СОДЕРЖАНИЕ

[Введение 7](#_Toc481589348)

[1 Исследование устройства и работы интернет поисковиков, особенности корпоративных поисковых систем 9](#_Toc481589349)

[1.1 Понятие интернет-поисковика, история появления и разновидности 9](#_Toc481589350)

[1.2 Основные компоненты информационно-поисковой системы и принцип ее работы 11](#_Toc481589351)

[1.3 Характеристика поискового робота 12](#_Toc481589352)

[1.4 Индексирование в поисковых системах 13](#_Toc481589353)

[1.5 Программное обеспечение поисковой системы и графический интерфейс пользователя 14](#_Toc481589354)

[1.6 Обзор современных поисковых систем 14](#_Toc481589355)

[1.7 Корпоративные поисковые системы и их особенности 17](#_Toc481589356)

[1.8 Анализ мирового рынка корпоративных поисковых систем 18](#_Toc481589357)

[2 Анализ процесса поиска информации и систем корпоративного поиска 23](#_Toc481589358)

[2.1 Общая характеристика компании EPAM Systems, сотрудничество с компанией SAP 23](#_Toc481589359)

[2.2 Общий анализ и характеристика компании SAP 25](#_Toc481589360)

[2.3 Анализ процесса поиска информации в компании SAP 27](#_Toc481589361)

[3 Разработка корпоративной информационно-поисковой системы 43](#_Toc481589362)

[3.1 Постановка задачи 43](#_Toc481589363)

[3.2 Спецификация вариантов использования информационно-поисковой системы 45](#_Toc481589364)

[3.3 Информационная модель поисковой системы IT-компании 48](#_Toc481589365)

[3.4 Модели представления информационно-поисковой системы 53](#_Toc481589366)

[3.5 Описание алгоритма работы поисковой системы при сохранении веб-страниц 58](#_Toc481589367)

[3.6 Руководство по развертыванию 59](#_Toc481589368)

[3.7 Результаты тестирования поисковой системы IT-компании 60](#_Toc481589369)

[4 Экономическое обоснование проекта по разработке корпоративной информационно-поисковой системы 84](#_Toc481589370)

[4.1 Описание функций, назначения и потенциальных пользователей корпоративной информационно-поисковой системы 84](#_Toc481589371)

[4.2 Расчет затрат на разработку корпоративной информационно-поисковой системы 85](#_Toc481589372)

[4.3 Определение экономического эффекта от разработки программного продукта 90](#_Toc481589373)

[4.4 Эффективность использования программного продукта 92](#_Toc481589374)

[Выводы и заключения 94](#_Toc481589375)

[Список использованных источников 95](#_Toc481589376)

[Приложение А (обязательное) Диаграмма вариантов использования информационно-поисковой системы 97](#_Toc481589377)

[Приложение Б (обязательное) Схема алгоритма работы поискового робота 98](#_Toc481589378)

[Приложение В (обязательное) Листинг основных элементов программы 99](#_Toc481589379)

# ВВЕДЕНИЕ

Информационные технологии в современном мире используются во всех сферах жизнедеятельности человека. Общество развивается с помощью изучения и запоминания информации. Все больше и больше с каждым днем люди пользуются интернетом. Глобальная информационная компьютерная сеть дала возможность людям быстро получать информацию из любой точки земного шара.

Сложно представить сегодня работу в сети без использования поисковых систем. Нет ничего проще, чем написать в строке поисковой запрос и получить большое количество ответов, подобранных индивидуально для вас, с учетом вашего местоположения, предпочтений, предыдущих запросов.

Поисковые системы являются одними из самых распространенных средств информационных технологий. Более двадцати лет назад они впервые появились в сети интернет. Но в то время число пользователей было ограниченным, а количество информации относительно небольшим. В настоящее время поисковики представляют собой многофункциональные системы, сочетающие в себе множество сервисов. С помощью таких систем пользователи могут искать самую разнообразную информацию, поэтому поисковые системы пользуются колоссальным спросом.

В конце двадцатого столетия с развитием информационных технологий проблема поиска и сбора сведений приобрела новый облик. Если ранее было недостаточно информации, ее сложно было отыскать, то теперь наблюдается ее переизбыток, объем данных постоянно растет. Виртуальные источники дублируют друг друга. С одной стороны, такое многообразие веб-ресурсов позволяет найти ответ на любой вопрос. С другой стороны, чтобы получить искомую информацию, иногда необходимо изучить несколько десятков ссылок.

С ростом объемов интернет ресурсов растут и требования к поисковым системам. Контент и уникальность веб-ресурса становятся наиболее значимыми. От поисковика теперь ожидают не просто результаты по запросу, а отсортированные релевантные страницы с правильными точными данными.

Таким образом, актуальность проблемы обусловлена наличием больших объемов информации, циркулирующих в современном мире, и необходимостью постоянно совершенствовать алгоритмы поиска. Актуальность определила тему дипломного проекта – «Корпоративный информационный поиск и его программная поддержка».

Объектом исследования является процесс поиска информации среди многочисленных веб-ресурсов крупных компаний. Предметом исследования являются современные корпоративные информационно-поисковые системы.

Целью данной работы является сокращение временных затрат при поиске информации сотрудниками компании, предоставление сотрудникам удобного способа поиска веб-ресурсов внутри компании.

В соответствии с поставленной целью определены следующие задачи:

1. Исследовать предметную область. Рассмотреть существующие информационно-поисковые системы, выявить их недостатки и предложить способы их устранения.
2. Рассмотреть и проанализировать процесс поиска информации с точки зрения сотрудников компании. Составить модели «как-есть» и «как-должно-быть».
3. Составить спецификацию вариантов использования поисковой системы.
4. Описать информационную модель.
5. Выбрать компоненты и технологии для реализации информационно-поисковой системы. Учесть необходимость ограничивать видимость ресурсов в зависимости от прав доступа.
6. Для визуализации разрабатываемой системы необходимо описать модели представления системы.
7. Составить руководство по развертыванию системы и предоставить руководство для различных пользователей.
8. Протестировать разработанную систему и оценить, выполнены ли поставленные задачи.
9. Оценить экономические затраты на разработку системы корпоративного поиска, рассчитать экономический эффект от разработки системы.
10. Сделать выводы о целесообразности создания информационно-поисковой системы IT-компании и о преимуществах данной системы.

После того, как определена цель и основные задачи, можно приступить к поэтапной разработке корпоративной информационно-поисковой системы.

# ИССЛЕДОВАНИЕ УСТРОЙСТВА И РАБОТЫ ИНТЕРНЕТ ПОИСКОВИКОВ, ОСОБЕННОСТИ КОРПОРАТИВНЫХ ПОИСКОВЫХ СИСТЕМ

* 1. **Понятие интернет-поисковика, история появления и разновидности**

В современном мире происходит постоянное увеличение объемов информации. Задача поиска информации становится все более актуальной. Создаются и совершенствуются поисковые средства, позволяющие найти необходимые документы. Одним из таких средств является поисковая система.

Что такое интернет-поисковик, или информационно-поисковая система? Это система, предназначенная для поиска информации [1]. Однако это определение является упрощенным. На самом деле информационно-поисковая система позволяет не только искать информацию, но в первую очередь обрабатывать, хранить, сортировать, фильтровать ее. Поисковая система обеспечивает поиск и отбор необходимых данных в специальной базе с описаниями источников информации (индексе) на основе правил поиска. Чтобы найти информацию с помощью этой системы, пользователю необходимо сформулировать поисковый запрос. Поиск может осуществляться по ключевым словам и словам, как-либо связанным с ключевыми словами. В ответ на поисковый запрос поисковая система генерирует результаты поиска.

Первой компьютерной программой для поиска в Интернете, была программа Арчи. Она была создана в 1990 году студентами, изучающими информатику в Монреале. Программа скачивала списки всех файлов со всех доступных анонимных FTP-серверов и строила базу данных, в которой можно было выполнять поиск по именам файлов. Однако программа Арчи не индексировала содержание этих файлов, так как объём данных был настолько мал, что всё можно было легко найти вручную.

Развитие и распространение сетевого протокола Gopher в 1991 году привело к созданию двух новых поисковых программ, Veronica и Jughead. Как и Арчи, они искали имена файлов и заголовки, сохранённые в индексных системах Gopher. Veronica позволяла выполнять поиск по ключевым словам большинства заголовков меню Gopher во всех списках Gopher. Программа Jughead извлекала информацию о меню от определённых Gopher-серверов.

К лету 1993 года ещё не было ни одной системы для поиска в вебе, хотя вручную поддерживались многочисленные специализированные каталоги. Вероятно, первым поисковым роботом, написанным на языке Perl, был «World Wide Web Wanderer» — бот Мэтью Грэя из Массачусетского технологического института в июне 1993 года. Этот робот создавал поисковый индекс. Цель робота Wanderer состояла в том, чтобы измерить размер всемирной паутины и найти все веб-страницы, содержащие слова из запроса. В 1993 году появилась и вторая поисковая система «Aliweb». Aliweb не использовала поискового робота, но вместо этого ожидала уведомлений от администраторов веб-сайтов о наличии на их сайтах индексного файла в определённом формате.

JumpStation, созданный в декабре 1993 год, искал веб-страницы и строил их индексы с помощью поискового робота, и использовал веб-форму в качестве интерфейса для формулирования поисковых запросов. Это был первый инструмент поиска в Интернете, который сочетал три важнейших функции поисковой системы (проверка, индексация и собственно поиск). Из-за ограниченности ресурсов компьютеров того времени индексация и, следовательно, поиск были ограничены только названиями и заголовками веб-страниц, найденных поисковым роботом.

Одной из первых попыток организации доступа к информационным ресурсам сети стало создание тематических каталогов сайтов. Первым, открывшимся в апреле 1994 г, стал Yahoo. Это еще не было поисковой системой, в современном понимании, так как возможность поиска информации ограничивалась ресурсами, зарегистрированными в каталоге Yahoo. Каталоги ссылок ранее использовались довольно широко, но в настоящее время практически утратили свою популярность. Объяснение этому очень простое – даже современные, содержащие огромное количество ресурсов каталоги, представляют информацию лишь о довольно незначительной части сети.

Первой полноценной поисковой системой в 1994 году стал проект WebCrawler. Далее в 1995 году появились поисковые системы AltaVista и Lycos. В 1997 году в Стэндфордском университете, в рамках исследовательского проекта, была создана Google - самая популярная поисковая система на данный момент в мире. В 1997 году появилась поисковая система - Yandex, лидер в русскоязычной части Интернета.

На данный момент существует четыре типа поисковых систем:

1. Системы, использующие поисковых роботов. Состоят из робота, индекса и программного обеспечения поисковой системы. Робот, или краулер, в таких системах постоянно обходит сеть и создает список веб-страниц. Благодаря этому информация всегда актуальная.
2. Системы, управляемые человеком. Система работает с каталогом веб-страниц, который наполняется и обновляется вручную человеком. Поэтому качество контента будет лучше по сравнению с системами на основе поисковых роботов, однако данные в каталоге обновляются медленно и не соответствуют реальным данным.
3. Гибридные системы одновременно имеют функции систем, использующих поисковых роботов, и систем, управляемых человеком. Они часто содержат не только индексную базу, но и структурированный каталог.
4. Мета системы. Они используют результаты сразу нескольких поисковых систем. Они пересылают поисковый запрос сразу нескольким поисковикам, собирают и ранжируют полученные результаты, удаляя дубликаты.

В зависимости от масштаба поисковые системы можно также разделить на виды: локальные, региональные и глобальные. Локальные поисковые системы разрабатываются для поиска информации на отдельном сервере или группе серверов. Региональные поисковые системы осуществляют поиск по информационным ресурсам определенного региона, например, только англоязычные страницы в интернете. Глобальные системы предназначены для описывания ресурсов всего информационного пространства интернета.

* 1. **Основные компоненты информационно-поисковой системы и принцип ее работы**

Информационно-поисковая система состоит из поискового робота, индексатора и поисковика. Каждый компонент поисковой системы представляет собой сложную подсистему с большим набором функций. Схема поисковой системы представлена на рисунке 1.1.

Поисковой робот (веб-паук, краулер) – программа, предназначенная для перебора страниц с целью занесения информации в базу поисковика.

Индексатор – это программа-анализатор веб-страниц. Она "разбирает" на части скачанную страницу и анализирует ее элементы, такие как текст, служебные html-теги, заголовки, особенности стилистики и структурные формы.

Поисковик – графический интерфейс для работы пользователя. Поисковик принимает пользовательские запросы, обрабатывает их при помощи индекса и возвращает результаты поиска.

Все вышеперечисленные компоненты представляют информационно-поисковую систему на основе поисковых роботов. Большинство систем в современном типе являются системами данного типа.

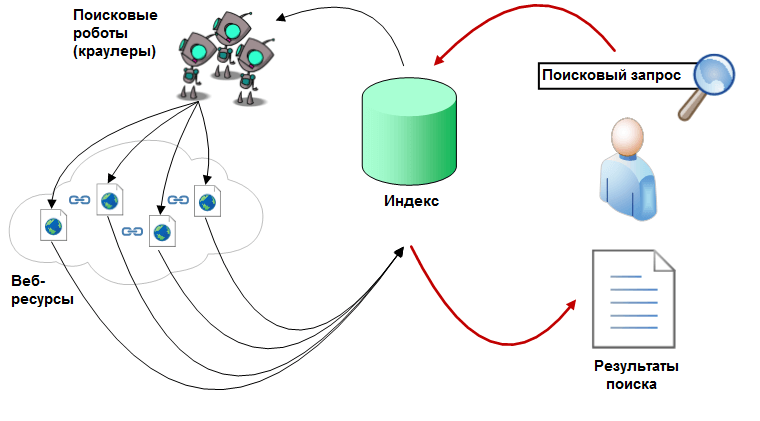


Рисунок 1.1 – Схема работы поисковой системы

Организация работы поисковой системы имеет несколько этапов [2]. Для начала необходимо определить данные, по которым будет осуществляться поиск. После определения этих данных необходимо определить набор правил, по которым поисковой робот найдет все эти данные для дальнейшей обработки. Поисковой робот осуществляет поиск всех страниц из заданного списка адресов и на каждой странице проходит по всем ссылкам, выделяя их. Как правило, системы работают поэтапно. Сначала поисковый робот получает контент, затем индексатор генерирует доступный для поиска индекс, и наконец, поисковик обеспечивает функциональность для поиска индексируемых данных. Чтобы обновить индекс, цикл выполняется повторно.

* 1. **Характеристика поискового робота**

Поисковой робот осуществляет сканирование, или автоматическое изучение, заданного списка страниц. Во время сканирования происходит либо добавление новых страниц в базу или обновление существующих. Процесс сканирования происходит регулярно в целях выявления изменений, удаления несуществующих ссылок и страниц и добавления новых.

Поисковой робот анализирует содержимое страницы, сохраняет его в некотором специальном виде на сервере поисковой машины, которой принадлежит, и отправляется по ссылкам на следующие страницы для анализа. Владельцы поисковых машин нередко ограничивают глубину проникновения паука внутрь сайта и максимальный размер сканируемого текста, поэтому чересчур большие сайты могут оказаться не полностью проиндексированными поисковой машиной.

Функционирование поискового робота строится по тем же принципам, по которым работает браузер. Паук заходит на сайты, оценивает содержимое страниц, переносит их в базу поисковой системы, затем по ссылкам переходит на другой ресурс, повторяя заданный алгоритм действий. Результат этих путешествий — перебор веб-ресурсов в строгой последовательности, индексация новых страниц, включение неизвестных сайтов в базу.

Однако веб-паук не делает анализа контента, он лишь передает его на серверы поисковых систем, где происходит дальнейшая оценка и обработка. Поисковые работы необходимо настроить так, чтобы они регулярно посещали сайты, оценивая их на предмет обновлений.

* 1. **Индексирование в поисковых системах**

Индексация – процесс сохранения полученной информации в базе данных для последующего извлечения информации. Индексатор — это модуль, который анализирует страницу, предварительно разбив её на части, применяя собственные лексические и морфологические алгоритмы. Все элементы веб-страницы вычленяются и анализируются отдельно. Данные о веб-страницах хранятся в индексной базе данных для использования в последующих запросах [3].

Индекс позволяет быстро находить информацию по запросу пользователя. Цель использования индекса — повышение скорости поиска релевантных документов по поисковому запросу. Без индекса поисковая машина должна была бы сканировать каждый документ, что потребовало бы большого количества времени и вычислительной мощности. Например, в то время как индекс 10 000 документов может быть опрошен в пределах миллисекунд, последовательный просмотр каждого слова в 10 000 больших документов мог бы занять часы. Дополнительная память, выделяемая для хранения индекса, и увеличение времени, требуемое для обновления индекса, компенсируется уменьшением времени на поиск информации.

Одним из наиболее распространённых типов поисковых индексов в наши дни является полнотекстовый индекс, используемый, как несложно догадаться, при полнотекстовом поиске. Полнотекстовый поисковый индекс включает в себя перечень всех слов, встречающихся в проиндексированных документах, и указание мест, в которых данные слова встречаются. Такой подход позволяет быстро искать в данных практически любые фразы.

Иначе говоря, индекс поисковой системы – база данных поисковой системы, в которой содержатся сведения о параметрах сайтов и копии всех найденных поисковым роботом страниц. Безусловно, индекс не виден для пользователей поисковой системы, к нему не обращаются напрямую. Пользователям предоставляется графический интерфейс, который использует специальное программное обеспечение для обработки поисковых запросов с использованием индекса.

* 1. **Программное обеспечение поисковой системы и графический интерфейс пользователя**

Когда пользователь вводит запрос, поисковая система производит поиск в базе и с помощью алгоритмов определяет актуальность данных, составляя рейтинг найденных страниц. Результаты, которые считаются более релевантными для пользователя поисковой системы, получают более высокий рейтинг. Результаты предоставляются пользователю по убыванию рейтинга. Однако иногда возникает необходимость получения результатов за определенный период или на основе какого-то критерия. Для этого всем пользователям поисковой системы предоставляется возможность сортировки и фильтрации полученных результатов, разбиение результатов по категориям в зависимости от источников.

Итак, программное обеспечение информационно-поисковой системы является ответственным за построение таблицы релевантности из данных с поискового сервера, исключение страниц, предназначенных для другого региона или недоступных для данной категории пользователей, верстку страницы выдачи результатов поиска. Также сейчас многие поисковики дополнительно добавляют рекламу.

* 1. **Обзор современных поисковых систем**

Во время путешествия по Интернету, вам обязательно понадобиться помощь поисковой машины. Очень часто приходится искать информацию в сети, не зная даже приблизительно адрес страницы, на которой она может располагаться. В таких случаях на помощь приходит поисковая машина. Среди ведущих поисковых систем на данный момент – Google, Яndex, Mail.ru, Rambler, Bing, Yahoo, AQL, Ask [4].

Рейтинг популярности поисковых систем можно оценить как долю поискового трафика, как количество посещений поисковой страницы, по количеству вопросов. В мировом рейтинге лидером, безусловно, является поисковик Google. Это самая популярная поисковая система в мире с долей на рынке 68,69 %, а значит, семь из десяти находящихся в сети людей обращаются к его странице в поисках информации в Интернете [5]. Важно отметить, что поисковые системы часто имеют разную популярность в зависимости от региона. Рассмотрим отдельно рейтинги поисковых систем в США, России и Беларуси.

Среди поисковиков в США лидером является Google (рисунок 1.2). Однако в 2016 году выяснилось, что у Google появился реальный конкурент. Это поисковая система Bing, выпущенная в 2009 году всем известной корпорацией Microsoft, предпочтение которой отдают 19,80% пользователей.



Рисунок 1.2 – Самые популярные поисковые системы в США

В России на сегодняшний день самой популярной поисковой системой является Яндекс – 50.34% всех поисковых запросов (рисунок 1.3).

Республика Беларусь на протяжении долгого времени оставалась единственной страной на планете, где Google и Яндекс имели одинаковые доли на местном рынке [6]. Именно такая ситуация наблюдалась в 2014 году, когда эти компании владели долей рынка в размере 42% каждая. За ними следовал поисковик Mail.ru с 10%. В 2015 году ситуация изменилась. На первое место вышла поисковая система Google, которой теперь отдают предпочтение 49,5% посетителей. Доля Яндекса снизилась до 40,1%, а Search.Mail.ru – до 6,5%. Национальный поисковик Tut.by расположился на четвертом месте рейтинга с 2,8%. Замыкает пятерку лидеров Bing, которому досталось 0,6% рынка. Рейтинг поисковых систем Беларуси представлен на диаграмме на рисунке 1.4.



Рисунок 1.3 – Самые популярные поисковые системы России

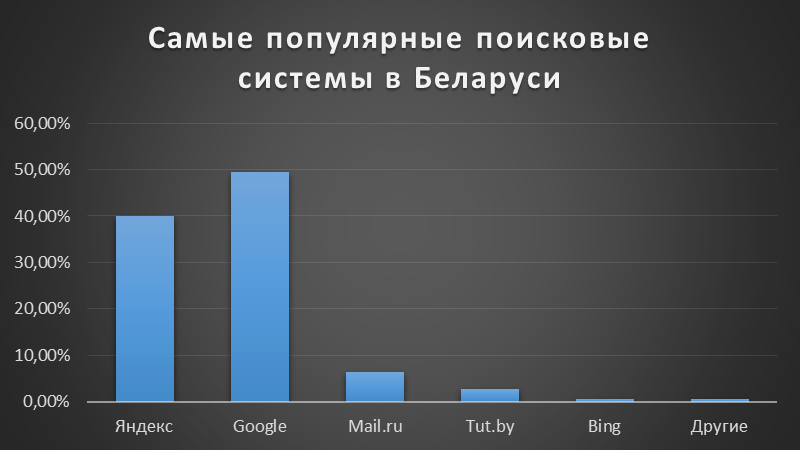


Рисунок 1.4 – Самые популярные поисковые системы в Беларуси

Интересно, что год выпуска поисковой системы мало влияет на ее популярность. Большую роль в выборе той или иной поисковой системы пользователи отдают полноте и точности результатов поиска. Важно отметить также, что каждая поисковая система предоставляет различные возможности для поиска информации, поэтому нельзя однозначно определить какая из систем является наилучшей.

* 1. **Корпоративные поисковые системы и их особенности**

Поиск данных в локальной сети организации имеет, с одной стороны, много общих черт с поиском во Всемирной паутине, а с другой стороны, заметно от него отличается. Обусловлены эти отличия, конечно же, отличиями самих тех сред, в которых производится поиск информации.

