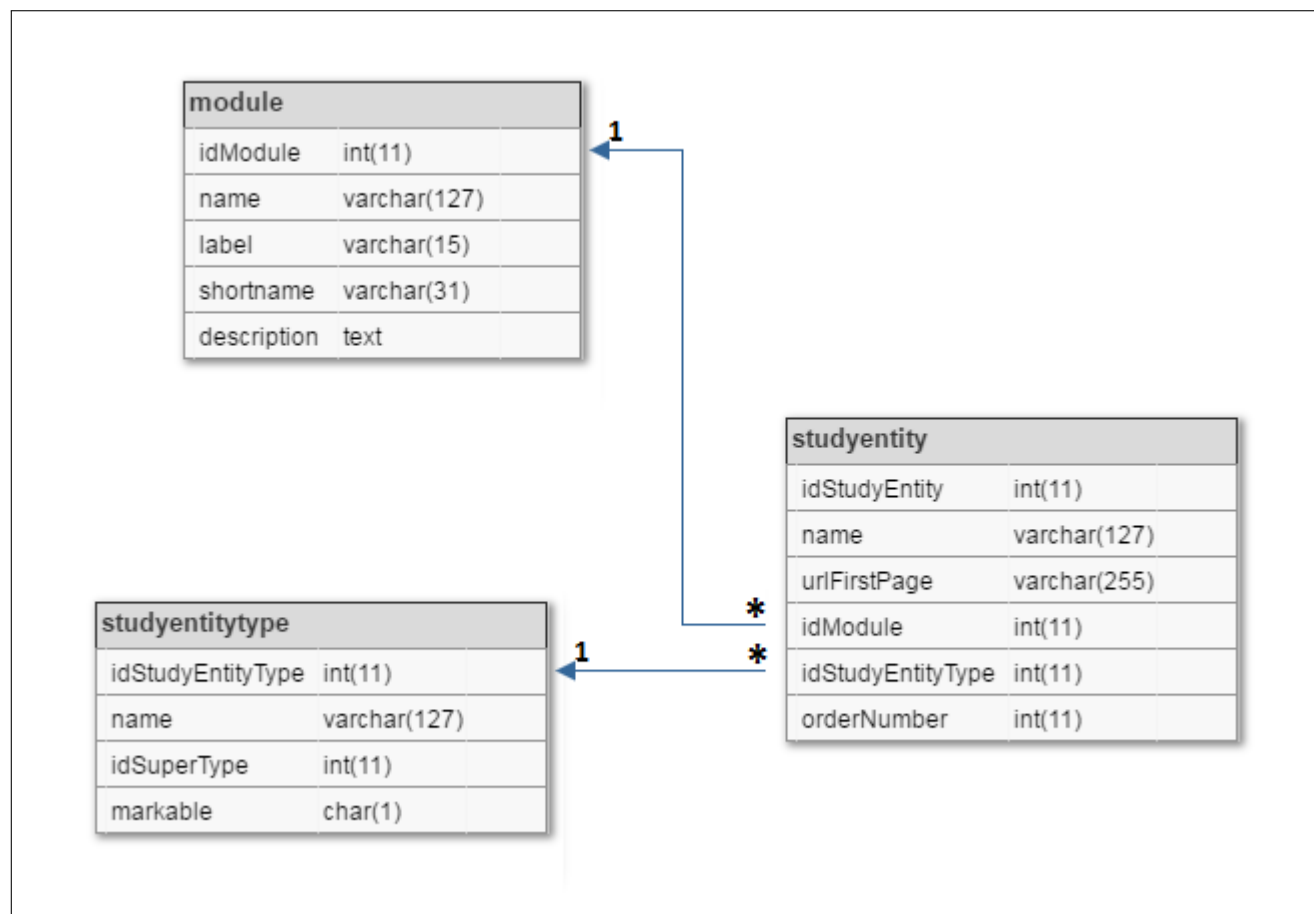
1. Apache Lucene 5.0.0 Documentation [Электронный ресурс]. — [Б. м. : б. и.]. — URL: [http://lucene.apache.org/core/5\\_0\\_0/](http://lucene.apache.org/core/5_0_0/) (дата обращения: 20.04.2017).
2. Apache Solr 5.0.0 Documentation [Электронный ресурс]. — [Б. м. : б. и.]. — URL: [http://lucene.apache.org/solr/5\\_0\\_0/](http://lucene.apache.org/solr/5_0_0/) (дата обращения: 24.04.2017).
3. Apache Tika 1.14 [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <http://tika.apache.org/1.14/> (дата обращения: 25.04.2017).
4. Discover IntelliJ IDEA [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <https://www.jetbrains.com/help/idea/discover-intellij-idea.html> (дата обращения: 02.06.2017).
5. Hibernate ORM documentation [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <http://hibernate.org/orm/documentation/5.1/> (дата обращения: 30.05.2017).
6. Java Platform, Standard Edition (Java SE) 8 [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <http://docs.oracle.com/javase/8/> (дата обращения: 01.04.2017).
7. JavaScript | MDN [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <https://developer.mozilla.org/en/docs/Web/JavaScript> (дата обращения: 31.05.2017).
8. JavaScript. Bootstrap [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <http://getbootstrap.com/javascript/> (дата обращения: 04.06.2017).