Во-первых, корпоративная сеть, в отличие от Web, достаточно разнородна в плане источников данных. Если во Всемирной паутине львиная доля всей информации, с которой работают поисковые системы, приходится на HTML-страницы, то в корпоративной среде наблюдается настоящий «зоопарк» из хранилищ файлов различных форматов, баз данных и других репозиториев информации. Во-вторых, если в интернете вся информация, которая доступна поисковой системе, доступна и любому из её пользователей, то в случае с корпоративным поиском показывать любые документы каждому из сотрудников будет в корне неправильным в силу политик информационной безопасности организации. То есть, корпоративная поисковая система должна, ко всему прочему, поддерживать разграничение доступа к искомым данным. В-третьих, корпоративная поисковая система должна интегрироваться с другими информационными сервисами, действующими в рамках ИТ-инфраструктуры данной организации, в то время как поисковые системы во Всемирной паутине предоставлены сами себе и никакой интеграции ни с кем обеспечивать, в общем-то, не обязаны.

В то же время технологии поиска, применяемые в корпоративных «поисковиках», уже, как правило, отработаны на поисковых системах, работающих в интернете. Интерфейсы корпоративных средств поиска тоже, как правило, выполнены по образу и подобию своих онлайновых «коллег».

Необходимость обеспечения надежной защиты значительно поднимает стоимость корпоративных продуктов, однако на рынке имеются различные варианты поисковых систем с разной степенью требований к информационной системе предприятия. Для сравнения: первый корпоративный продукт Google, выпущенный в 2002 году, продавался по цене от 30 тыс. долл. и осуществлял поиск среди 100 тыс. и более документов (сегодня, как мы уже упоминали, компания Google готовит новую поисковую систему Оne box, которая предложит заказчикам дополнительные средства защиты и другие функции). В то же время компания Google надеется расширить свой бизнес поисковых устройств, охватив им более мелкие предприятия с меньшим числом документов и более скромным бюджетом. Им предлагается поисковая система Google Mini, которая подключается к корпоративной интрасети и может обрабатывать до 50 тыс. документов. Стоимость такой системы на Google.com — около 5 тыс. долл.

Однако использование различных вариантов корпоративных поисковых систем может создать угрозу безопасности. Например, неприятный инцидент произошел с поисковой системой Google. Поисковый робот добавил в свою базу документ, содержащий конфиденциальную информацию об учащихся одной из американских школ. Хотя пострадавшая сторона уверяет, что доступ к этому документу был закрыт паролем, все персональные сведения (включая имена, номера социального страхования и показатели успеваемости) более 600 учащихся были доступны любому пользователю Интернета [7].

Средства корпоративного поиска — уже давно не новинка на софтверном рынке, но все же данное направление до сих пор находится в состоянии процесса формирования. Информационный поиск стал основой работы в сети интернет. А корпоративный поиск еще только выходит на уровень достаточно зрелого самостоятельного рынка.

## Анализ мирового рынка корпоративных поисковых систем

Уже давно поиск широко и успешно применяется при работе в открытом и почти бесконечном пространстве Интернета, но при этом многие ИТ-пользователи испытывают значительные трудности в деле нахождения нужной информации в рамках своей корпоративной системы и даже на отдельном собственном ПК. Однако, можно утверждать, что на рынке есть довольно широкий спектр предложений по решению задач поиска, от приложения локального уровня до систем масштаба предприятия.

Можно выделить три категории корпоративных поисковых систем по функциональности: "тяжелые", "средние" и "легкие".

"Тяжелые" системы поиска оснащены множеством функций поиска различных типов документов и предназначены для интеграции в корпоративные информационные системы. Они учитывают специфику организации локальных сетей, баз данных, систем хранения и резервного копирования данных, электронного документооборота и почтовых систем предприятий. Обработка поисковых запросов в "тяжелых" системах позволяет учитывать логические и морфологические связи, синонимичность, исправлять опечатки, проводить глубинный анализ текстов [8].

"Средние" системы обладают облегченной функциональностью, однако в них часто предусмотрено усиление определенных функций или групп функций. Этот класс систем ориентирован на поиск в локальной сети. Системы "среднего класса" не предусматривают возможности разделения доступа к информации на уровне запроса, индекса или источника данных. Кроме того, в таких системах не всегда есть исправление опечаток запроса, ограничено число поддерживаемых форматов документов.

"Легкие" системы предназначены для решения определенного типа задач - локального поиска данных. Как правило, они поддерживают форматы МS Office, PDF и HTML/TXT, позволяют работать с архивами почтовых систем и сетевых мессенджеров.

Спектр наиболее известных решений, доступных на мировом рынке, наглядно представлен в таблице 1.1. Для сравнения предлагаемых на рынке продуктов следует выделить набор факторов, определяющих интерес потенциальных заказчиков к решениям в сфере поиска информации в корпоративных сетях. При анализе рынка поисковых систем и возможностях их использования в Республике Беларусь следует выделить функцию поддержки русского языка, а также корпоративные функции, определяющие применимость решения и удобство использования в корпоративной среде.

Таблица 1.1 – Мировой рынок корпоративного поиска

|  |  |  |
| --- | --- | --- |
| Компания | Продукты | Уровень |
| Vivisimo | Velocity | тяжелый |
| dtSearch Corp. | dtSearch | средний |
| Google | Google Desktop | легкий |
| Google | Google Search Appliance | тяжелый |
| IBM | Lotus Extended Search | тяжелый |
| ISYS | ISYS Full Product Suite | тяжелый |
| Microsoft | MSN Desktop Search | легкий |
| Microsoft | SOL Server | тяжелый |
| Oracle | Ultra Search | тяжелый |
| YourAmigo | Enterprise Search | средний |
| Sybase | Enterprise Portal | тяжелый |
| Yahoo! | Yahoo! Desktop Search | легкий |

Среди поисковых функций обычно выделяют:

* 1. Поддержка запросов на естественном языке.
  2. Поиск с учётом опечаток и словоформ.
  3. Возможность фильтрации результатов поиска (время, география, язык, тип результатов).
  4. Возможность автоматического структурирования/группировки результатов поиска.
  5. Количество поддерживаемых форматов файлов.
  6. Поддержка Unicode (для возможности работы с русским языком).
  7. Поддержка морфологии русского языка.

Корпоративные функции:

1. Поддержка федеративного доступа к данным.
2. Поддержка разделения доступа и работы с конфиденциальной информацией.
3. Сложность интеграции с другими продуктами компании.
4. Поддержка распределённых корпоративных сетей.
5. Поддержка безаварийной работы.
6. Поддержка сбора статистики запросов.
7. Наличие программного интерфейса (API).
8. Способ лицензирования/ценообразования.

Наиболее функционально развитыми являются поисковые системы, предназначенные для работы с англоязычными документами. Работа с русским языком в решениях зарубежных поставщиков поддерживается на уровне поиска слов по точному совпадению (требуется лишь поддержка Unicode), поиск словоформ и морфологический поиск (с учётом опечаток, словоформ и синонимов), как правило, либо не поддерживается, либо поддерживается с помощью модулей отечественной разработки. Решения отечественных поставщиков опережают зарубежные продукты по глубине анализа текстов на русском языке, но обладают слабой функциональностью в остальных областях. В частности, поддерживают небольшое число форматов файлов, слабо реализована поддержка индексации данных в корпоративных СУБД, нет функций разделения доступа и работы с конфиденциальной информацией.

Рассмотрим ценовую сегментацию решений. Данные о стоимости лицензий приведены в таблице 1.2. Стоимость лицензий указана по данным на март 2017 года. Из таблицы видно, что самым дорогим решением является Ultra Search от Oracle. Приведенная таблица также позволяет выделить ценовые сегменты рынка корпоративных поисковых систем. Для "тяжелых" систем, предназначенных для работы с большими объемами корпоративных документов и значительным числом пользователей, цена не составит менее $25 тыс. Следует обратить внимание на средний ценовой сегмент, который отличается наиболее благоприятным соотношением цена/функциональность. Ценовой разброс здесь $10000-$25000. Стоимость "легких" систем составляет около $3000. Локальные утилиты поиска чаще всего распространяются бесплатно, как, например, система MSN Desktop Search, поставляемая компанией Microsoft.

Таблица 1.2 – Ценовая сегментация решений

|  |  |
| --- | --- |
| Продукт | Стоимость лицензии в год, $ |
| Velocity | от 10000 |
| dtSearch | 23000 |
| Google Desktop | 2995 |
| Google Search Appliance | 25000 – 78000 |
| Lotus Extended Search | 9995 |
| ISYS Full Product Suite | 28100 |
| MSN Desktop Search | бесплатно |
| SOL Server | 24999 |
| Ultra Search | от 100000 |
| Enterprise Search | $15000 |
| Enterprise Portal | $38000 |
| Yahoo! Desktop Search | бесплатно |

Примечательным является также факт, что стоимость поисковых систем российских поставщиков, относящиеся к классам "тяжелых" и "средних", наиболее низка. Этот фактор учитывает платежеспособность российских предприятий, а также психологическую готовность российских потребителей.

Представим данные о средней стоимости корпоративной поисковой системы в зависимости от сложности в виде диаграммы (рисунок 1.5).

Рисунок 1.5 – Средняя стоимость корпоративных поисковых систем

Отметим, что наибольший интерес для рассмотрения представляют лишь «тяжелые» и «средние» решения, так как производится оценка в первую очередь многофункциональных поисковых систем, подходящих крупным компаниям с большим количеством внутренних веб-ресурсов.

Итак, одним из главных технических требований при выборе поисковой системы является поддержка разнородности документов корпоративной ИТ-среды, которая включает порой десятки различных систем и хранилищ данных (в отличие от Интернета с достаточно однородной средой). Следующим очень важным требованием является необходимость учета профессиональных ролей сотрудников-пользователей и в решении вопросов управления правами доступа к документам (чего нет в случае Интернета). Отсюда вытекает финансовый аспект: приобретение, внедрение и поддержка серьезного поискового решения корпоративного уровня требует весьма значительных денежных и интеллектуальных затрат. В это время заказчики, давно привыкшие к бесплатности интернет поиска, порой просто не понимают, почему они должны платить деньги, причем немалые, за “то же самое”. Эффективность и окупаемость поисковых средств являются совсем не очевидными для потребителей.

Высокая стоимость корпоративных поисковых систем и возможность утечки информации при нарушении работы сторонней системы могут стать основными причинами создания собственной защищенной корпоративной информационно-поисковой системы.

1. **АНАЛИЗ ПРОЦЕССА ПОИСКА ИНФОРМАЦИИ И СИСТЕМ КОРПОРАТИВНОГО ПОИСКА**
   1. **Общая характеристика компании EPAM Systems**, **сотрудничество с компанией SAP**

Компания EPAM Systems основана в 1993 году. Полное название предприятия: иностранное общество с ограниченной ответственностью «ЭПАМ Системз». Компания является крупнейшим разработчиком заказного программного обеспечения и ведущим поставщиком ИТ-услуг с отделениями в Центральной и Восточной Европе. Штаб-квартира компании расположена в США, штат сотрудников насчитывает более 19600 инженеров и IT-консультантов [9].

В 2013 году EPAM заняла 6 место в списке Forbes «25 самых быстрорастущих технологических компаний Америки». По данным издательского дома «Коммерсант» EPAM Systems возглавляет список крупнейших разработчиков программного обеспечения в России. Компания входит в число лидеров в сфере оказания услуг по разработке программных продуктов по мнению аналитических компаний Forrester и Zinnov.

EPAM Systems неизменно является признанным лидером в таких областях, как:

* разработка, тестирование, сопровождение и поддержка заказного программного обеспечения бизнес-приложения;
* интеграция приложений на базе продуктов SAP, Oracle, IBM, Microsoft;
* миграция приложений на новую интеграционную платформу;
* создание выделенных центров разработки (центров компетенции), центров тестирования и контроля качества программного обеспечения.

Преимуществами сотрудничества с EPAM Systems пользуются сотни компаний из различных секторов экономики, в том числе:

* банки и финансовые компании;
* страхование;
* поставщики программного обеспечения;
* розничная торговля и потребительские товары;
* информационный и медиа-бизнес;
* индустрия путешествий;
* телекоммуникации;
* нефть, газ и энергетика;
* автобизнес;
* государственный сектор и др.

Основными заказчиками являются: SAP, Microsoft, Oracle, Thomson Reuters, The Coca-Cola Company, Лондонская фондовая биржа, ММВБ, Viacom/MTV Networks, Wolters Kluwer, "Ренессанс-Капитал", Американский Красный Крест, Clarins Group и многие другие.

Компания EPAM Systems занимает особое место среди партнеров SAP в СНГ. Взаимодействие двух компаний началось в 1997 году, когда EPAM стала участвовать в разработке программных продуктов SAP. В дальнейшем приобретенные компетенции были использованы при разработке сложных и уникальных решений на основе SAP для крупных заказчиков по всему миру. Уже более десяти лет в компании функционирует выделенный центр компетенций SAP - это специализированное подразделение EPAM Systems, которое задействовано во всех проектах, так или иначе связанных с использованием технологий SAP. К основным направлениям деятельности центра относится участие в создании программных продуктов SAP, разработка решений на базе SAP-технологий для независимых производителей программного обеспечения. Осуществляется реализация проектов для клиентов SAP (разработка портальных и BI-решений, хранилищ данных, систем для стратегического управления и поддержки принятия решений и т.д.), а также внедрение и сопровождение решений SAP [10].

С 2004 года в компании EPAM Systems в рамках центра компетенций SAP действует консалтинговая практика, специалисты которой сфокусированы на разработке, внедрении и развитии ИТ-систем на основе SAP для клиентов из России и стран СНГ. Еще одно направление деятельности консалтинговой практики EPAM - разработка и внедрение аналитических приложений на основе продуктов SAP BusinessObjects.  
Сегодня в число заказчиков, выбравших услуги EPAM для разработки, внедрения, интеграции, сопровождения систем на базе SAP, входят такие организации, как "Газпром нефть", "Концерн Росэнергоатом", ОАО "Силовые машины", Национальный банк Украины, MERX, "Мозырский нефтеперерабатывающий завод", ТКС, ГК "Армтек", Корпорация "Богдан", "Белорусский металлургический завод" и другие.

EPAM Systems предлагает широкий набор услуг в области разработки корпоративных систем и консалтинга: от анализа и формализации бизнес-процессов до внедрения систем класса ERP, BI, CRM и других бизнес-приложений, основанных в первую очередь на продуктах и технологиях SAP. Уже более 15 лет SAP и EPAM Systems связывают многоуровневые партнерские отношения. Компания сотрудничает с SAP AG, SAP Labs, SAP America, SAP СНГ. Работая с SAP как разработчик, ЕРАМ принимала участие в создании продуктов SAP NetWeaver, SAP xApps (композитные кросс-приложения). Сейчас выделенная команда EPAM Systems занимается разработкой новых технологий, которые SAP будет применять через два-три года.

ЕРАМ Systems внедряет практически всю линейку продуктов SAP, имеет большой опыт работы с SAP NetWeaver, SAP BI, SAP BusinessObjects, в разработке портальных решений, систем для стратегического управления и поддержки принятия решений, создании хранилищ данных, интеграции и миграции приложений на основе SAP.

* 1. **Общий анализ и характеристика компании SAP**

SAP SE – производитель [программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) для организаций. Штаб-квартира расположена в [Вальдорфе](https://ru.wikipedia.org/wiki/%D0%92%D0%B0%D0%BB%D1%8C%D0%B4%D0%BE%D1%80%D1%84) [11].

Компания занимается разработкой автоматизированных систем управления такими внутренними процессами предприятия, как: бухгалтерский учёт, торговля, производство, финансы, управление персоналом, управление складами и т. д. Приложения обычно можно адаптировать под правовой контекст определённой страны. Кроме поставок программного обеспечения, фирма предлагает услуги по его внедрению, используя для этого собственную методологию внедрения (изначальное название ASAP — Accelerated SAP, впоследствии — ValueSAP).

Самый известный продукт компании — ERP-система SAP R/3, ориентированная на крупные и средние предприятия, разрабатываемая и продаваемая компанией с начала 1990-х годов. Создана в продолжение линеек RF (позднее идентифицированной как R/1) и R/2. Начиная с выпусков середины 2000-х годов название R/3 не используется; ядро ERP-системы, созданной в продолжение линейки, производитель называет SAP ERP.

Решения (программное обеспечение и комплексные системы) SAP помогают предприятиям во всем мире совершенствовать взаимоотношения с клиентами, расширять совместную деятельность с партнерами и повышать эффективность деятельности компаний. Основой для создания подобных программных комплексов является уникальная разработка компании - платформа SAP NetWeaver. Специализированные отраслевые пакеты SAP эффективно работают в таких областях деятельности, как наукоемкие производства, машиностроение, энергетика, сфера высоких технологий, здравоохранение, образование, государственный и финансовый сектора и многие другие.

Технологии SAP представляют собой систему управления предприятием и включают в себя как управление ресурсами (сырьевыми, производственными, человеческими), так и систему реализации проектной (научной, образовательной, иной специализированной) и производственной деятельностью.

Компания SAP имеет представительства в более чем 50 странах мира.

В 1992 году открылось представительство SAP в Москве. Открылись представительства в Санкт-Петербурге, Алматы, Киеве и Минске, а численность сотрудников составила около 7000 человек. В странах СНГ и Балтии осуществлены 974 инсталляции программных комплексов и решений SAP.

Из 183 тысяч клиентов подавляющее большинство составляют средние и крупные предприятия. Клиенты SAP в различных областях представлены на рисунке 2.1. По подсчётам продукты SAP используют около 12 миллионов пользователей, работающие на существующих 91 500 установленных копиях.

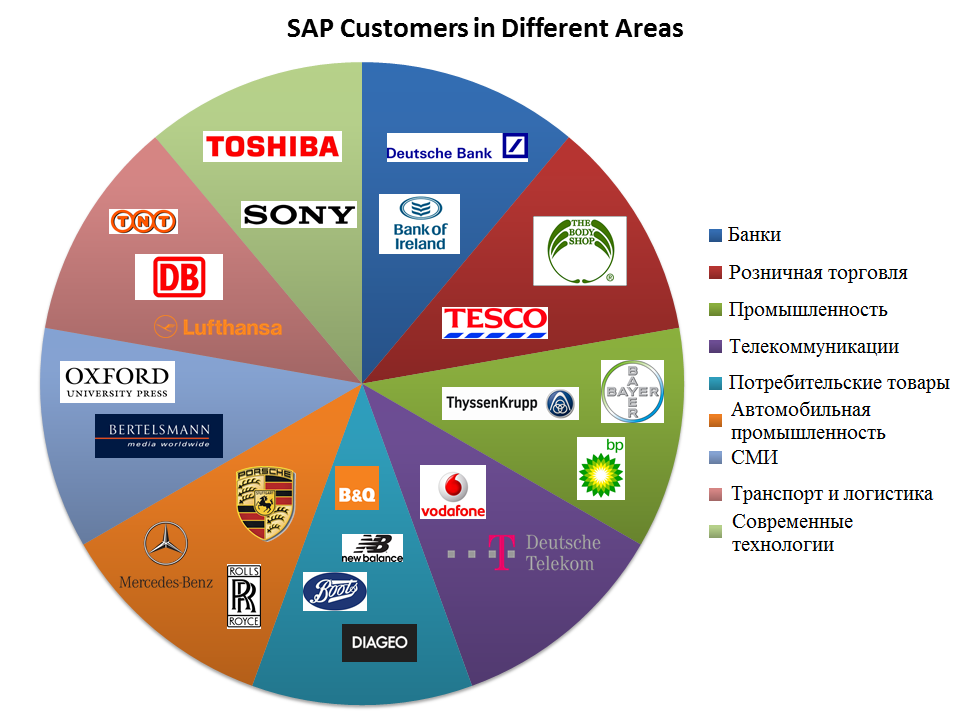


Рисунок 2.1 – Клиенты SAP в различных сферах

В 2007 году более 39 400 компаний, расположенных в более чем 120 странах мира использовало решения SAP. На 2008 год фирма насчитывала порядка 43,8 тысяч сотрудников. В 2015 году в компании работало 76,9 тысяч сотрудников по всему миру.

Итак, компания SAP является ведущим в мире поставщиком программных решений для управления бизнесом и предлагает решения и услуги, которые позволяют предприятиям любого масштаба, работающим более чем в 25 отраслях, повысить эффективность бизнеса. Сейчас компания активно сотрудничает с EPAM Systems. При этом EPAM Systems участвует в создании и развитии программного обеспечения SAP, такого как системы принятия решений, хранилища данных и информационный поиск. Процесс поиска информации в компании SAP неоднократно анализировался и подвергался обсуждениям. Каждый сотрудник компании ежедневно ищет информацию среди электронных ресурсов компании. А так как SAP является крупной компанией с большим количеством внутренних сервисов, поиск информации может занимать большую часть рабочего времени сотрудников. Именно поэтому в рамках дипломного проектирования проведена работа по исследованию и поиску вариантов улучшения процесса поиска в компании SAP.

* 1. **Анализ процесса поиска информации в компании SAP**

Поиск — это уже давно один из ключевых механизмов доступа пользователей к нужной им информации. Парадокс же заключается в том, что сегодня многие люди проще решают подобные задачи при работе в открытом и почти бесконечном пространстве Интернета и при этом испытывают значительные трудности в рамках своей корпоративной системы и даже на отдельном собственном ПК. Сотрудники многих компаний считают, что найти корпоративную информацию гораздо сложнее, чем открытую в Сети. Объяснение этому понятно: на предприятиях имеются порой десятки различных разнородных систем и хранилищ данных. Обычно каждое решение имеет свои собственные средства поиска, но они, как правило, функционируют только внутри своей системы.

SAP – крупная компания с большим количеством сотрудников по всему миру. Компания была основана около 45 лет назад и все это время ее внутренние электронные ресурсы постоянно росли. Сейчас в компании множество внутренних сервисов, таких как система управления расходами сотрудников в командировках, сервис учета рабочего времени, система оплаты труда, сервис поддержки заказчиков, а также большой архив документации по всем созданным продуктам компании.

В компании был проведен опрос, по результатам которого определено, что 52% сотрудников на поиск информации тратят не менее двух часов в день, 27% утверждают, что на поиск информации они тратят около часа. И лишь 21% сотрудников говорят, что тратят лишь пару минут в день (рисунок 2.2).

Рисунок 2.2 – Время, затрачиваемое сотрудниками компании SAP на поиск информации внутри компании

Важно понимать, что такие представленные результаты были получены при опросе сотрудников различных должностей. Все зависит от вида деятельности. Менеджеры, выполняющие руководящие функции, тратят на поиск меньше времени, за них поиск документов выполняют подчиненные. А инженеры, занимающиеся вопросами соблюдения законодательных и иных требований, постоянно приходится искать разные документы. Для них поиск и сверка электронных документов – основная часть работы. Следует также учесть различия между инженерами, выполняющими разные обязанности. Потому каждой категории персонала соответствуют свои временные затраты на поиск документов и свой потенциал снижения этих затрат.

Рассмотрим подробнее, как зависит время, затрачиваемое сотрудником компании от его должности. В компании SAP определены пять уровней должностей:

* младший сотрудник (например, младший разработчик, младший помощник менеджера, младший помощник по поддержке и управлению, младший инженер технической поддержки);
* специалист (например, разработчик, специалист технической поддержки, специалист по управлению);
* старший специалист (например, старший разработчик, старший специалист технической поддержки);
* эксперт, или ведущий специалист;
* руководитель (например, начальник отдела поддержки).

Зависимость времени, которое тратит сотрудник на поиск электронных ресурсов в компании, от должности сотрудника представлена на рисунке 2.3.

Рисунок 2.3 – Зависимость времени на поиск информации от должности сотрудника

Младшие сотрудники, только прибывшие на предприятие, тратят более трех часов в день, чтобы найти документы, требуемые им для выполнения своих прямых обязанностей. В то же время ведущие специалисты и руководители делегируют эти задачи подчиненным.

Как говорилось ранее, в компании SAP по данным на 2015 год работало более 76,9 тысяч сотрудников по всему миру. Из них более половины занимают должности первого или второго уровня – младшие специалисты и специалисты. А это значит, что около 38 тысяч сотрудников в среднем 2-2,5 часа уделяют переходу по ссылкам и просмотру страниц и электронных документов, чтобы найти данные.

Из 57% сотрудников, поиск информации для которых занимает значительную часть времени, 81% утверждают, что поиск информации – одна из их непосредственных обязанностей. А значит, следует сконцентрировать внимание на этой группе и выяснить, что мешает осуществлять поиск быстрее. Результаты исследования представлены на рисунке 2.4.

Рисунок 2.4 – Факторы, затрудняющие поиск информации сотрудниками компании

На рисунке 2.4 видно, что распределенная информация затрудняет поиск, 85% людей считают это основной проблемой. Уровни доступа являются проблемой лишь для 9% сотрудников. Под уровнем доступа в данном случае понимаются неправильные настройки сервисов и невозможность получить доступ к информации без получения помощи третьих лиц. 6% считают, что сложность при поиске обусловлена в первую очередь разнородностью информации.

Кроме того было выяснено, что для экономии времени на поиск информации среди электронных ресурсов компании, сотрудники начинают сохранять ценную информацию на своих рабочих компьютерах. Однако некоторые из таких сотрудников позднее сталкиваются со следующей проблемой: невозможностью найти информацию на своем ПК.

Полезная информация часто оказывается настолько труднодоступной, что иногда работникам проще было создать ее заново. А с уходом ключевого сотрудника важные данные просто утрачивались.

Представим, что в компании нет корпоративной поисковой системы и нет общего каталога ссылок всех ресурсов. Рассмотрим ситуацию, когда сотруднику необходимо найти, например, бланк заявления на отпуск. Что в такой ситуации будет делать молодой неопытный сотрудник компании? Он обратится к начальнику с вопросом: «Подскажите, где я могу найти пример заявления?» Начальник не станет искать заявление сам, он посоветует молодому специалисту обратиться в отдел кадров. Молодой специалист, вероятно, может и не знать, где находится отдел кадров, поэтому потратит какое-то время, чтобы уточнить это у коллег. Далее он пойдет в нужный ему отдел, где выяснится, что нужный ему сотрудник с примерами бланков в данный момент в отпуске. После этого молодой специалист вернется на свое рабочее место и, открыв браузер, введет поисковый запрос в интернете. По результатам запроса «заявление на отпуск» ему будет предложено около 591 тысячи результатов (рисунок 2.5).

После этого неопытный сотрудник перейдет на несколько ссылок с первой страницы результатов поиска. Он поймет, что каждый раз ему предлагаются различные формы заявления. Некоторые из них строгой формы, некоторые в свободной форме. Так как молодые специалисты склонны относиться к работе серьезнее опытных и давно работающих людей, сотрудник посчитает строгую форму документа правильной.

Для составления заявления на отпуск теперь молодому специалисту понадобится знать: полное название структурного подразделения, регистрационный индекс, адрес предприятия и прочие реквизиты заявления. Для получения этой информации необходимо будет вновь побеспокоить коллег или поискать реквизиты среди других документов в компании. Очевидно, что на составление простого заявления на отпуск у сотрудника уйдет не менее часа рабочего времени, которое он мог бы провести с пользой для компании, выполняя свои непосредственные обязанности. Сколько же усилий пришлось потратить неопытному сотруднику для поиска документа в компании, где нет системы корпоративного поиска? Представим ответ на этот вопрос в виде схемы на рисунке 2.6.

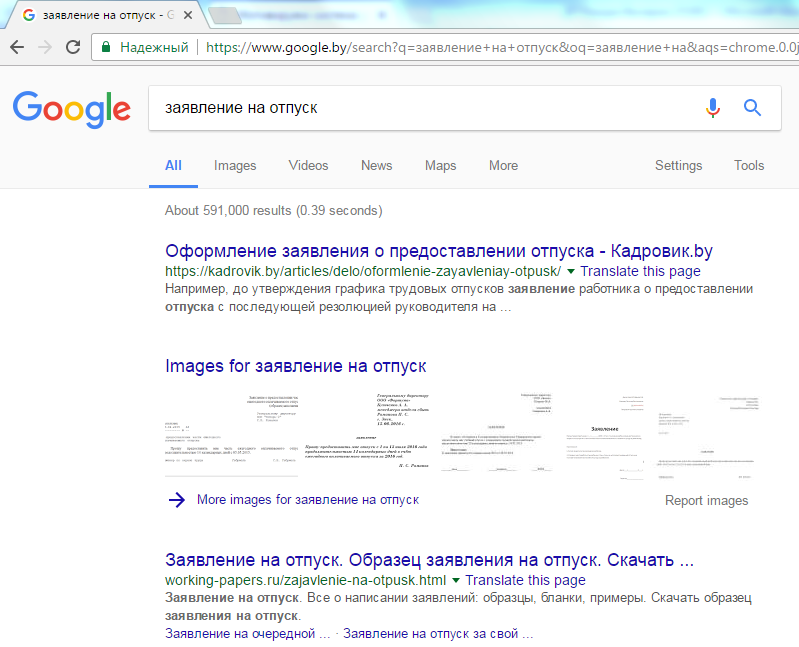


Рисунок 2.5 – Пример результатов поиска по запросу «заявление на отпуск»



Рисунок 2.6 – Действия молодого специалиста для поиска заявления на отпуск в компании

Чтобы избежать этих и других проблем, связанных с поиском информации, необходима хорошо налаженная система поиска. Это не только облегчит доступ сотрудников к знаниям, но и позволит работникам целесообразно использовать свое рабочее время. Ввод поискового запроса занимает считанные секунды, пару минут займет просмотр результатов поиска. Но больше не потребуется посещать всевозможные ссылки самостоятельно и тратить на это часы.

В компании «SAP» система корпоративного поиска представлена каталогом веб-ресурсов компании. Эта система, несомненно, упрощает поиск и доступ к электронным документам компании, но не решает проблему полностью.

Рассмотрим основной процесс работы информационно-поисковой системы: поиск информации в компании. Так как задачей проектируемой системы является совершенствование и оптимизация уже существующего поиска электронных ресурсов в компании SAP, то будут составлены две модели. Первая модель «AS-IS» отобразит, как был организован поиск. Вторая диаграмма «TO-BE» представит улучшенную модель работы информационно-поисковой системы.

На контекстной диаграмме верхнего уровня «AS-IS» на рисунке 2.7 представлен основной процесс – «Найти информацию в компании».

Определим входные и выходные потоки, основные механизмы и управление.

Входными данными являются электронные ресурсы компании и поисковой запрос, который формируется сотрудником компании, когда ему необходимо что-либо найти. Электронные ресурсы в данном случае представляют собой набор веб-страниц, размещенных на внутренних серверах компании.

Управляющими механизмами являются:

1. Политика конфиденциальности компании. Она представляет собой набор правил по работе с документами компании, описывает процесс сбора, хранения, использования и распространения внутренней информации компании. Политика включает инструкции по пользованию веб-ресурсами компании, определяет доступность ресурсов для каждой категории работников.
2. Правила работы с электронными документами компании. В данных правилах описаны основные инструкции по электронному документообороту, особенности хранения электронных документов. Здесь также предоставляются инструкции по работе с защищенными документами, а также основные правила именования документов.
3. Требования к искомой информации определяются сотрудником, желающим найти необходимые ему ресурсы компании. Эти требования могут быть представлены ожидаемым объемом полученных по запросу результатов, ожиданиями относительно релевантности полученных результатов, представлениями сотрудника о формате представления искомой информации.

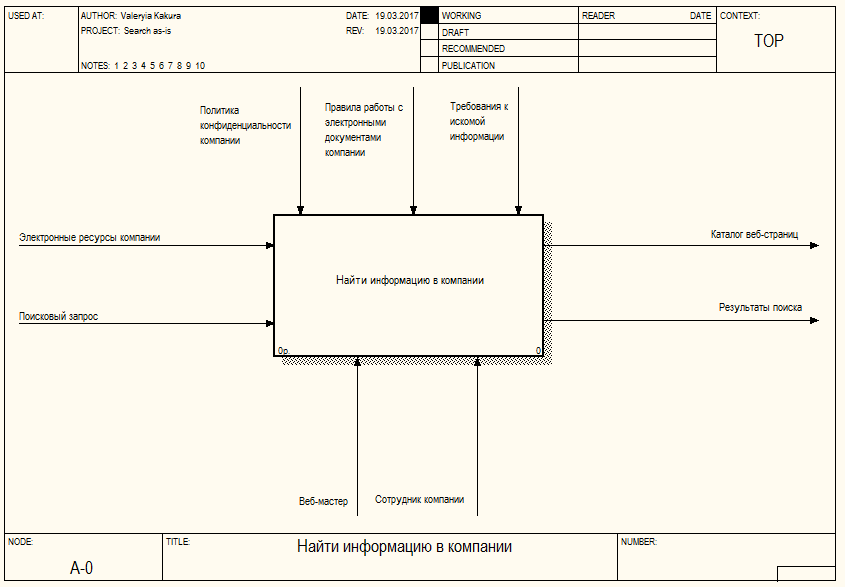


Рисунок 2.7 – Контекстная диаграмма «Найти информации в компании»

Механизмами реализации являются:

* веб-мастер;
* сотрудник компании.

На выходе получаем результаты поиска, предоставленные системой сотруднику, который ввел поисковой запрос, и каталог веб-страниц, созданный веб-мастером.

Процесс поиска информации в компании может быть условно разбит на два дочерних процесса:

1. Составить каталог веб-страниц.
2. Искать информацию в каталоге веб-страниц.

Декомпозиция представлена на рисунке 2.8. На рисунке видно, что составлением каталога будет заниматься веб-мастер. И только после того, как будет составлен такой каталог, сотрудник сможет им воспользоваться.

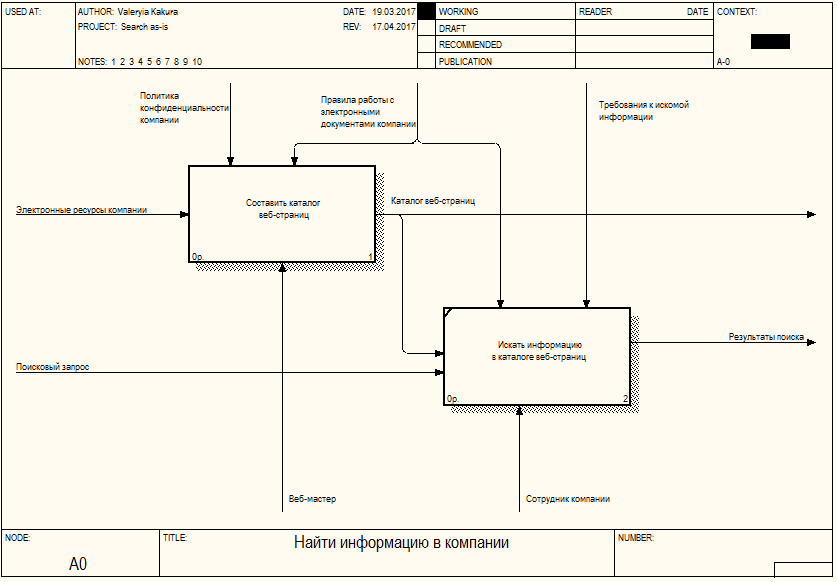


Рисунок 2.8 – Декомпозиция блока «Найти информацию в компании»

Процесс же составления каталога веб-страниц представляет собой четкую последовательность действий веб-мастера. Для начала ему необходимо изучить все предоставленные электронные ресурсы компании, составить список ссылок на каждую страницу. Проанализировать все ресурсы, учесть настройки доступа к ресурсам для различных категорий сотрудников. Определить категории, на которые можно будет условно разбить данные. После этого веб-мастер создает каталог. Он добавляет в него категории. В каждой категории он размещает соответствующие веб-страницы. После чего ему необходимо установить настройки доступа для каждой категории ресурсов. Описанные действия можно представить в виде блоков на рисунке 2.9, где отображена декомпозиция блока «Составить каталог веб-страниц».

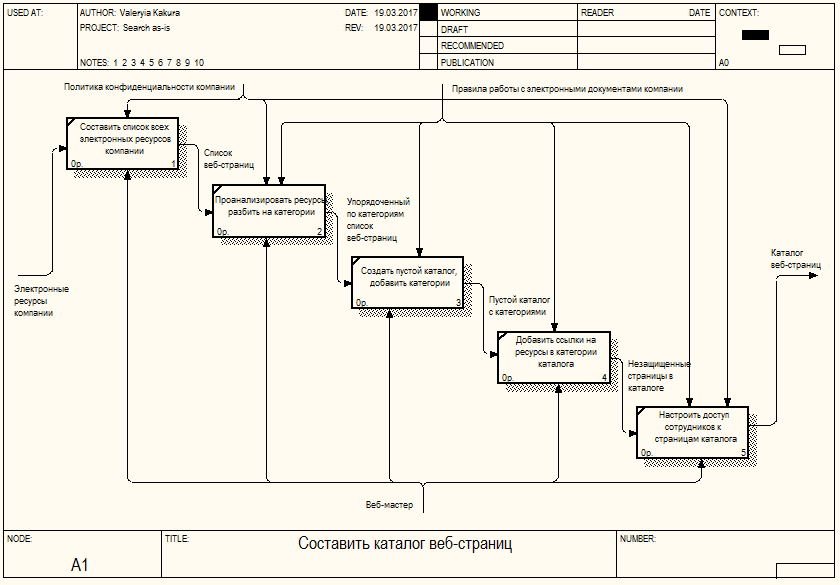


Рисунок 2.9 – Декомпозиция блока «Составить каталог веб-страниц»

На рисунке 2.9 видно, как постепенно электронные ресурсы на входе блока «Составить каталог веб-страниц» обрабатываются, на их основе составляется список всех веб-страниц. Блок «Составить список всех электронных ресурсов компании» выполняется веб-мастером. Его задачей в данном случае является сбор и предоставление веб-страниц для последующей обработки. Составление списка веб-страниц происходит вручную, что является одним из плюсов существующей системы, все ресурсы проверяются веб-мастером, качество контента будет лучше, чем при автоматическом поиске страниц. Веб-мастер должен знать обо всех существующих серверах компании, где может храниться информация, и должен руководствоваться политикой конфиденциальности компании, чтобы не отправить для обработки документы, закрытые для обычных сотрудников компании. Дальше этот список упорядочивается по категориям. После чего будет создан каталог, в него добавлены категории, и только после этого ссылки на страницы помещены в категории каталога.

Необходимо заметить, что указанная выше последовательность действий веб-мастера выполняется им не один раз. Кроме усилий, затрачиваемых на составление каталога веб-ресурсов нужно учесть сложность обновления каталога. Работа крупных IT-компаний построена таким образом, что многие документы постоянно обновляются или удаляются, появляются новые документы и ресурсы. Все это отслеживается вручную человеком, и, соответственно, содержит угрозы и возможность возникновения ошибок. Если веб-страница была удалена с внутреннего веб-сервера компании, эти изменения должны отразиться и в каталоге всех веб-ресурсов компании. Иначе каталог будет содержать несуществующую ссылку. Ручная проверка всех ссылок является постоянным и очень медленным процессом.

Перейдем к описанию второго дочернего блока «Искать информацию в каталоге веб-страниц» рисунка 2.8, проанализируем полученный каталог. Выглядит он как показано на рисунке 2.10.

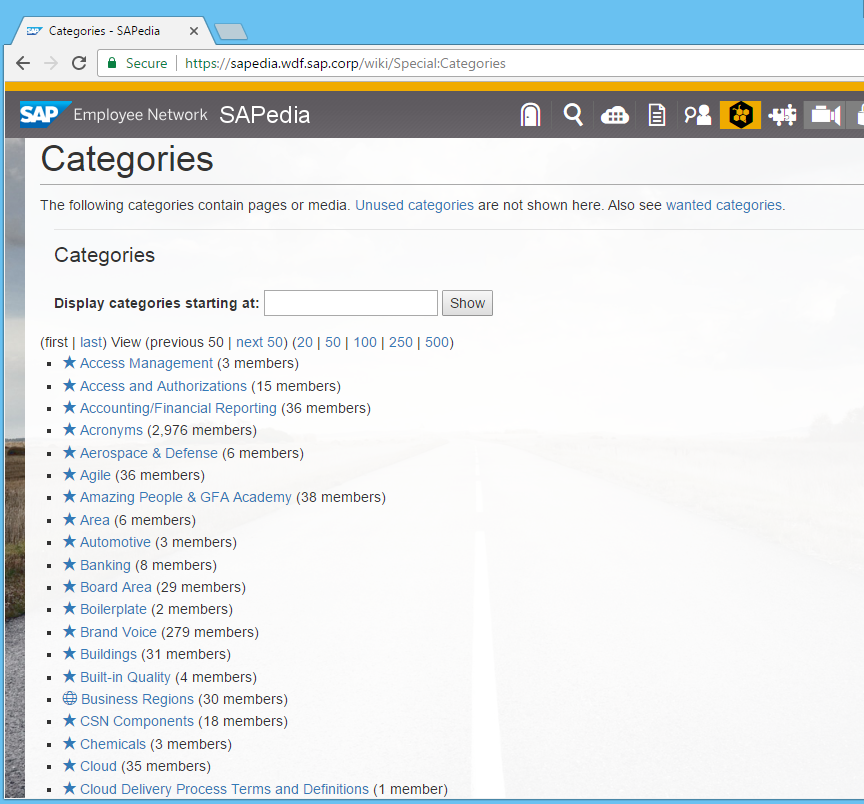


Рисунок 2.10 – Каталог с категориями, составленный веб-мастером

Сотрудник, желающий найти информацию, открывает каталог с категориями. В данном случае ему не предлагается ввести свой поисковый запрос и получить результаты. Вместо этого сотрудник сам должен определить, к какой из выданных категорий может относиться тот или иной документ или веб-страница. Вернемся к вышеописанному примеру, когда сотрудник желает найти форму заявления на отпуск. Оказавшись на странице с категориями, сотрудник должен предположить, к какой из предложенных категорий может относиться требуемая ему информация. Допустим, сотрудник уже знает, что такие заявления находятся в категории управления человеческими ресурсами - «HR». Он кликает по ссылке и переходит в выбранную категорию.

После перехода в категорию «HR» сотруднику вновь предлагается выбрать категорию, к которой может относиться искомая им информация, как показано на рисунке 2.11. Сотрудник вновь полагается на интуицию, предполагает, что инструкция по пользованию облаком будет находиться внутри категории «SAP Personal Management». И только перейдя в выбранную категорию, сотрудник найдет интересующую его форму заявления.

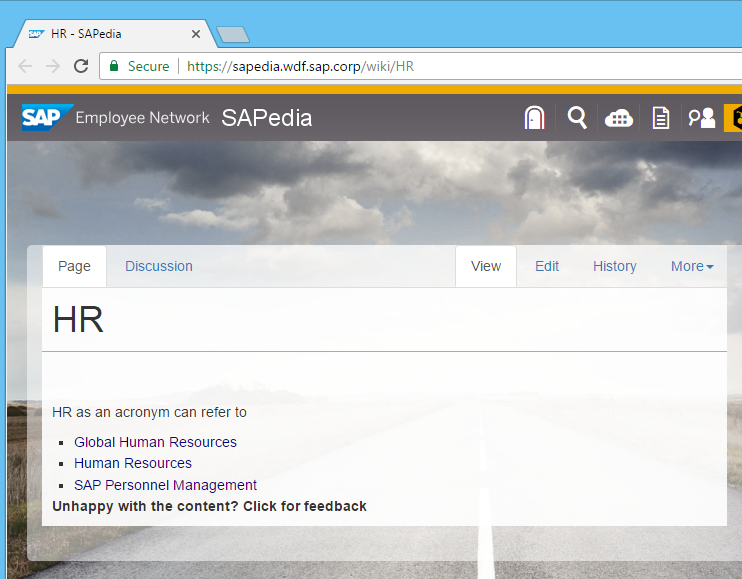


Рисунок 2.11 – Подкатегории в категории «HR»

Чем отличается последовательность действий сотрудника при использовании каталога для поиска формы заявления от его действий в случае отсутствия такой системы (рисунок 2.6)? В данном случае ему не придется ходить по офису в поиске нужных ему людей, все действия он будет осуществлять сидя за своим рабочим компьютером. Рассмотрим, как выглядит последовательность действий в таком случае на рисунке 2.12.



Рисунок 2.12 – Последовательность действий сотрудника при поиске формы заявления с использованием каталога веб-страниц

Безусловно, работа с каталогом упрощает поиск данных среди ресурсов компании. Однако такой процесс легко может быть автоматизирован. Необходим способ сделать поиск информации еще быстрее, чтобы результаты могли быть получены без лишних усилий со стороны сотрудника, лишь по вводу поискового запроса.

Рассмотрим модель «TO-BE», которая представлена на рисунке 2.13. Обратим внимание, что в модели «TO-BE» не изменились механизмы, входные данные и управление. Однако изменились выходные данные.

Чтобы узнать, почему вместо каталога с веб-ресурсами мы получим на выходе обновленный индекс с веб-страницами, проведем декомпозицию процесса поиска информации в компании на дочерние процессы.

Представим дочерние процессы:

1. Настроить и запустить краулер и индексатор.
2. Ввести поисковый запрос и получить результаты.