9. MySQL Documentation [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <https://dev.mysql.com/doc/> (дата обращения: 31.05.2017).
10. Sphinx. Documentation [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <http://sphinxsearch.com/docs/latest/> (дата обращения: 25.05.2017).
11. Spring Documentation [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <https://spring.io/docs> (дата обращения: 01.05.2017).
12. Spring MVC — JavaConfig либо конфигурация проекта без XML файлов [Электронный ресурс]. — [Б. м. : б. и.], 2014. — URL: <https://habrahabr.ru/post/226663/> (дата обращения: 01.02.2017).
13. Twitter Bootstrap Select Plugin [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <http://plugins.upbootstrap.com/bootstrap-select/> (дата обращения: 05.06.2017).
14. Xapian — the open source search engine [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <https://xapian.org/docs/> (дата обращения: 01.06.2017).
15. bootpag — dynamic pagination jQuery plugin [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <http://botmonster.com/jquery-bootpag/> (дата обращения: 04.06.2017).
16. jQuery API Documentation [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <https://api.jquery.com> (дата обращения: 02.06.2017).
17. jQuery UI API Documentation [Электронный ресурс]. — [Б. м. : б. и.]. — URL: <http://api.jqueryui.com> (дата обращения: 03.06.2017).
18. Астрахов, А. В. Научно-образовательный материал «Технология информационного поиска» [Электронный ресурс]. — [Б. м. : б. и.], 2011. — URL: <http://geum.ru/next/art-212707.php>.

19. Ингерсолл, Грант С. Обработка неструктурированных текстов. Поиск, организация и манипулирование. / Пер. с англ. Слинкин А. А. [Текст] / Грант С. Ингерсолл, Томас С. Мортон, Эндрю Л. Фэррис. — М. : ДМК Пресс, 2015. — 414 с.
20. Лифшиц, Ю. Модель информационного поиска. PageRank. Лекция № 3 курса «Алгоритмы для Интернета» [Электронный ресурс]. — [Б. м. : б. и.], 2006. — URL: <http://yury.name/internet/03ianote.pdf> (дата обращения: 04.06.2017).
21. Маннинг, Кристофер Д. Введение в информационный поиск [Текст] / Кристофер Д. Маннинг, Прабхакар Рагхаван, Хаинрих Шютце. — М. : ООО "И.Д. Вильямс", 2011. — 528 с.

## ПРИЛОЖЕНИЕ 1.

### Фрагмент ER-диаграммы базы данных ИОС «DOMIC»



## ПРИЛОЖЕНИЕ 2.

### Структура классов модуля JDoX