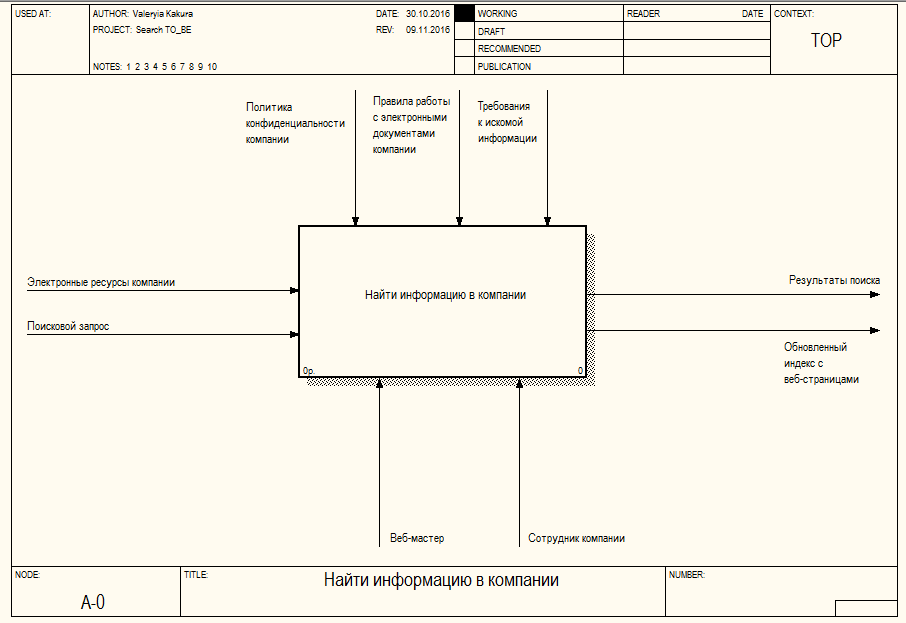


Рисунок 2.13 – Контекстная диаграмма «Найти информации в компании» в модели «TO-BE»

Как можно обратить внимание, мы условно получили два процесса при декомпозиции. Они представлены на рисунке 2.14.

Различие декомпозиции блока «Найти информацию в компании» в моделях «AS-IS» и «TO-BE» обусловлено изменением принципа обработки веб-страниц. В первом случае веб-мастер обрабатывал страницы вручную, в то время как во втором случае ему лишь необходимо ввести стартовую страницу веб-сайта, а краулер сам может переходить по всем ссылкам, обрабатывая все больше страниц.

Информация, получаемая на страницах, индексируется, после чего сотрудник компании может сразу начинать поиск необходимых ему данных. Изменится также и механизм обновления веб-страниц. Веб-мастер больше не должен самостоятельно отслеживать добавленные и удаленные ресурсы, достаточно лишь один раз настроить поисковой робот, чтобы он, к примеру, каждые несколько дней совершал обход сети компании и проверял, какие страницы еще не добавлены в индекс, а какие должны быть удалены. Все эти конфигурации могут быть установлены веб-мастером при первом запуске системы. Никаких дальнейших действий от веб-мастера не требуется.

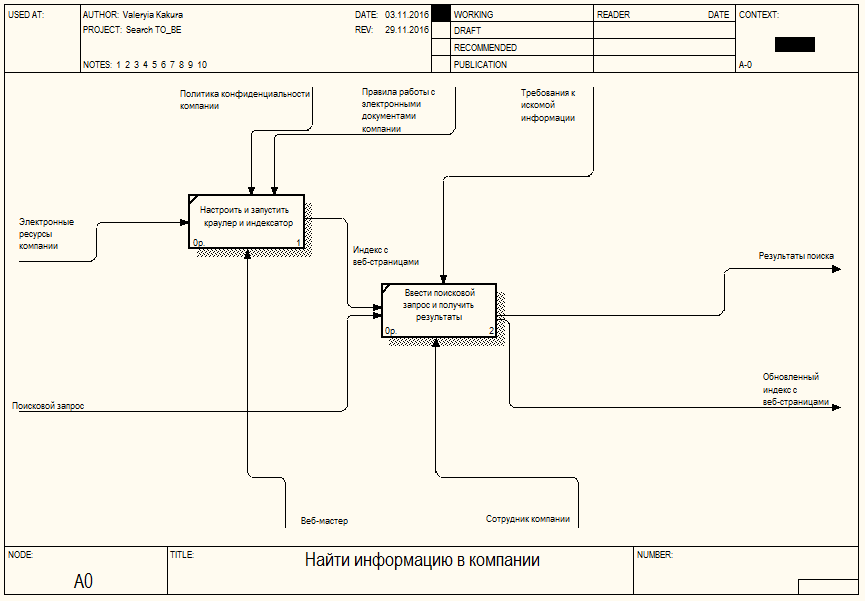


Рисунок 2.14 – Декомпозиция блока «Найти информацию в компании» в модели «TO-BE»

Вернемся к примеру с поиском заявления на отпуск. Последовательность действий сотрудника при использовании корпоративной поисковой системы выглядит как на рисунке 2.15.



Рисунок 2.15 – Последовательность действий сотрудника при поиске формы заявления с использованием корпоративной поисковой системы

По сравнению с использованием каталога веб страниц, поиск с использованием корпоративной поисковой системы имеет большие преимущества: сотруднику не нужно думать, в какой категории искать тот или иной документ. Достаточно ввести поисковой запрос и сразу же просмотреть полученные результаты.

Сравнивая модели «AS-IS» и «TO-BE» можно обратить внимание на то, что в проектируемой модели поиск информации сотрудником сможет занимать считанные секунды, время обновления информации в индексе будет минимальным. В такую модель легко можно добавить функцию удаления из индекса несуществующих страниц, в отличие от модели «AS-IS», в которой веб-мастеру приходится регулярно проверять существование веб-страниц, сохраненных в индексе. Модель «TO-BE» соответствует требованиям современных IT-компаний, предъявляемых к поиску информации.

Итак, был исследован процесса поиска информации в компании. На основе результатов исследования был проведен анализ существующей поисковой модели, а также предложена новая модель корпоративной информационно-поисковой системы. Новая модель позволит значительно сократить время, которое затрачивают сотрудники на поиск информации. Специалисты, веб-мастера, которые работают с поисковой системой, также получат удобный способ управления поисковыми роботами и веб-ресурсами компании, доступными для поиска.

1. **РАЗРАБОТКА КОРПОРАТИВНОЙ ИНФОРМАЦИОННО-ПОИСКОВОЙ СИСТЕМЫ**
   1. **Постановка задачи**

Для реализации корпоративной информационно-поисковой системы необходимо поставить и решить следующие задачи:

* изучить мировой опыт создания корпоративных поисковых систем;
* разработать базу данных будущей системы;
* на основе изученной информации разработать собственную модель поисковой системы;
* составить UML-модели разрабатываемой системы;
* разработать поискового робота для обработки веб-ресурсов;
* разработать удобный и понятный дизайн приложения для администратора для управления содержимым базы;
* разработать приложение для сотрудников, желающих воспользоваться поисковой системой.

Разрабатываемый автоматизированный модуль расчета трудовых ресурсов IT-проектов должен представлять собой систему со следующими уровнями:

* база данных;
* серверная часть;
* веб-паук, или робот;
* приложения для администратора;
* клиентское приложение.

Серверная часть должна реализовать REST Web-сервис, методы которого могли бы использовать любые клиенты. Сохранность данных должна быть обеспечена их хранением на сервере в базе данных.

В качестве базы данных была выбрана SAP HANA. Она поддерживает создание полнотекстовых индексов, что ускоряет поиск данных.

Работа с базой данных на сервере должна быть организована при помощи JDBC. Это позволит создавать запросы с синтаксисом, характерным базе данных HANA и упростить взаимодействие с базой данных.

Для эффективной работы серверной части использовать Spring – это свободно распространяемый фреймворк, который был создан с целью устранения сложности разработки корпоративных приложений. Данный фреймворк обладает весьма широкими возможностями. В своем устремлении на сложности, связанные с разработкой на языке Java, фреймворк Spring использует четыре ключевые стратегии:

* легковесность и ненасильственность благодаря применению простых Java-объектов (POJO);
* слабое связывание посредством внедрения зависимостей и ориентированности на интерфейсы;
* декларативное программирование через аспекты и общепринятые соглашения;
* уменьшение объема типового кода через аспекты и шаблоны.

Клиентское приложение должно быть реализовано в привычном пользователям виде. Должна присутствовать главная страница со строкой для поискового запроса и страница с результатами поиска.

Разрабатываемый программный продукт должен обеспечивать:

* возможности удобного добавления и хранения информации;
* возможности удобного просмотра и редактирования информации;
* просмотр сохраненных веб-страниц при помощи фильтров;
* расчет распределения значений содержимого.

К поисковой системе также предъявлены бизнес-требования:

* возможности подбора по ключевым словам;
* учет ограничения по числовым значениям и по дате, возможность уточнения запросов и объединение результатов различных запросов;
* обеспечение авто преобразования в формат HTML, предсказание запросов;
* использование технологий оценки качества и ранжирования результатов;
* кэширование страниц;
* формирование сложных логических поисковых запросов;
* поддержка полнотекстового поиска с учетом морфологии, возможность сегментации поискового индекса в зависимости от профессиональной роли;
* использование различных фильтров (например, ограничение областей поиска по отдельным языкам, типам файлов, Web-сайтам, метатегам);
* определение синонимов для принятых в компании сокращений и терминов;
* персонализация параметров поиска;
* обеспечение группировки результатов поиска по источникам информации.
  1. **Спецификация вариантов использования информационно-поисковой системы**

При проектировании корпоративной информационно-поисковой системы компании необходимо определить функциональное назначение программного продукта и разработать требования к его функциональным характеристикам. Рассмотрим основные группы пользователей системы и определим необходимые функциональные возможности системы.

Все пользователи информационно-поисковой системы могут быть условно разделены на две группы: администраторы и сотрудники компании.

Администратор базы данных — лицо, отвечающее за выработку требований к базе данных, её проектирование, реализацию, эффективное использование и сопровождение, включая управление учётными записями пользователей БД и защиту от несанкционированного доступа. Не менее важной функцией администратора БД является поддержка целостности базы данных [12].

Однако в случае разрабатываемой защищенной информационно-поисковой системы для IT компании следует среди администраторов в отдельную подгруппу поместить владельцев системы. Владелец системы, или владелец репозитория, создает и конфигурирует базу данных для хранения всей информации. Он также настраивает работу краулера и запускает его. В свою очередь обычные администраторы не имеют таких возможностей. Но и те и другие могут просматривать и настраивать конфигурации системы, права доступа сотрудников компании к искомым документам и ресурсам компании.

Поэтому для начала рассмотрим полный список возможностей администратора по работе с системой (рисунок 3.1):

1. Войти в систему.
2. Выйти из системы.
3. Изменить пароль.
4. Управлять репозиториями, что включает в себя просмотр и удаление репозитория в случае необходимости.
5. Управлять конфигурациями репозитория. К этой возможности относится не только просмотр существующих конфигураций и создание новых, но также и возможность отмены действия конфигурации системы.
6. Управлять настройками доступа к репозиторию, что включает создание правил доступа пользователей и групп пользователей к API поискового приложения, возможность запретить доступ пользователю и группе пользователей к репозиторию, изучение существующих настроек доступа.
7. Управлять моделями данных, что подразумевает способность администратора просмотреть модели хранилища, определить новые модели в хранилище, удалить или изменить модель, объединение моделей.
8. Управление содержимым модели данных является еще одной возможностью администратора, которая также включает добавление, создание, удаление, изменение содержимого модели. Немаловажной функцией является изучение распределения значений поля модели данных в хранилище.
9. Добавить администратора репозитория.
10. Просмотреть документацию по работе с поисковым приложением.
11. Просмотреть владельцев репозитория.

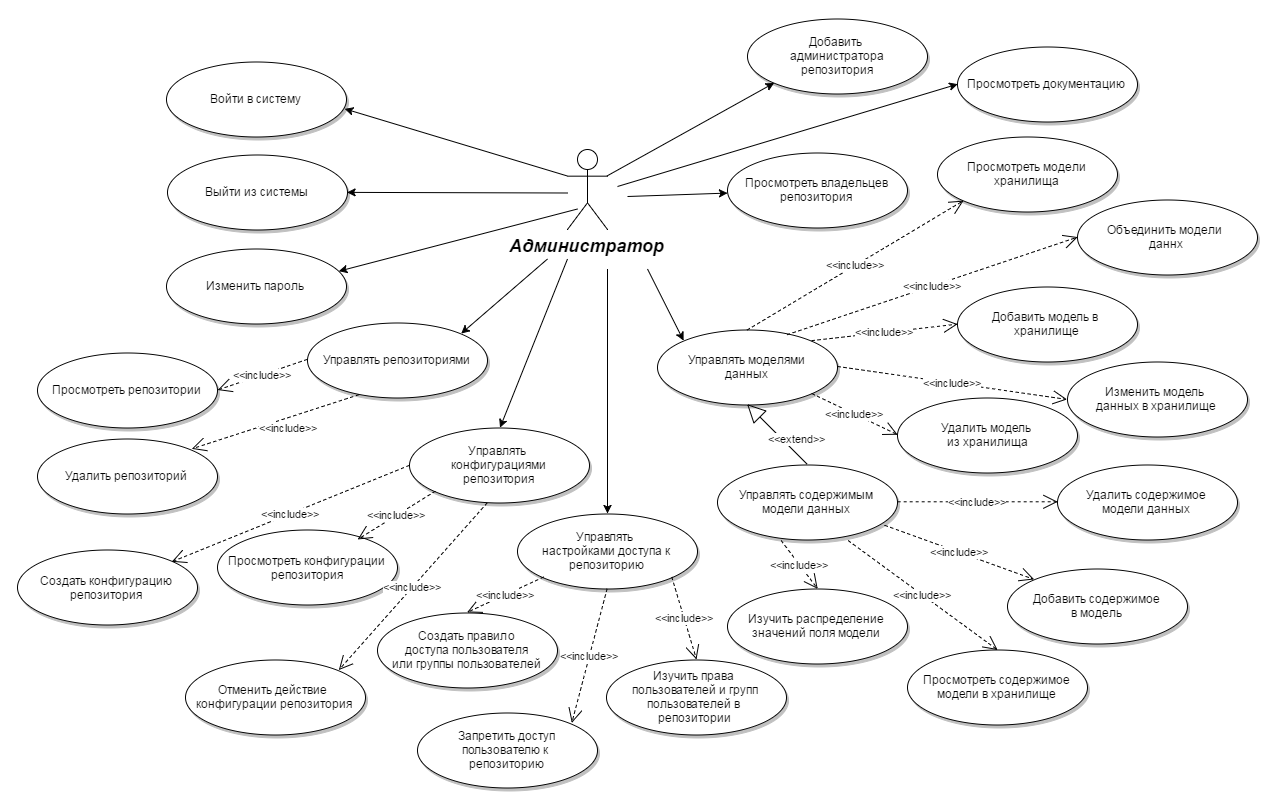


Рисунок 3.1 – Диаграмма вариантов использования информационно-поисковой системы администратором

Владелец репозитория имеет, как сказано выше, более широкие возможности, однако все фукнции администратора ему также доступны. Поэтому определим владельца базы данных еще одним актером на диаграмме вариантов использования, зададим связь наследования между администратором и владельцем хранилища таким образом, что владелец наследует все возможности администратора и имеет доступные только ему возможности. Диаграмма вариантов использования для владельца представлена на рисунке 3.6.

На рисунке 3.2 видно, что в возможности владельца входят:

* создание репозитория для хранения данных;
* управление владельцами репозитория;
* управление краулером;
* настройки поиска.

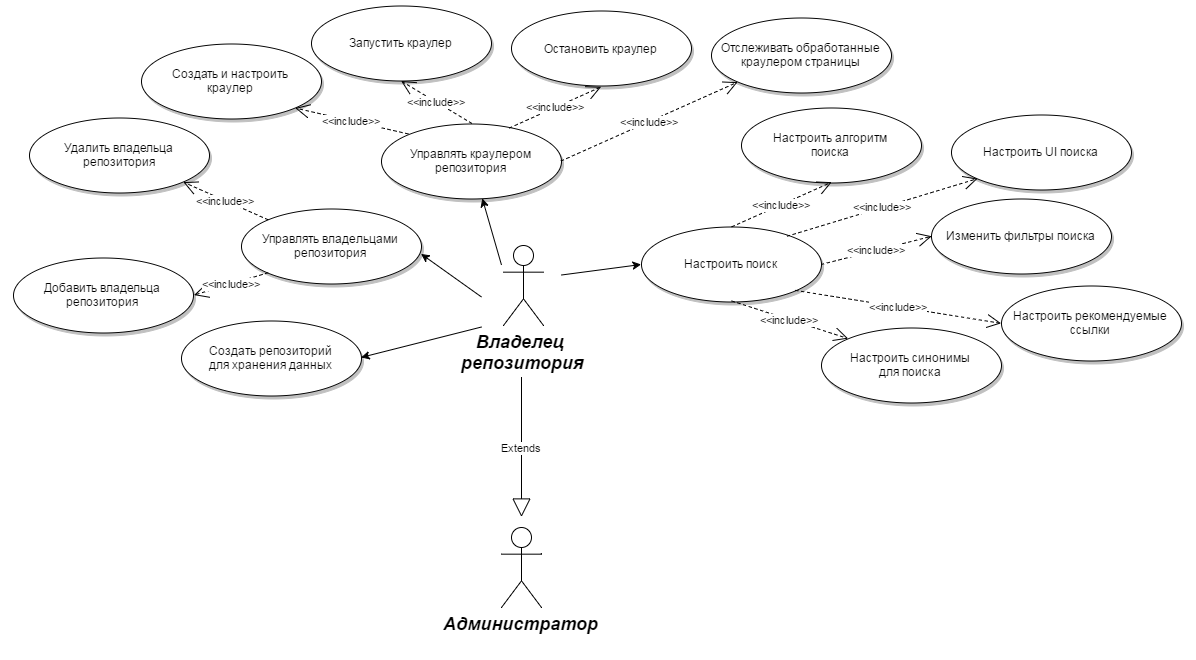


Рисунок 3.2 – Диаграмма вариантов использования информационно-поисковой системы для владельца

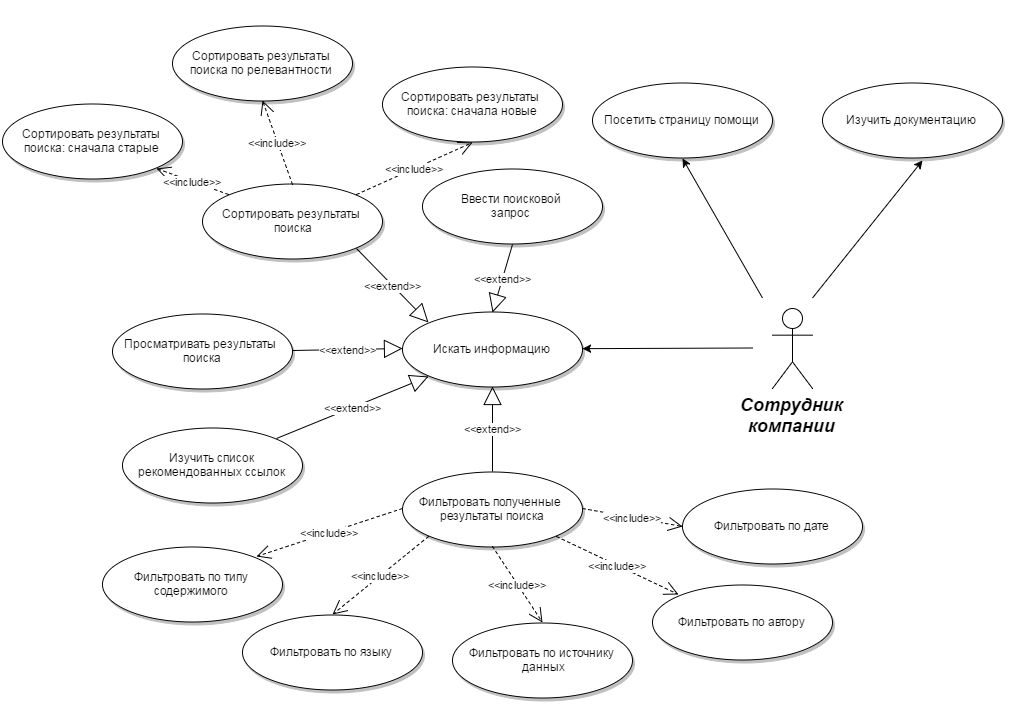


Рисунок 3.3 – Диаграмма вариантов использования информационно-поисковой системы для сотрудника компании

Сотрудники компании в данном случае являются конечными пользователями, желающими найти ресурсы. Безусловно, администратор и владелец могут также работать в той же самой компании и являться ее сотрудниками, но в данном случае, имеются в виду в первую очередь обычные служащие, работающие с веб-ресурсами компании. Они не имеют полномочий настраивать и редактировать систему поиска. Диаграмма вариантов использования для сотрудника представлена на рисунке 3.3.

Полная диаграмма вариантов использования информационно-поисковой системы представлена на рисунке А.1 в приложении А.

* 1. **Информационная модель поисковой системы IT-компании**

Чтобы разработать поисковую систему для IT-компании необходимо представить информационную модель этой системы. Для этого рассмотрим поисковую систему в виде совокупности объектов и связей между ними.

Информационная модель «Search Repository» представлена на рисунке 3.4. Под репозиторием понимается в данном случае база данных. Система спроектирована и разработана так, чтобы обеспечить безопасность хранения всей информации IT-компаний. Поэтому для каждой компании перед началом работы с поисковой системой будет создаваться свой репозиторий.

Начнем с описания сущности OWNERS. Она необходима для хранения информации о владельцах репозитория. Репозиторий создается веб-мастером компании и содержит всю информацию о нем – имя NAME, фамилию SURNAME, пароль PASSWORD.

Сущность CLIENTS представляет всех пользователей репозитория. Данная сущность имеет свойство CLIENT\_NAME для хранения имени и фамилии пользователя системы, а также свойство PASSWORD – пароль пользователя. Пользователю могут быть даны права администратора, что отображается наличием у него в поле IS\_ADMIN.

Сущность CLIENT\_CONFIG позволяет задать конфигурации работы пользователя с содержимым. Поэтому первым свойством является идентификатор пользователя CLIENT\_ID, определяющий связь с сущностью CLIENTS как многие к одному. Поле ALLOW\_READ определяет, есть ли у пользователя возможность читать контент, ALLOW\_WRITE определяет возможность записывать его. CONTENT\_ID является идентификатором контента, так как данная сущность связана с сущностью CONTENT отношением многие к одному.

Сущность USER\_GROUP необходима для хранения групп пользователей. Каждой группе можно задать имя GROUP\_NAME. Объединение пользователей в группы позволит задавать конфигурации и параметры доступа к контенту сразу для многих пользователей, а не для каждого в отдельности, как это делается в сущности CLIENT\_CONFIG.

Конфигурации для группы пользователей хранятся в сущности GROUP\_CONFIG, которая связана с сущностью USER\_GROUP отношением многие к одному. Конфигурация представлена полем ALLOW\_READ и полем CONTENT\_ID, которое является внешним ключом для связи с сущностью CONTENT.

Сущность AUTHORIZATIONS позволяет настроить доступ для групп пользователей ко всем доступным API. В ней хранятся правила доступа, которые состоят из имени правила NAME, ссылки на группу USER\_GROUP\_ID, для которой это правило определяется, паттерна правила PATTERN и описания DESCRIPTION. Имя, ссылка на группу и паттерн являются обязательными свойствами. Описание не является обязательным, но оно очень полезно при конфигурации системы различными администраторами. Идентификатор ID является автоматически генерируемым числом.

Кроме настроек для пользователей и групп пользователей в репозитории должны храниться конфигурации системы и общие настройки. Для данных целей введем сущность CONFIGURATIONS. Сущность позволяет описать элементы конфигурации, которые будут универсальной системой хранения внутренних настроек. Каждый элемент конфигурации должен иметь категорию, ключ и значение. Поэтому данная сущность содержит соответствующие поля – CATEGORY, KEY, VALUE. Категория представляет собой строку, задающую описание настройке. Ключ является уникальной строкой для идентификации настройки, значение хранит сами данные конфигурации.

Модели являются основной средой для хранения данных. Для хранения всего содержимого и описания веб-страниц, обработанного краулером, будет использоваться сущность по умолчанию, которая называется CONTENT. Идентификатор пользователя, инициализировавшего сохранение контента (либо запустив краулер, либо используя API), будет сохранен в сущности в свойстве CLIENT\_ID, таким образом связывая сущность содержимого с сущностью пользователей отношением многие к одному. Свойства CREATED\_TIMESTAMP, MODIFICATED\_TIMESTAMP, UPDATED\_ TIMESTAMP необходимы для хранения соответственно времени сохранения, изменения, обновления контента. Идентификатор CLUSTER\_ID хранит информацию об идентификаторе кластера, на котором был обработан контент. Идентификаторы SOLUTION\_ID, CONTENT\_TYPE\_ID и LANGUAGE\_ID служат связью с сущностями SOLUTIONS, CONTENT\_TYPE и LANGUAGE соответственно. Фактор FACTOR представляет собой числовое свойство типа decimal для хранения фактора поиска и может иметь значение от 1 до 2 включительно. Свойство LOCATION определяет местоположение. Свойство USERURL определяет URL, заданный краулеру для обработки страницы или используемый самим пользователем, для сохранения содержимого и описания веб-страницы в сущность контента. Идентификатор CITY\_ID служит для связи с сущностью CITIES. Поле TITLE хранит заголовок обрабатываемой страницы. COMPANY\_CODE\_ID задает связь многие к одному с таблицей COMPANY\_CODE. Свойство FILE\_TYPE хранит типа обрабатываемого файла, является необязательным полем.

Сущность CLUSTERS необходима для хранения информации о кластере, на котором был обработан контент, содержит идентификатор ID и значение CLUSTER\_VALUE.

Так содержимое может быть разного типа, введем сущность CONTENT\_TYPES для хранения названий типов контента. Для этого определим в сущности поле CONTENT\_TYPE и идентификатор ID.

Сущность LANGUAGES необходима для определения и сохранения возможных вариантов языков текста. Язык будет храниться в свойстве LANGUAGE.

Возможна ситуация, когда база будет использоваться не одной, а группой компаний. Чтобы идентифицировать, к какой компании относится содержимое, определим таблицу COMPANY\_CODES, где и будем хранить код компании в поле CODE.

Кроме того статьи и документы компании могут иметь авторов. Для хранения авторов введем сущность AUTHORS с полем AUTHOR и идентификатором ID. Отношение между сущностями CONTENT и AUTHORS может быть определено как многие ко многим. Поэтому для их связи будет использована еще одна таблица CONTENT\_AUTHORS, где будут поставлены в соответствие идентификаторы содержимого и авторов.

Важной частью работы поисковой системы являются ключевые слова содержимого. Они будут храниться в таблице KEYWORDS. Где содержимому по идентификатору CONTENT\_ID будут ставиться в соответствие ключевые слова в поле KEYWORDS.

Поисковая система должна выдавать релевантные результаты, а это значит, что содержимое, найденное по запросу, должно не только соответствовать запросу, но и также быть отсортировано в зависимости от предпочтений пользователя и его местоположения. Для хранения сферы и возможных интересов пользователя будут использоваться сущности CLIENT\_INTERESTS, INDUSTRIES. Для определения возможных интересов пользователя будут использоваться свойства INTERESTS, JOBFUNCTION, DEPARTMENT, INDUSTRY. А для сортировки результатов в зависимости от расположения будут использоваться сущности CITIES, REGIONS, COUNTRIES, имеющие поля CITY, REGION и COUNTY соответственно для хранения города, региона, страны.

Условия поиска каждого пользователя будут сохраняться в таблице SEARCH\_TERMS, куда кроме идентификатора пользователя, даты поиска и самого условия будет также записываться информация о количестве и типе ответов.

Содержимое также может иметь теги, по которым его будет проще найти. Они могут выставляться вручную авторами содержимого при создании страниц, статей, документов. Все возможные теги будут сохранены в сущности TAGS, которая связана с сущностью CONTENT отношением многие ко многим. Поэтому для этой связи будет использована еще одна таблица CONTENT\_TAGS.

Одной из важнейших частей информационно-поисковой системы является работа поискового робота. При этом необходимо предоставить возможность не только определять и сохранять параметры его работы, но также вести журналирование. Это может пригодиться при сбоях в работе и поможет восстановить данные и найти причину сбоя. Для работы с краулером будут использованы модели CRAWLER, CRAWLER\_LOG, WORK, MIGRATION\_ERROR\_LOG, MIGRATION\_LOG.

Сущности CRAWLER\_LOG и MIGRATION\_LOG необходимы для ведения журналирования работы краулера и процесса миграции информации в хранилище данных. А сущность MIGRATION\_ERROR\_LOG необходима для хранения ошибок миграции.

Сущность CRAWLER описывает параметры самого краулера. Поле OWNER\_ID хранит идентификатор владельца, который осуществлял настройки. Поле CRAWLER\_URLS представляет собой свойство для хранения URLS, с которых поисковой робот начнет обработку веб-страниц. Также введены свойства для иных конфигураций работы краулера:

* необходимое имя пользователя USER\_NAME\_AUTHENTICATION и пароль PASSWORD\_AUTHENTICATION, если для веб-сайта необходима аутентификация;
* порт прокси-сервера PROXY\_PORT и прокси-хост PROXY\_HOST;
* максимальный размер загрузки MAX\_DOWNLOAD\_SIZE;
* лист типов содержимого для обработки CONTENT\_TYPE\_LIST, при необходимости обрабатывать, например, только электронные документы в формате PDF;
* максимальная глубина разбора страниц MAX\_DEPTH\_OF\_CRAWLING;
* максимальное количество ссылок, по которым перейдет поисковой робот в процессе обработки MAX\_OUTGOING\_LINKS;
* свойство для хранения метаданных METADATA.



Рисунок 3.4 – Информационная модель системы

* 1. **Модели представления информационно-поисковой системы**

При проектировании корпоративной информационно-поисковой системы был проведен обзор пользователей данной системы, а с помощью диаграммы IDEF0 был представлен процесс поиска. Для успешной разработки системы необходимо определить архитектуру приложения и выбрать компоненты для реализации проекта.

Узнаем, что происходит, когда сотрудник компании желает найти информацию и вводит поисковой запрос. Диаграмма последовательностей представлена на рисунке 3.5. Приложение на диаграмме рассмотрено как система взаимодействующих между собой объектов во времени. Поэтому сотрудник компании – первый объект, который начинает цепочку взаимодействий. Предположим, что он успешно прошел аутентификацию. Сотрудник заходит на главную страницу поисковика и вводит запрос в строке для поиска. Ему необходимо получить список результатов по данному поисковому запросу.



Рисунок 3.5 – Диаграмма последовательностей взаимодействия объектов информационно-поисковой системы

Главная страница передает поисковой запрос логике поисковика, которая также получает информацию о местоположении пользователя, чтобы в дальнейшем у поискового приложения была возможность учитывать этот фактор при сортировке результатов для отображения. Сервисы поисковика также передают поисковому приложению информацию пользователя, чтобы при возможности использовать существующую информацию о данном сотруднике для составления релевантных результатов. Поисковое приложение отправляет запрос Web API, который в свою очередь извлекает данные по запросу из базы.

Данные из базы возвращаются через API поисковому приложению в формате json. После этого поисковое приложение обрабатывает полученные данные и формирует релевантный для данного сотрудника результат. Далее формируется страница результатов, которая кроме непосредственно самих результатов содержит также фильтры, сортировки, рекомендуемые ссылки. Страница отображается сотруднику. Далее он может изучать результаты поиска.

С помощью диаграммы последовательностей был показан жизненный цикл поискового приложения и взаимодействие объектов системы. Данный анализ позволяет определить необходимые компоненты системы и физическое размещение приложений и серверов.

Рассмотрим структурные логические компоненты системы. Из диаграммы последовательностей видно, что пользователю для работы с поисковой системой необходимо поисковик, который позволит ввести поисковой запрос и отобразить результаты поиска. Таким компонентом системы будет приложение srch-searchui.

Далее необходим компонент, способный обрабатывать результаты поиска и с помощью алгоритмов сортировать их по релевантности для данного пользователя. Этот компонент представлен приложением srch-search.

Запросы к базе с реализацией всех CRUD операций будут отправляться приложением srch-web-api.

В качестве базы данных выбрана платформа SAP HANA.

Выше описана лишь часть работы информационно-поисковой системы при учете того, что в базе уже присутствуют данные. Однако база наполняется поисковым роботом, который является отдельным компонентом системы. Приложение srch-crawler будет является краулером системы.

Как говорилось ранее, владелец хранилища сначала должен настроить поисковых роботов. Для данных целей определим отдельный модуль srch-migration, который кроме конфигурации поисковых роботов, позволит также запускать их вручную, либо настраивать расписание запуска. Это приложение будет следить за работой краулера.

После того как краулер начал обрабатывать страницы и заполнять базу, администраторы базы могут проверять и редактировать ее содержимое, создавать группы сотрудников компании, которым содержимое будет недоступно, или наоборот определять права сотрудников на поиск и просмотр данного содержимого. Администратор для этих целей будет использовать отдельный компонент srch-admin-ui.

Невидимым извне, но также очень важным, является компонент srch-common, в котором содержатся основные классы сущностей и логики приложений. Он совместно используется модулями srch-search, srch-migration.

Структурные компоненты представлены на рисунке 3.6 с помощью диаграммы компонентов.

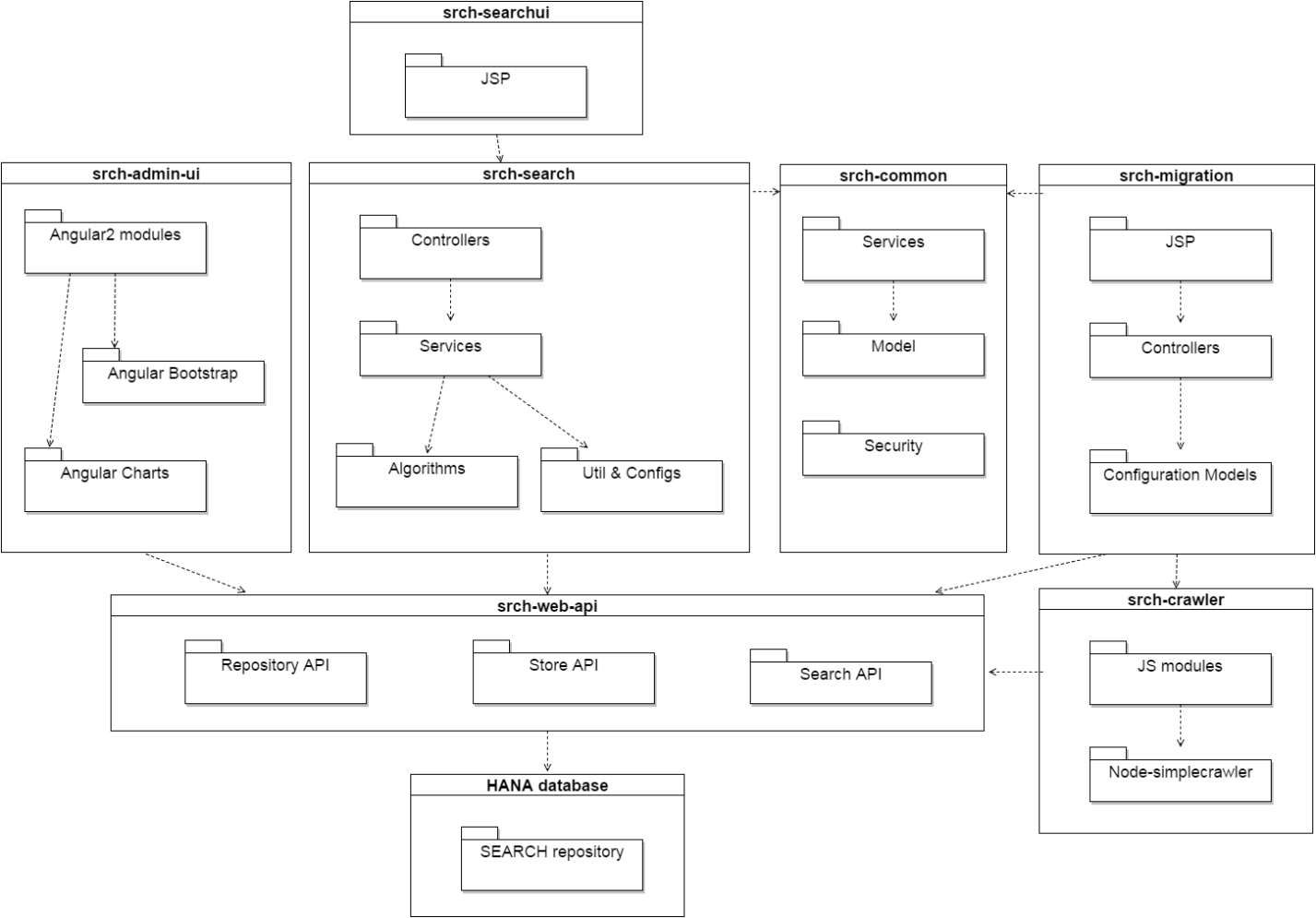


Рисунок 3.6 – Диаграмма компонентов информационно-поисковой системы

Перейдем к визуализации тех элементов системы, которые существуют во время выполнения. Будут использованы облачные технологии. Для каждого элемента необходима отдельная сущность в облаке. Получим следующие узлы:

* узел с Web API и кластерами базы данных Hana;
* узел с веб-сервером nginx для приложения Admin UI;
* узел с веб-сервером Tomcat с поисковиком Search для конечных пользователей;
* узел с JVM для приложения Search;
* узел c JVM для работы владельцев базы и настройки краулеров через приложение Search Migration;
* одна или более облачных сущностей для работы приложений Crawler.

Диаграмма развертывания представлена на рисунке 3.7.



Рисунок 3.7 – Диаграмма развертывания информационно-поисковой системы

* 1. **Описание алгоритма работы поисковой системы при сохранении веб-страниц**

В Приложении Б на рисунке Б.1 представлена блок-схема обобщенного алгоритма работы поисковой системы при сохранении страницы, найденной поисковым роботом.

Алгоритм начинается со считывания конфигураций поискового робота. Далее проверяется правильность конфигураций, отсутствие в них смысловых ошибок и опечаток. Если конфигурации не верны, поисковой робот не должен начинать свою работу, вместо этого будет выведено сообщение об ошибках в конфигурации поискового робота. Если конфигурации верны, начинается обработка страницы. Для начала страница должна быть считана по указанному URL. При этом, если указанный URL не доступен, также должно быть выведено сообщение об ошибке. Иначе происходит загрузка содержимого страницы. Также поисковая система должна проверить, была ли ранее уже обработана эта страница и есть ли данные о странице в таблице с содержимым. Если такая страница уже была обработана, то необходимо обновить метаинформацию о ней.

Обработка не сохраненной ранее страницы начинается со считывания конфигураций мапинга для нее. Далее по всем элементам из мапинга происходит поиск соответствия на странице. В мапинге, например, указано, что со страницы необходимо извлечь название из тэга «title» и поместить его в поле TITLE таблицы CONTENT в базе данных. Аналогичные мапинги ищутся по всей странице для определения языка, даты создания и обновления страницы, содержимого страницы. При этом возможна ситуация, когда конфигурации соответствий не найдены на странице. В таком случае используется статическая заданная фраза, если она указана. Например, если не найден язык на странице, то в статическом мапинге можно указать, что по умолчанию будет сохранен английский язык. После обработки всех конфигураций мапинга данные сохраняются и обработка страницы заканчивается.

В результате, проектирования автоматизированной поисковой системы дипломного проекта были разработаны модели системы и алгоритмы программных модулей, реализующих бизнес-логику и веб-интерфейс приложения.

* 1. **Руководство по развертыванию**

Разработанная система поставляется в формате нескольких артефактов:

* srch-web-api;
* srch-search;
* srch-searchui;
* srch-admin-ui;
* srch-migration.

Итак, приложения готовы для развертывания на серверах. Сначала необходимо убедиться, что у нас есть все необходимое для развертывания программное обеспечение. На серверном компьютере должны быть установлены JDK 1.7 версии или выше, Tomcat 7 или выше. Чтобы убедиться, что все установлено правильно, зайдите в Свойства компьютера – Дополнительные параметры системы – Переменные среды. В системных переменных обязательно должны присутствовать CATALINA\_HOME, которая указывает на папку с установленным Tomcat, например C:\Program Files\Tomcat, и JAVA\_HOME, значение которой указывает на папку с установленной JGK, например, C:\Program Files\Java\jdk1.7.0\_79. Приложения srch-web-api, srch-search, srch-migration поставляются в формате архива war. Первый из списка – srch-web-api – должен быть запущен первым. На трех различных серверах достаточно лишь поместить файлы war в папку webapps Tomcat-a, в моем случае это C:\Program Files\Tomcat\webapps. В случае приложения srch-web-api необходимо предварительно установить сервер базы данных HANA. Сама база должна быть создана владельцем при настройке поисковой системы через приложение. Теперь запускаем Tomcat. Для этого заходим в папку C:\Program Files\Tomcat\bin и запускаем файл startup.bat. Приложения развернуты.

Для запуска приложения для администратора srch-admin-ui необходимо настроить HTTP-сервер. На компьютере с установленным веб-сервером nginx необходимо создать каталог (папку) и поместить туда файл srch-admin-ui.zip. Далее распакуем архив в данную папку. Зайдем в конфигурационный файл веб-сервера, изменим каталог для запуска. Выполним команду «nginx –s reload» для перезагрузки конфигурации веб-сервера. Теперь приложение доступно.

* 1. **Результаты тестирования поисковой системы IT-компании**
     1. Результаты тестирования для владельца информационно-поисковой системы. Рассмотрим работу с приложением для владельца. Начнем с описания настройки приложения, которая выполняется лишь один раз при вводе в эксплуатацию всей информационно-поисковой системы на предприятии. А после рассмотрим, какие возможности конфигурирования системы присутствуют у владельца.

При запуске приложения владелец должен перейти в браузере в панель администрирования (рисунок 3.8).

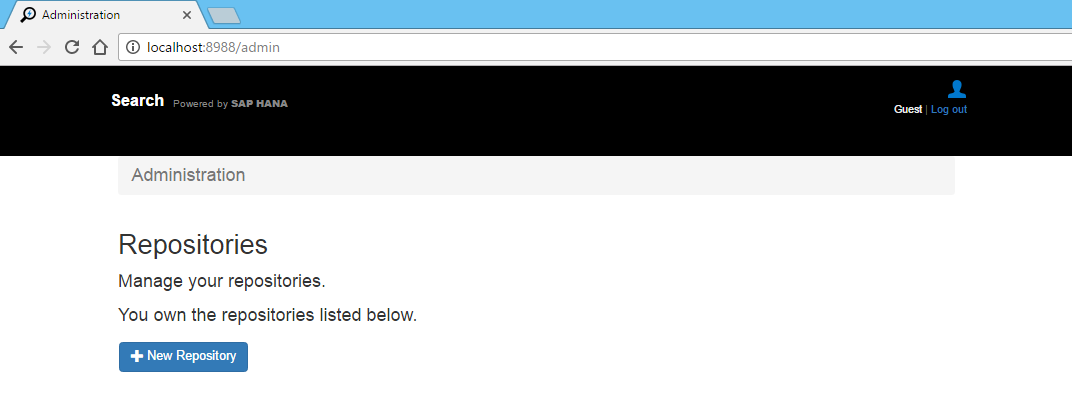


Рисунок 3.8 – Главная страница панели владельца репозитория

Для начала работы с поисковой системой необходимо создать репозиторий, в который краулером будет сохранена информация, по которой будет вестись поиск. Репозиторий представляет собой базу данных с таблицами. На странице администрирования нужно нажать «New Repository» для перехода на страницу его создания, как на рисунке 3.9.

На странице создания необходимо ввести имя для нового репозитория. Удобнее будет использовать название компании или принятое сокращение компании. После нажатия кнопки «Submit» вы будете перенаправлены на главную страницу (рисунок 3.10). Теперь на ней будет отображаться созданная вами база.

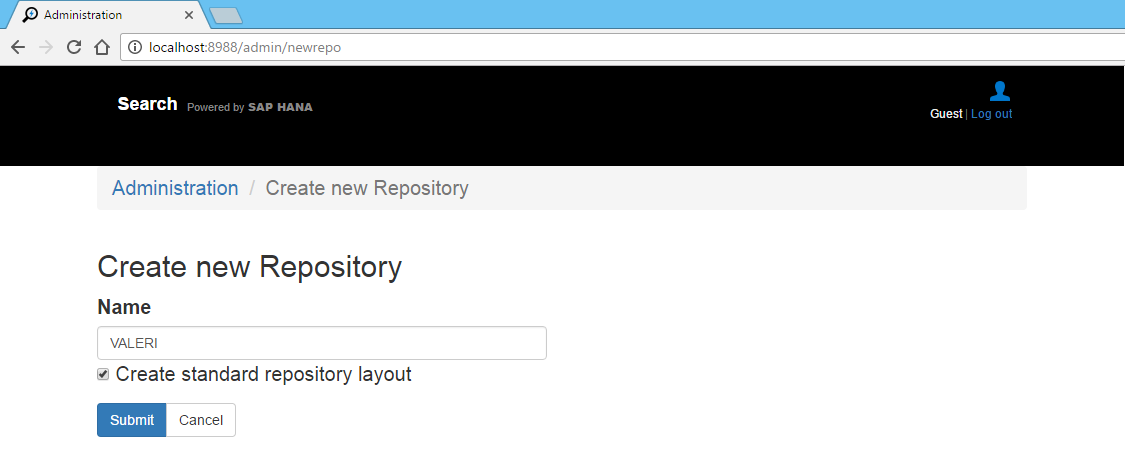


Рисунок 3.9 – Создание нового репозитория

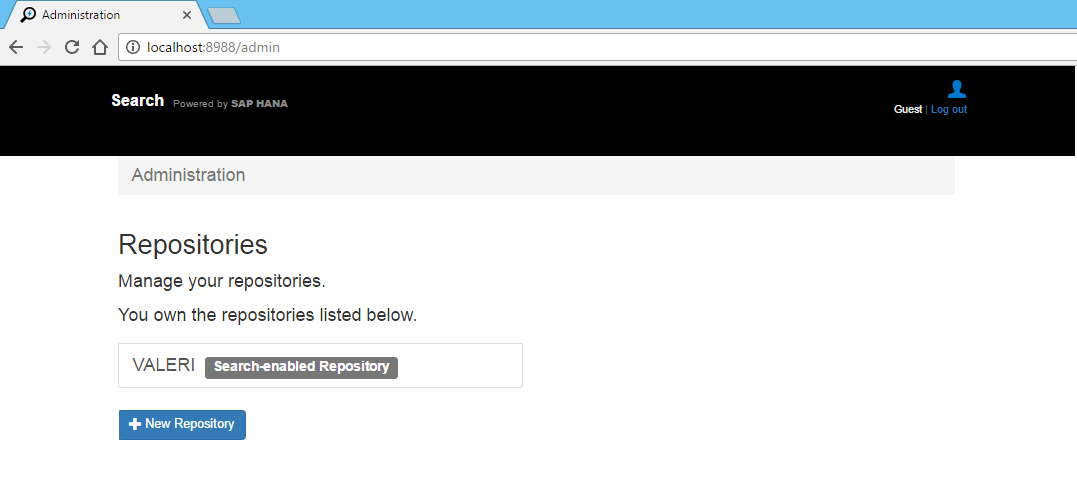


Рисунок 3.10 – Главная страница с созданной базой «VALERI»

При создании базы данных приложение автоматически создает таблицы в соответствии со схемой данных по умолчанию. Чтобы посмотреть возможности конфигурирования и настройки репозитория VALERI, необходимо нажать на него. Администратор увидит страницу управления как на рисунке 3.11.

Перейдем на страницу управления структурой базы. Для этого кликнем по меню «Manage Repository». Увидим страницу (рисунок 3.12), где рядом с названием репозитория есть различные кнопки для конфигурации. Для перехода на страницу управления администраторами базы кликнем по иконке пользователя. Владелец увидит страницу (рисунок 3.13), где отображены владельцы базы, а также есть возможность добавить новых администраторов.

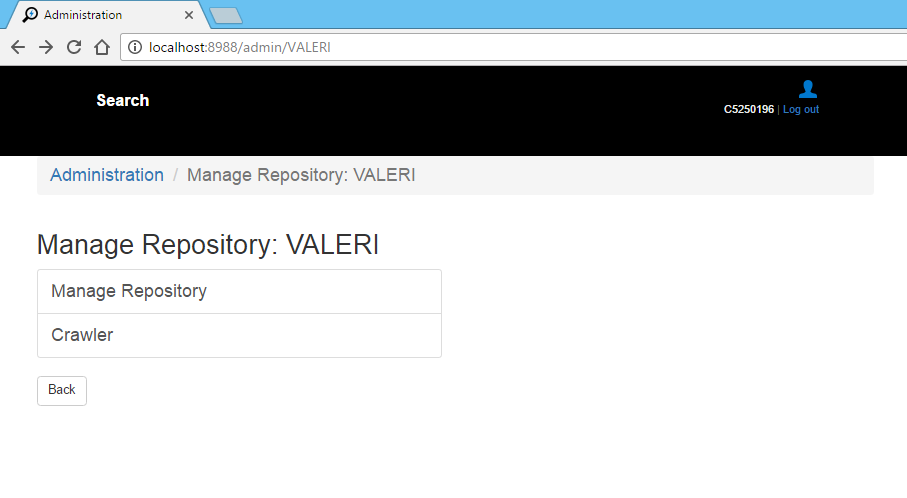


Рисунок 3.11 – Страница управления репозиторием «VALERI»

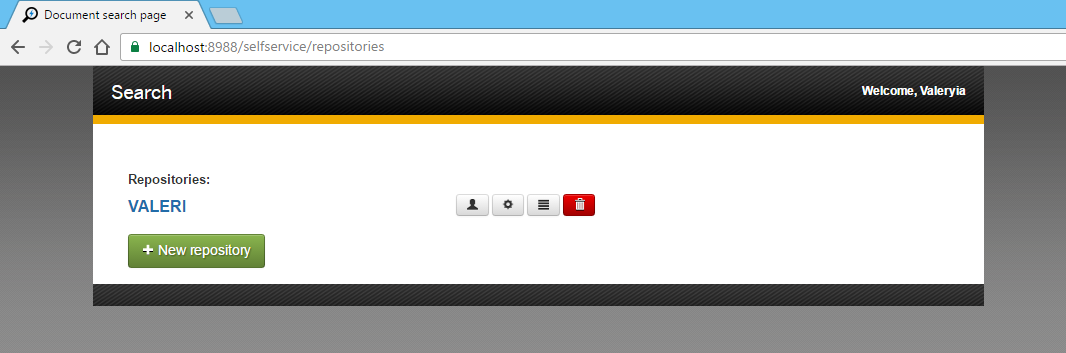


Рисунок 3.12 – Страница конфигурации репозитория

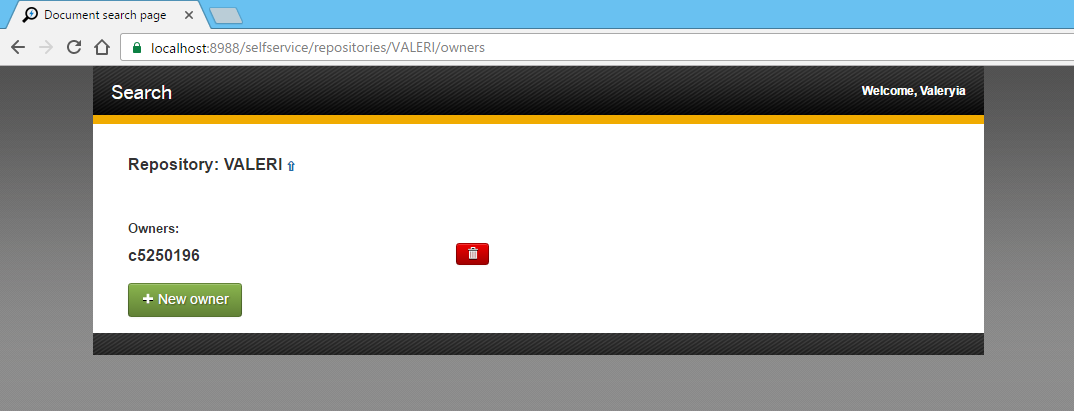


Рисунок 3.13 – Страница настройки списка администраторов базы

Для настройки пользователей репозитория необходимо вернуться на страницу конфигурации и щелкнуть по значку шестеренки. Так как на данный момент пользователи не были добавлены, можно сразу перейти на страницу создания нового пользователя (рисунок 3.14).

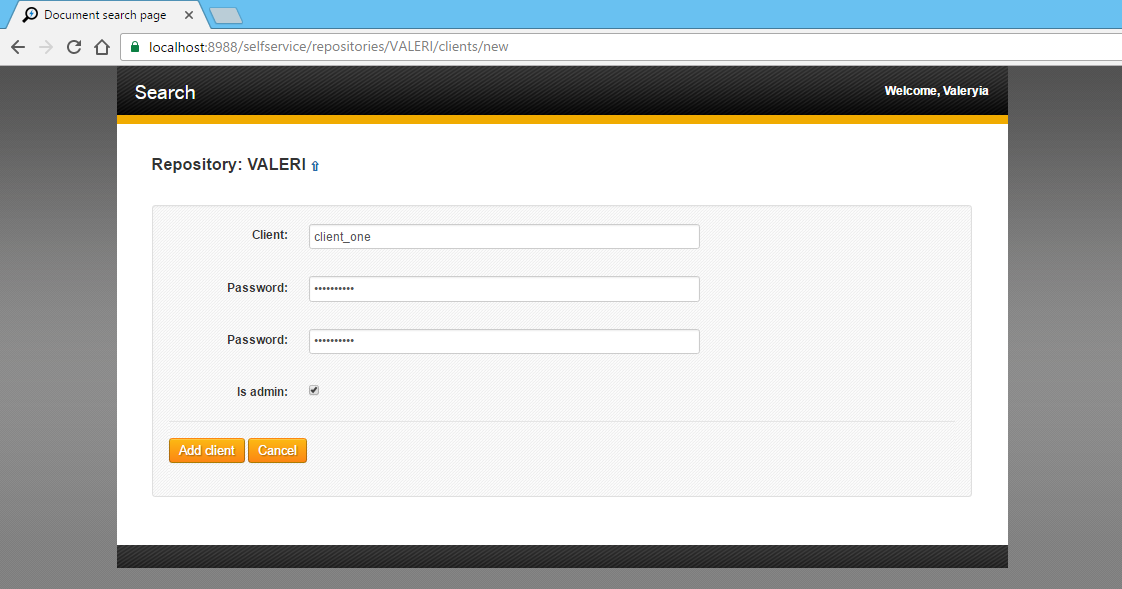


Рисунок 3.14 – Страница создания нового пользователя

Создадим для примера несколько любых пользователей и перейдем на страницу вернемся к странице настройке списка пользователей. Рисунок 3.15 показывает, как теперь будет выглядеть данная страница.

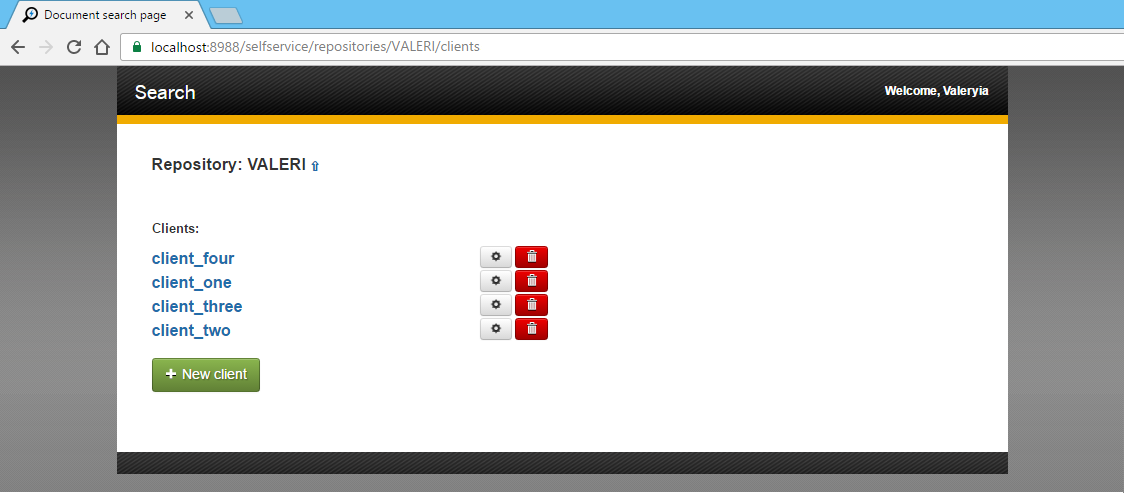


Рисунок 3.15 – Список пользователей репозитория

На странице со списком пользователя есть возможность удалить пользователя, нажав на красную кнопку со знаком корзины. Чтобы настроить конфигурации пользовательского доступа, необходимо кликнуть по кнопке с шестеренкой. Будет отображена страница, где будет указано, что у данного пользователя пока нет конфигураций (рисунок 3.16). Нажмем на кнопку «New type configuration» для перехода на страницу создания новой конфигурации для выбранного пользователя (рисунок 3.17).

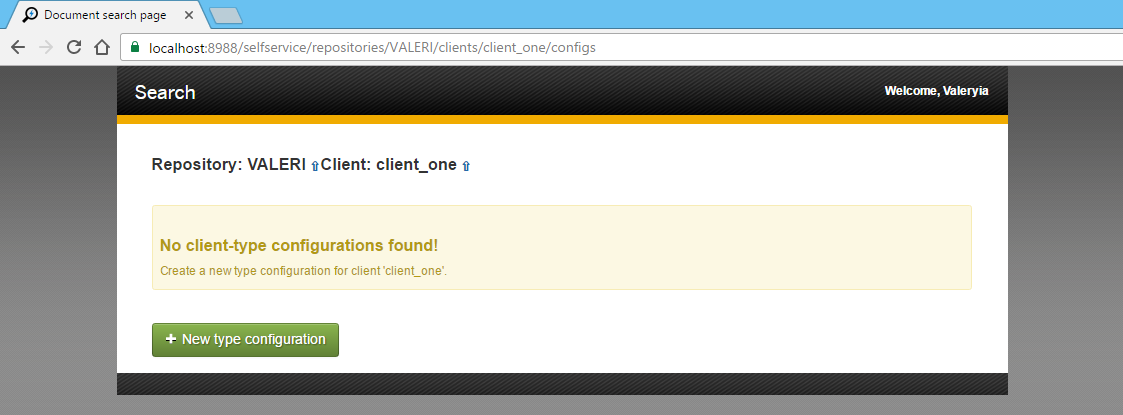


Рисунок 3.16 – Пустой список конфигураций пользователя client\_one

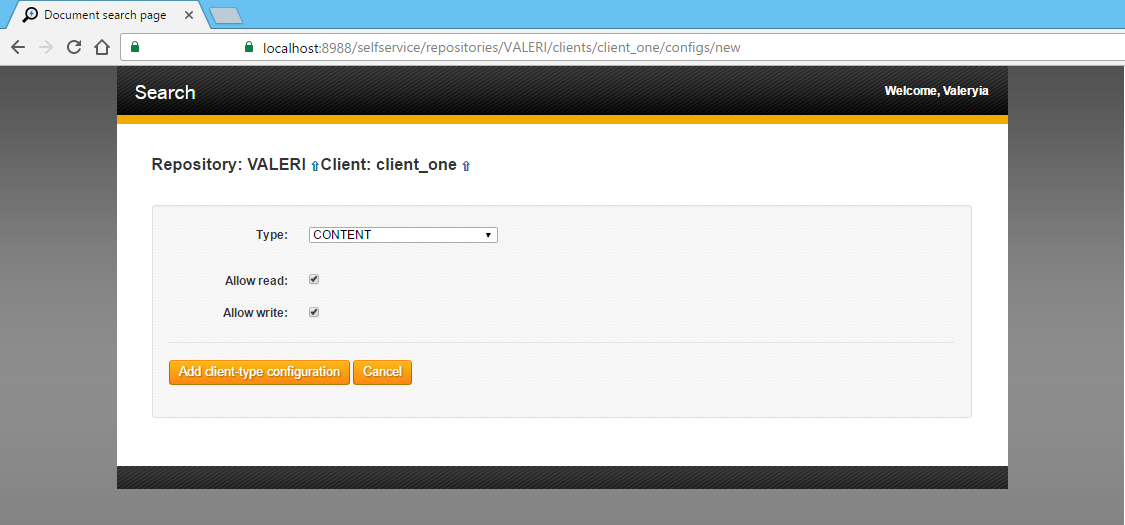


Рисунок 3.17 – Страница создания новой конфигурации для пользователя client\_one

Для типа из выпадающего списка выберем таблицу CONTENT, отметим галочками «Allow read» и «Allow write». После нажатия кнопки «Add client type configuration» администратор будет направлен на страницу конфигураций пользователя, где он может увидеть созданную для типа CONTENT конфигурацию, рисунок 3.18.

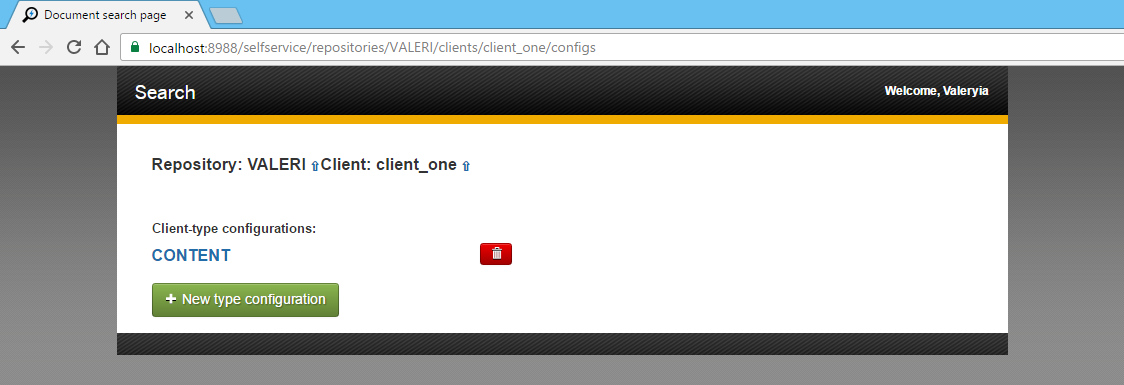


Рисунок 3.18 – Страница с конфигурацией пользователя client\_one

Вернемся на страницу управления репоиторием (рисунок 3.12), нажав на стрелочку возле названия репозитория. Кроме управления владельцами и пользователями базы есть возможность изменить структуру базы. Кликнем по иконке списка и перейдем таким образом на страницу конфигурации типов (рисунок 3.19).

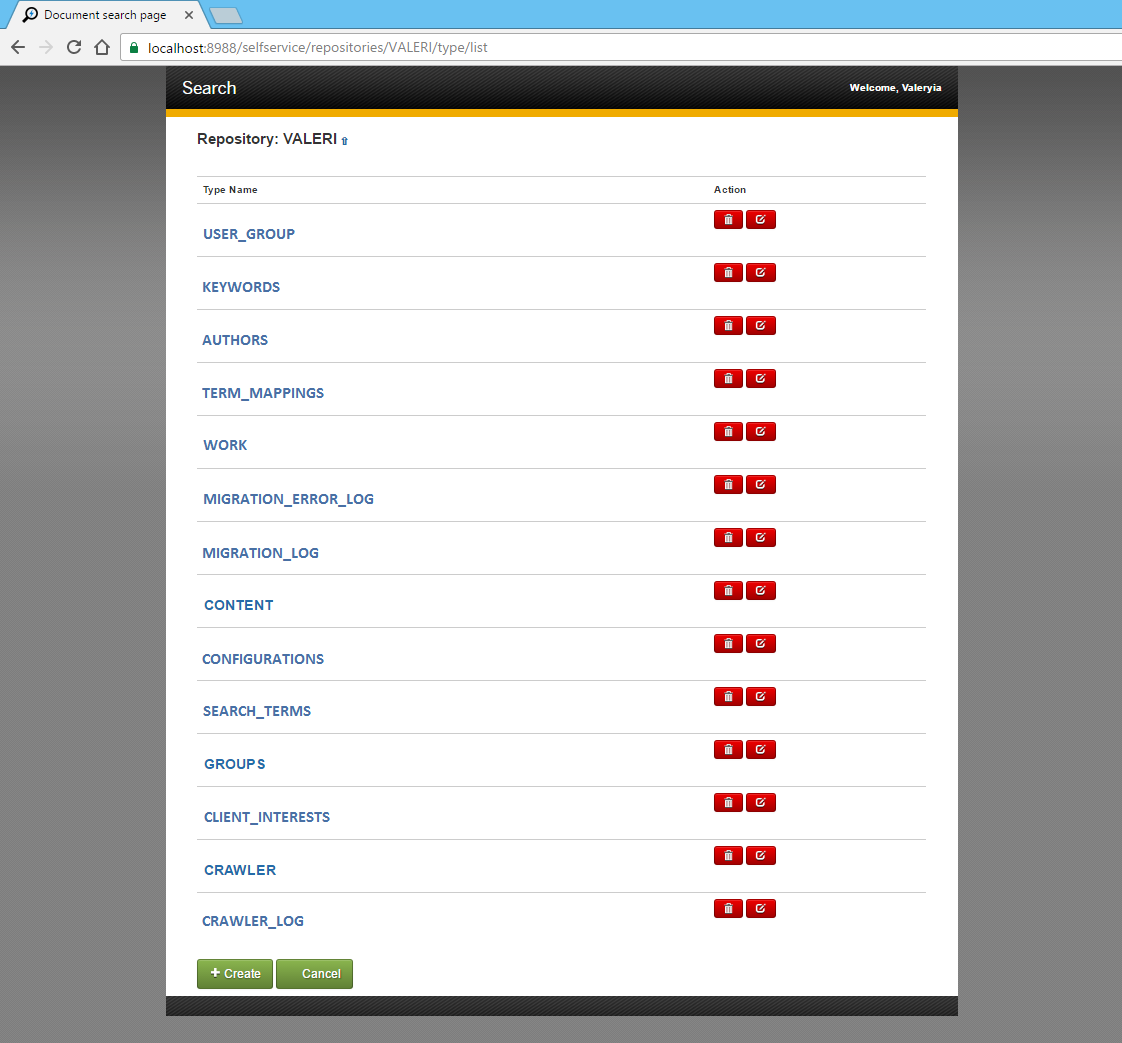


Рисунок 3.19 – Список типов в репозитории VALERI

Типы представлены таблицами в базе. У каждого типа есть список полей, тип и параметры которых могут быть редактированы. Для этого необходимо щелкнуть по значку редактирования рядом с названием типа. На странице редактирования можно также добавить новые поля, изменить их тип или значение по умолчанию. Можно создать свой новый тип. Форма создания типа имеет тот же вид, что и форма редактирования. Для этого кликнем по кнопке «Create» внизу страницы со списком типов. Страница создания нового типа представлена на рисунке 3.20. Заполним название типа как PRIVATE, например. Добавим несколько полей. Создание нового пользовательского типа позволит хранить не все данные в одной таблице CONTENT, как задано по умолчанию. В новую таблицу PRIVATE могут быть позже сохранены данные, доступ к которым удобно настроить лишь для определенной группы пользователей.

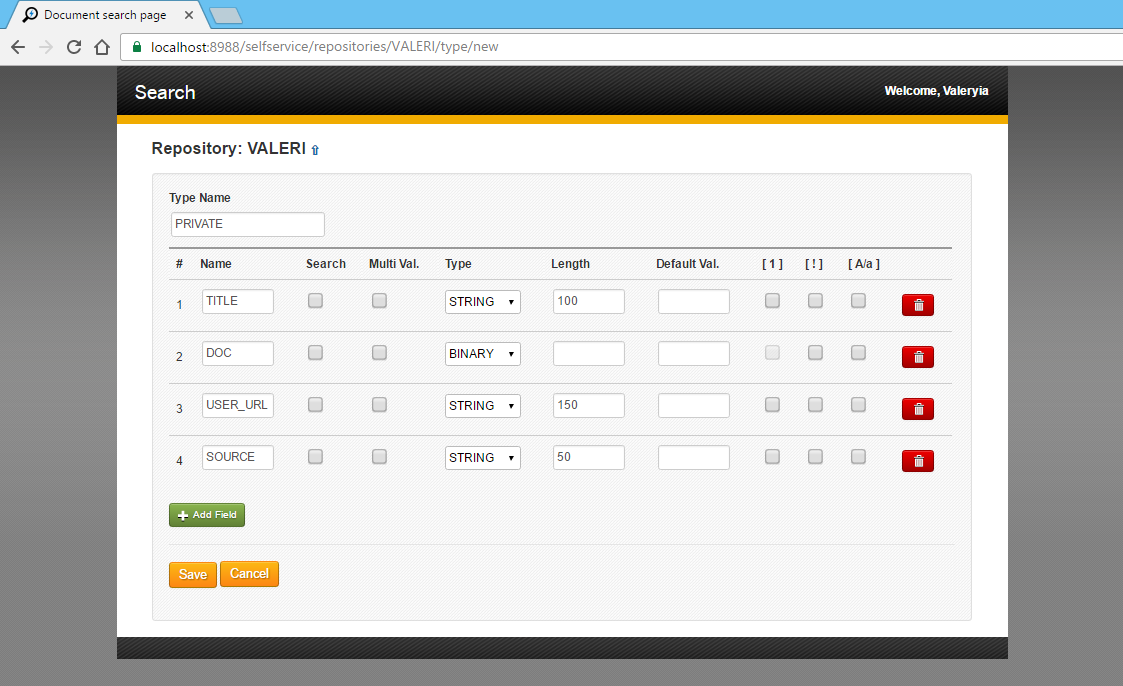


Рисунок 3.20 – Страница создания нового типа

Настроив типы и рассмотрев основные конфигурации базы можно перейти к запуску краулера. Для этого на странице меню администратора необходимо выбрать пункт «Crawler». На отобразившейся странице на рисунке 3.21 видно, что пока не создана ни одна задача для краулера. Поэтому нажмем кнопку «Create item» для перехода на страницу создания первой задачи краулера (рисунок 3.22).

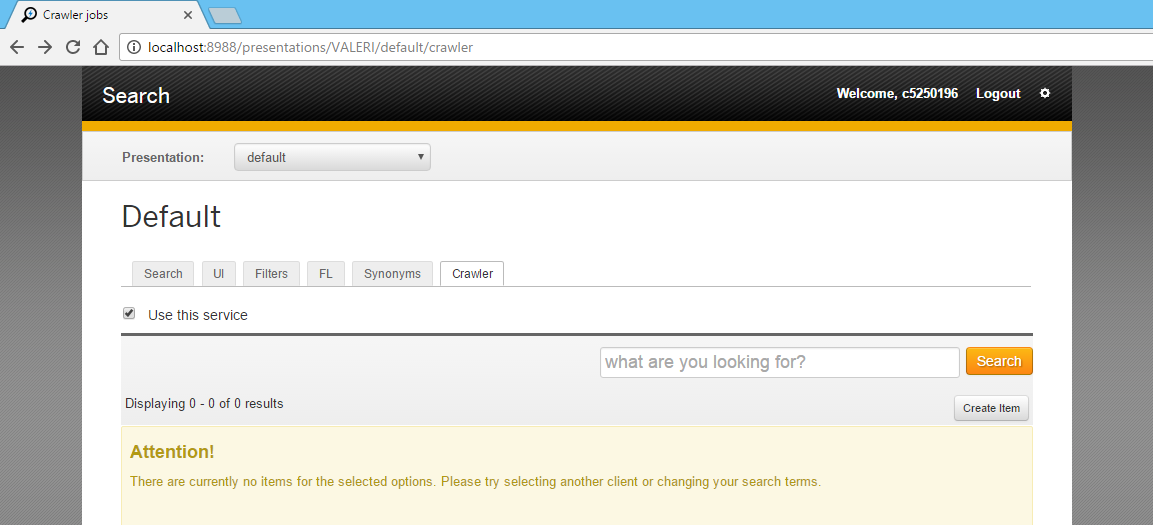


Рисунок 3.21 – Пустой список задач краулера

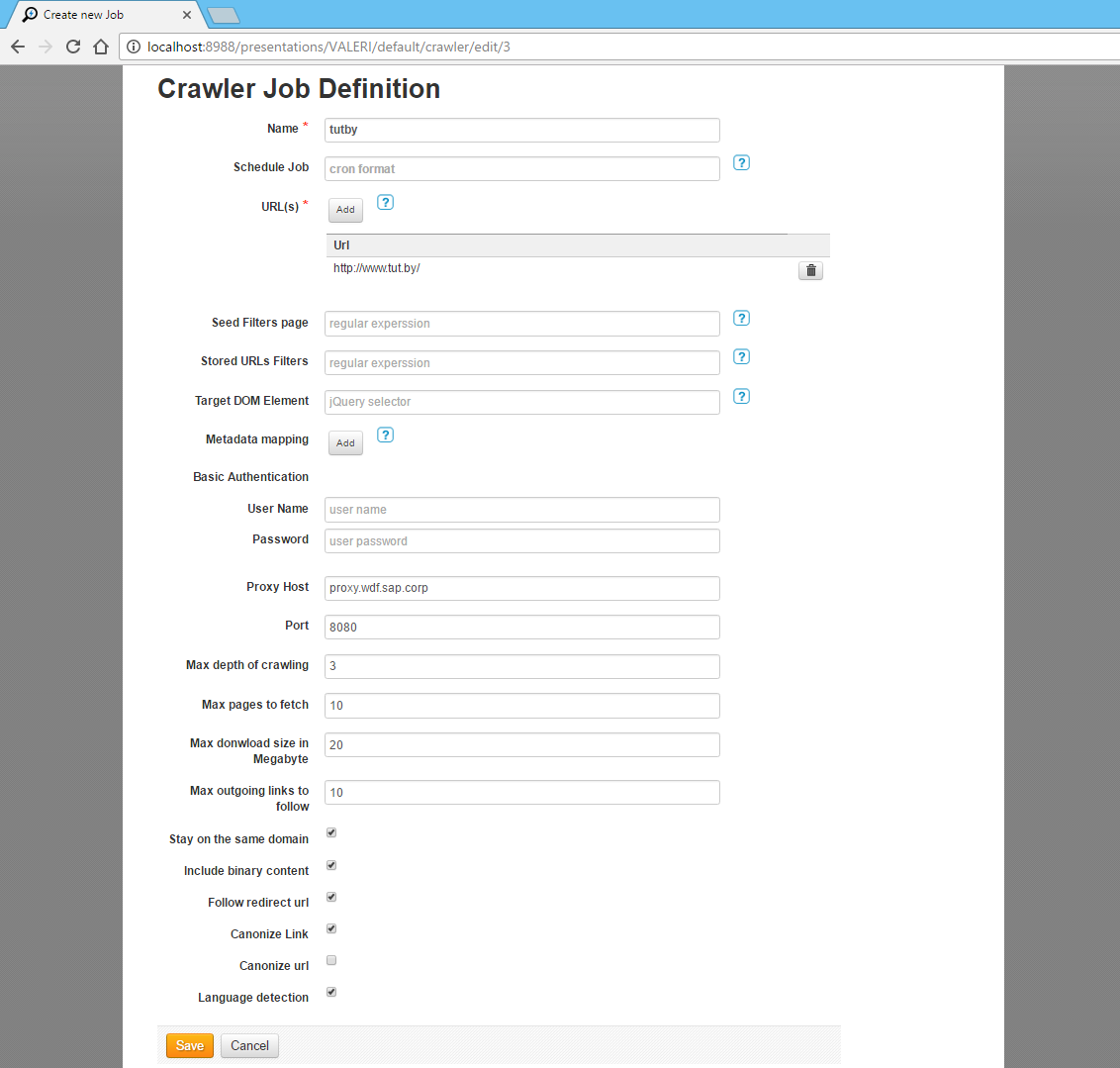


Рисунок 3.22 – Создание задачи краулера

На странице создания задачи краулера есть возможность задать все возможные его параметры. Имя нужно выбрать таким образом, чтобы оно поясняло суть задачи. Задаче можно назначить расписание запуска в поле «Schedule job» в формате cron. Например, чтобы краулер автоматически начинал работать каждую субботу в 8 утра, нужно записать следующее выражение: «0 0 8 ? \* SAT \*». В поле URL нужно внести адрес страницы, с которой краулер начнет свою работу. Есть возможность добавить несколько страниц. Чаще всего это стартовые страницы веб-приложения, так как именно с нее можно по ссылке перейти на любую следующую страницу. В поле Seed Filter page можно добавить свои фильтры для контроля посещения веб-страниц, используя регулярное выражение. В поле Stored URLs Filter можно записать регулярное выражение, которое позволит определить, какие страницы из проходимых краулером должны быть сохранены. В поле Target DOM Element можно внести селектор jQuery, из которого будет сохраняться контент. Метаданные могут быть определены с помощью специальной кнопки «Add» напротив текста Metadata mapping. Если страница, которую необходимо сохранить, не может быть доступна без аутентификации, то имя пользователя и пароль следует определить в разделе Basic Authentication. Оставшиеся поля помогут определить максимальное количество страниц для извлечения, глубину краулера, максимальный объем загружаемых данных в мегабайтах, максимальное количество ссылок для перехода и другое.

Заполним поля как указано на рисунке 3.22, создав новую задачу краулера. Теперь она отобразится в списке задач, рисунок 3.23.

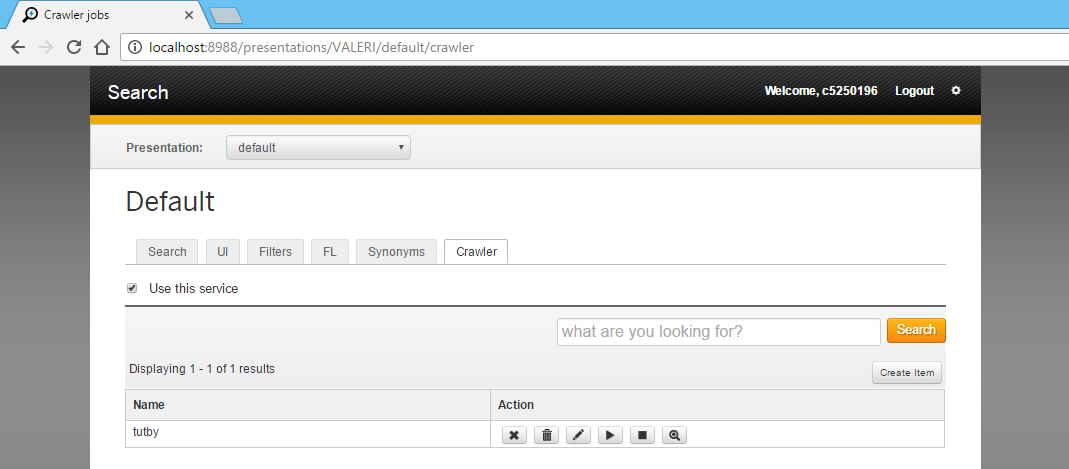


Рисунок 3.23 – Созданная задача tutby

Нажмем на значок запуска, чтобы запустить задачу, и кликнем по значку лупы, чтобы перейти на страницу, где будут отображены подробности выполнения задачи (рисунок 3.24). Когда краулер завершит свою работу, данная страница будет отображать результаты обработки либо в колонке success, либо в колонке failed. В данном случае можно увидеть, что успешно обработана одна страница.

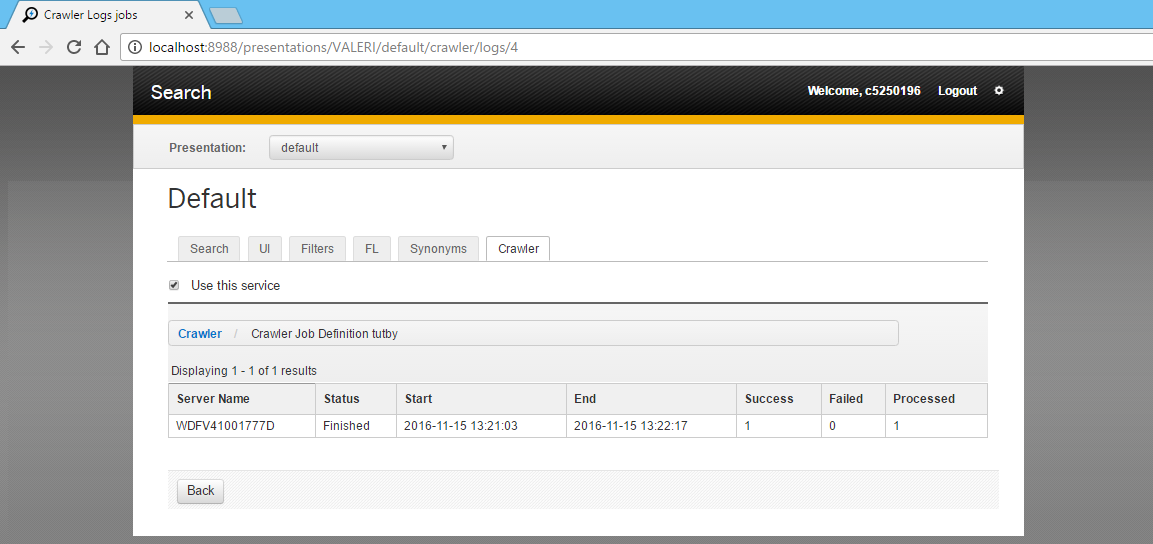


Рисунок 3.24 – Результаты работы краулера

Рассмотрим подробнее, какие возможности есть у владельца репозитория после настройки и запуска краулера. На рисунке 3.24 можно обратить внимание на панель навигации. Перейдем на первую из шести вкладок под названием «Search». Владелец репозитория будет направлен на страницу, как на рисунке 3.25, где у него есть возможность конфигурировать алгоритм поиска с помощью настроек полнотекстового поиска по полям модели хранилища. При переходе на вкладку «UI» владелец будет направлен на страницу настройки графического интерфейса для пользователей поисковика (рисунок 3.26). Также есть возможность настроить общие для всех пользователей фильтры поиска, перейдя во вкладку «Filters» (рисунок 3.27). Настройка рекомендуемых ссылок осуществляется на вкладке «FL», страница которой представлена на рисунке 3.28. Также присутствует возможность добавления синонимов. Это делается на странице «Synonyms» (рисунок 3.29), переход на которую осуществляется по соответствующей вкладке.

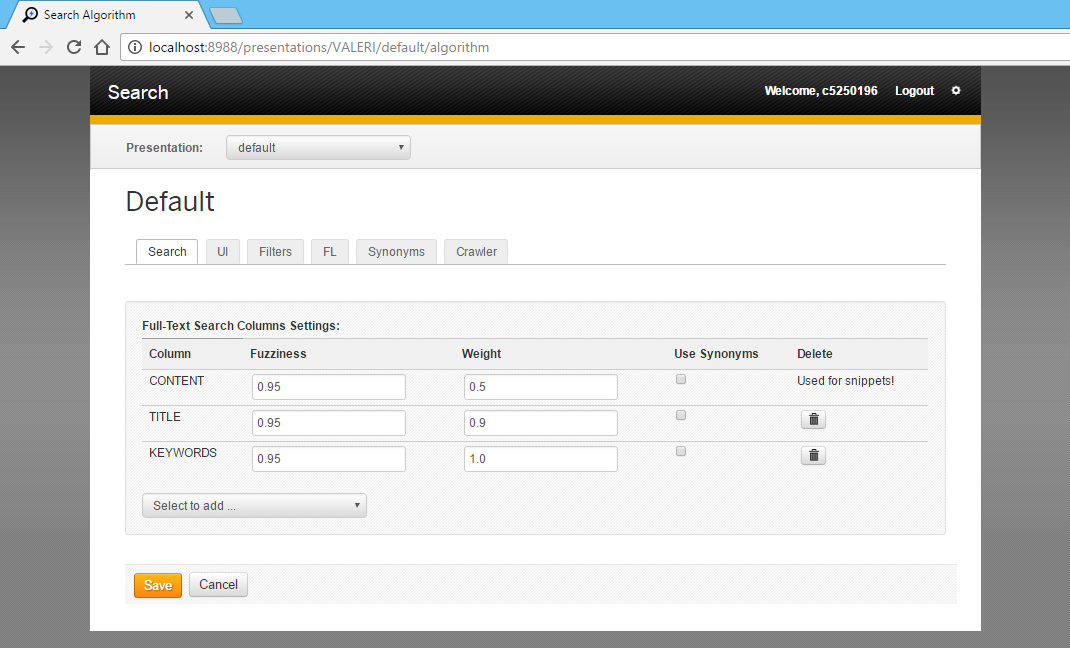


Рисунок 3.25 – Настройка поиска

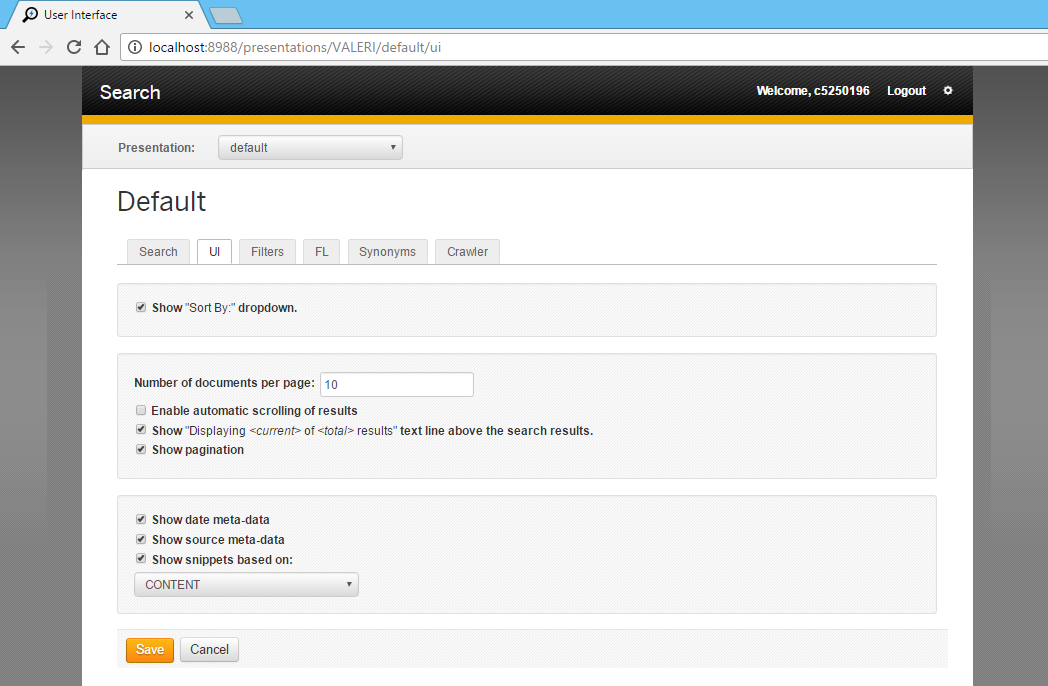


Рисунок 3.26 – Настройка графического интерфейса пользователей

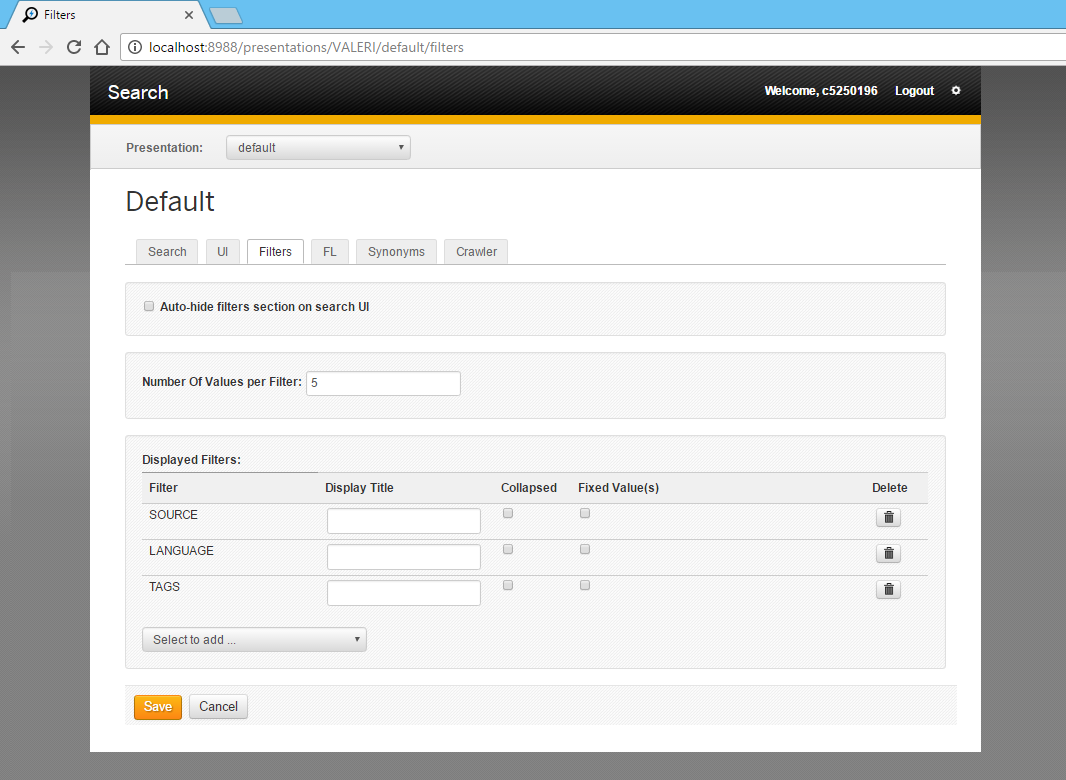


Рисунок 3.27 – Настройка фильтров

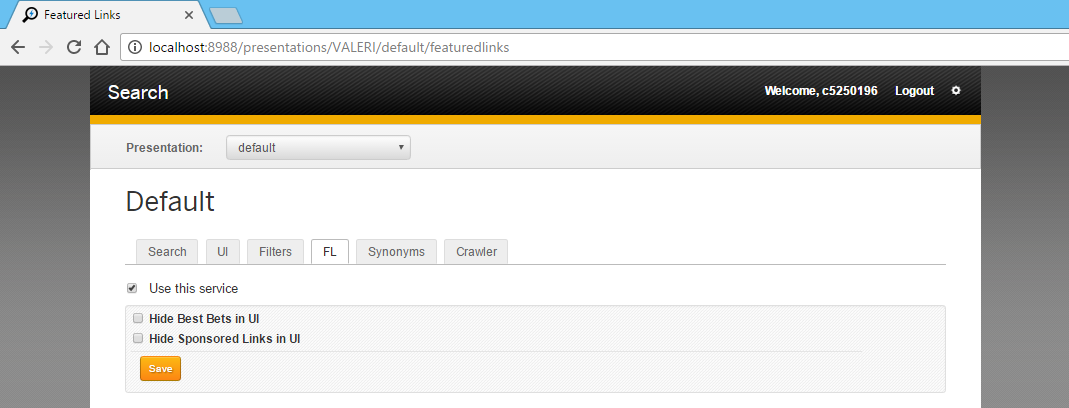


Рисунок 3.28 – Настройка рекомендуемых ссылок



Рисунок 3.29 – Настройка синонимов

* + 1. Результаты тестирования для администратора. Администраторы работают с конкретной базой, заранее созданной владельцем. Предположим, что владелец создал базу «VALERI» и добавил в нее администратора «admin\_client». На главной странице (рисунок 3.30) администратору необходимо ввести свое имя и пароль, а также выбрать из выпадающего списка название репозитория, с которым необходимо работать. После нажатия кнопки «Log in» при правильном имени и пароле администратор получит доступ к управлению выбранным репозиторием и увидит страницу как на рисунке 3.31.

На рисунке 3.31 слева отображено меню администратора, используя которое он может конфигурировать базу, наполнять ее содержимым, просматривать и проверять настройки. После успешного входа в систему администратор находится на первой в меню странице – странице репозиториев. Ему предоставляется просмотреть базы, к которому у него есть доступ и удалить базу, если необходимо. Перейдем ко второму пункту меню «Configurations». Администратор увидит страницу конфигураций базы (рисунок 3.32).

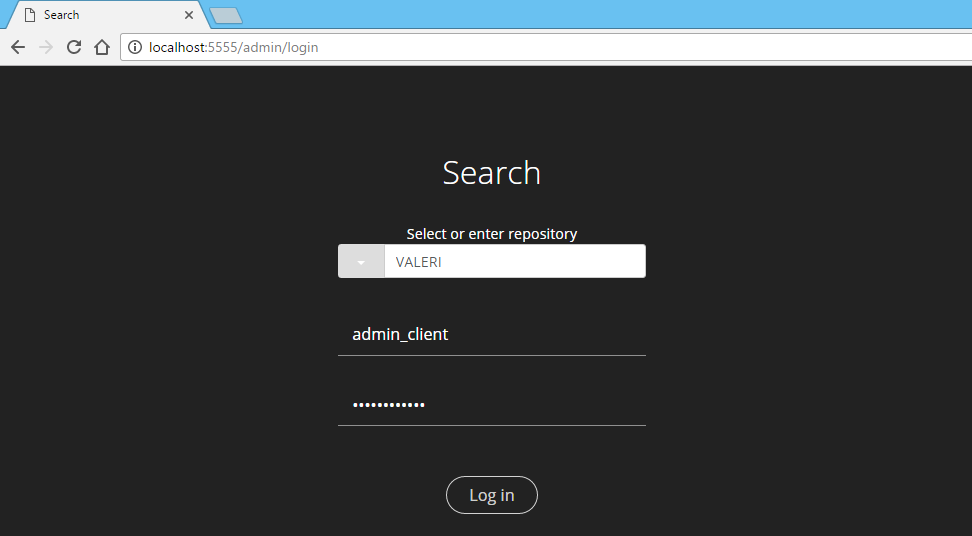


Рисунок 3.30 – Главная страница администратора

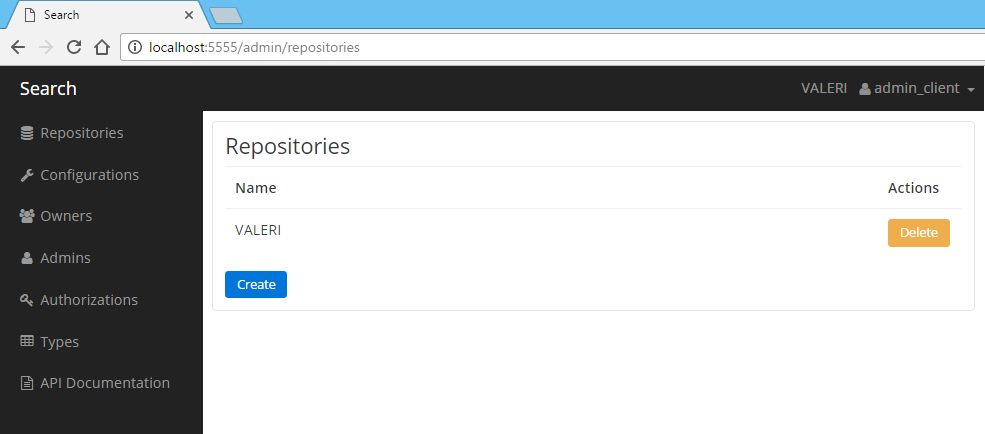


Рисунок 3.31 – Управление выбранным хранилищем



Рисунок 3.32 – Конфигурации базы «VALERI»

На данной странице можно увидеть настройки базы: какого типа база, когда она была создана, кто создал, описание. Можно удалить или добавить конфигурации.

Перейдем к следующему пункту меню «Owners» - владельцы базы. На рисунке 3.33 представлена страница с владельцами базы «VALERI». Информация на странице носит справочную информацию.

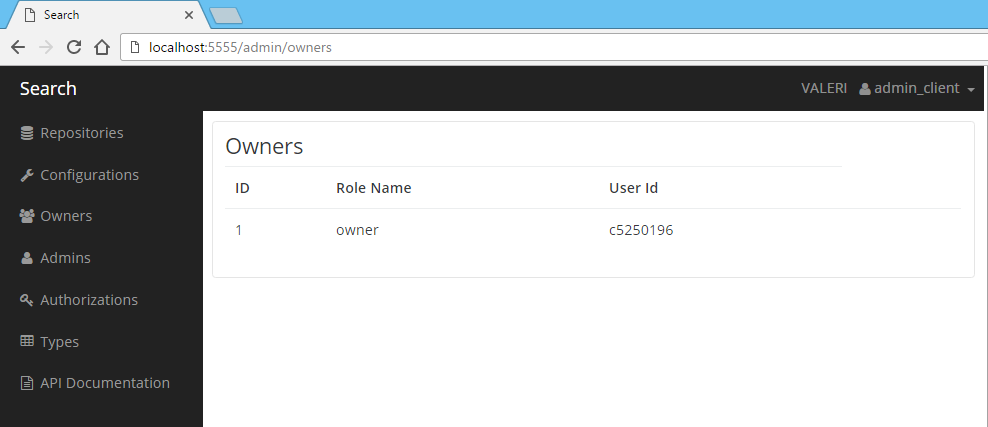


Рисунок 3.33 – Владельцы репозитория

Следующий пункт меню «Admins» направит на страницу администраторов, рисунок 3.34. На этой странице администратор может поменять свой пароль или добавить новых администраторов.

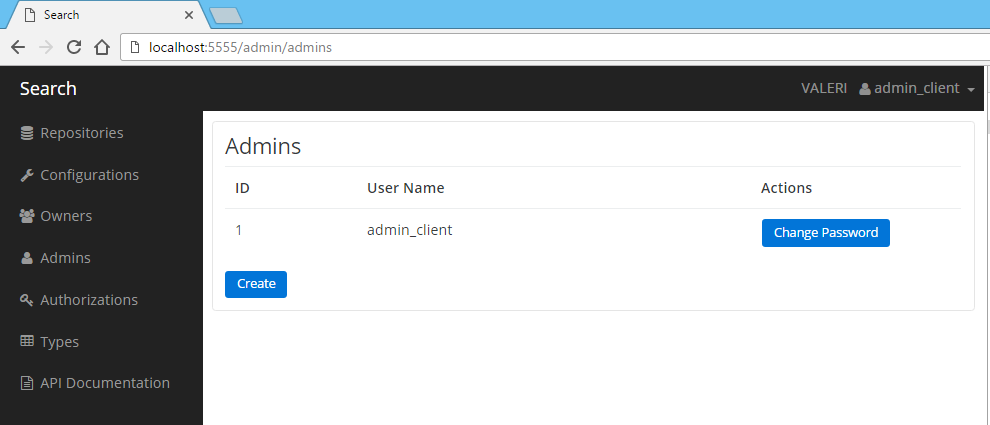


Рисунок 3.34 – Администраторы репозитория

Страница настроек прав доступа, перейти на которую можно кликнув пункт меню «Authorizations», предоставляет информацию о настроенных правах различных пользователей и групп пользователей (рисунок 3.35).

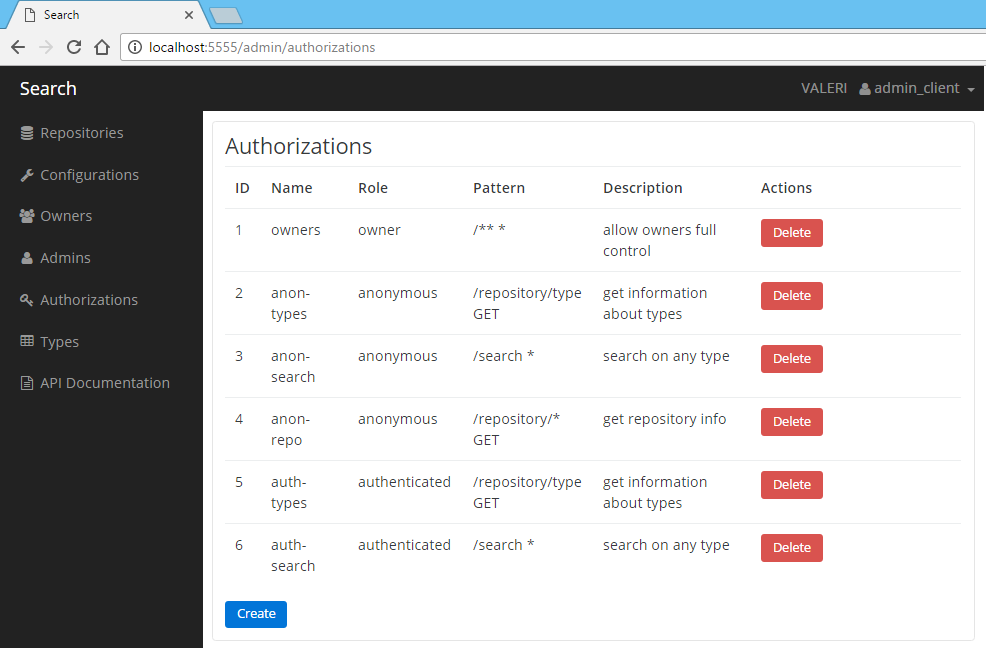


Рисунок 3.35 – Страница настроек прав доступа

При необходимости на странице 3.35 можно удалить лишние конфигурации доступа к API приложения, или же создать новые, перейдя по ссылке «Create» внизу страницы. Для создания новой настройки доступа на странице 3.36 необходимо заполнить поле имя, роль пользователя или группы пользователей, паттерн и описание.

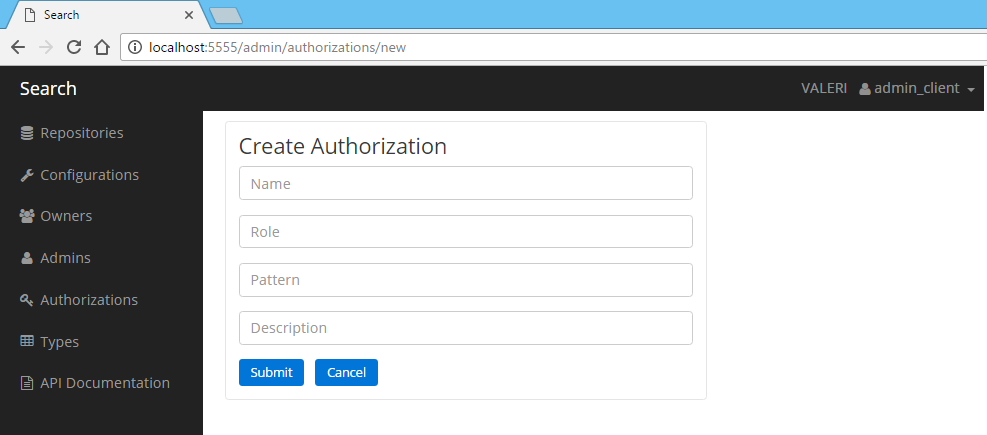


Рисунок 3.36 – Страница создания новой настройки доступа к API поискового приложения

Перейдем на страницу типов в базе, кликнув по пункту меню «Types». Как говорилось ранее, типы представлены отдельными таблицами в базе данных. Но для настроек администратору доступны лишь две стандартные таблицы: GROUPS и CONTENT. Можно увидеть доступные типы на рисунке 3.37.

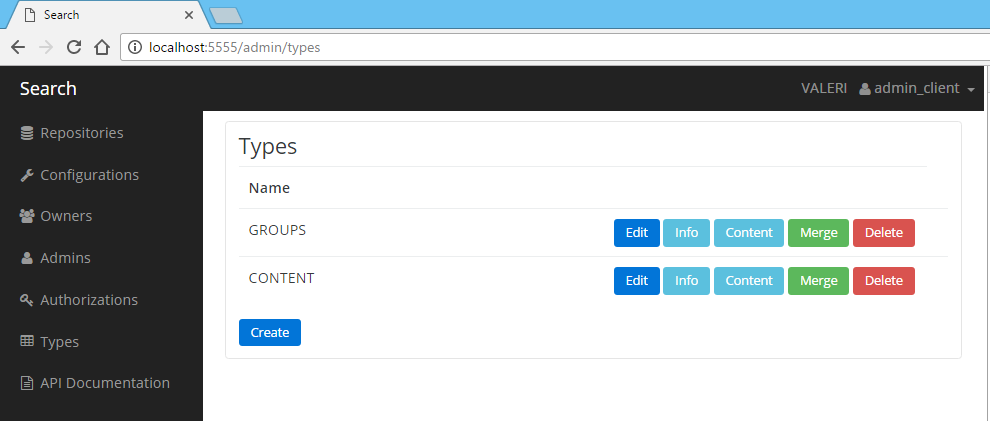


Рисунок 3.37 – Типы, доступные для настройки администратору

По умолчанию все содержимое обрабатываемых краулером страниц сохраняется в таблице CONTENT. Но иногда возникает необходимость настроить отдельное хранилище для определенных документов и веб-страниц. Перейдем по ссылке «Create» внизу страницы на рисунке 3.37 для создания нового типа. Администратор будет направлен на страницу, где он может создать новые поля типа (колонки таблицы в базе). Заполним поля по примеру на рисунке 3.38, кликнем «Submit».

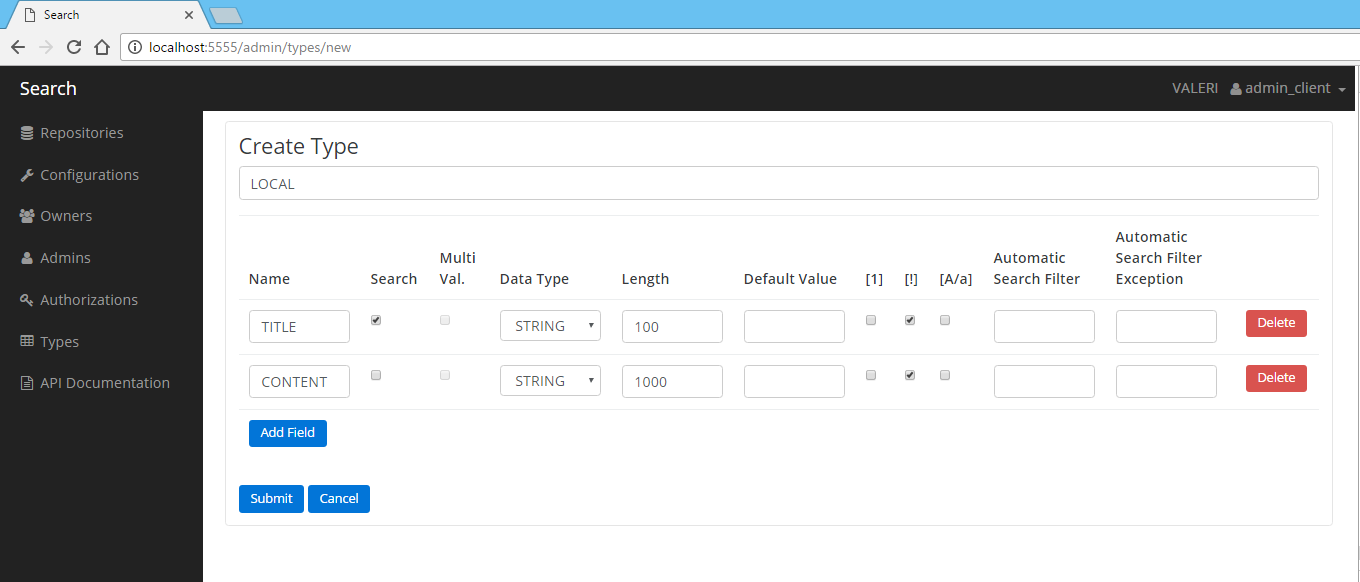


Рисунок 3.38 – Пример создания нового типа «Local»

После подтверждения создания нового типа администратор будет перенаправлен на страницу со списком типов, где он сможет увидеть добавленный тип, рисунок 3.39.

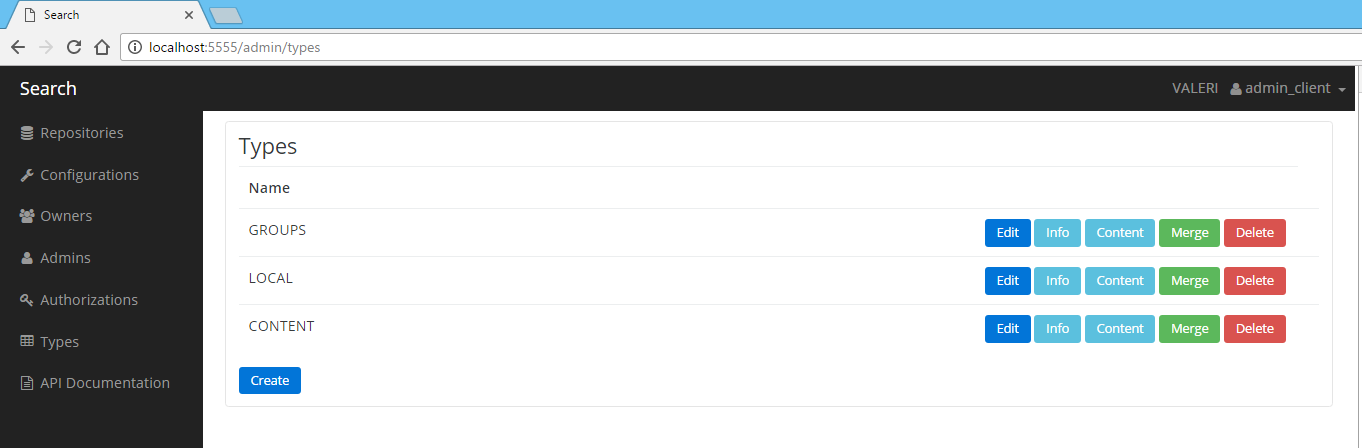


Рисунок 3.39 – Результат добавления нового типа «LOCAL»

Для каждого типа в списке доступны следующие функции: редактировать тип «Edit», просмотреть информацию о типе «Info», управлять содержимым типа «Content», объединить типы «Merge», удалить тип «Delete». Рассмотрим подробнее управление содержимым типа, кликнув по соответствующей кнопке в строке с типом LOCAL. Администратор будет направлен на страницу, как на рисунке 3.40.

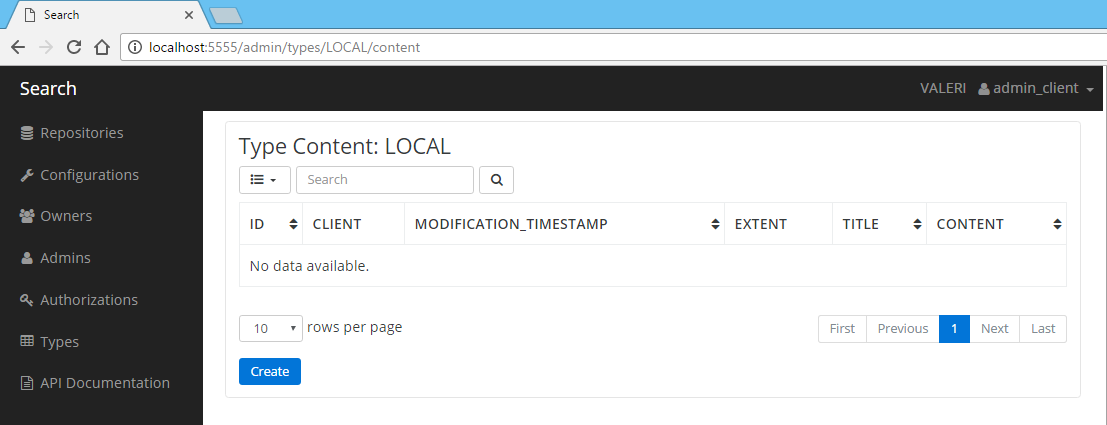


Рисунок 3.40 – Содержимое типа LOCAL

Как видно из рисунка 3.40, пока в таблице LOCAL нет содержимого. Оно может быть добавлено либо при обработке веб-страниц краулером, либо администратором вручную. Рассмотрим вторую опцию и добавим в таблицу LOCAL содержимое, кликнув по клавише «Create» внизу страницы. Необходимо будет заполнить только созданные ранее поля типа, а поля с датой и идентификатором будут добавлены автоматически. На рисунке 3.41 приведен пример заполнения полей при добавлении контента.

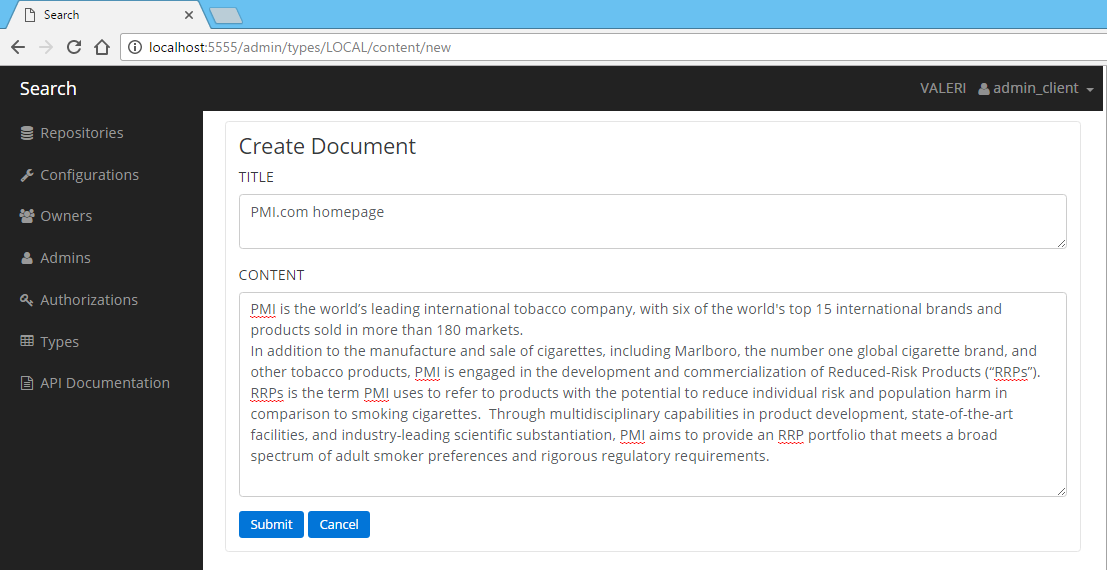


Рисунок 3.41 – Страница создания содержимого типа LOCAL

Добавим содержимое с других страниц веб-сайта pmi.com для примера. Тогда страница содержимого типа LOCAL будет выглядеть как на рисунке 3.42.

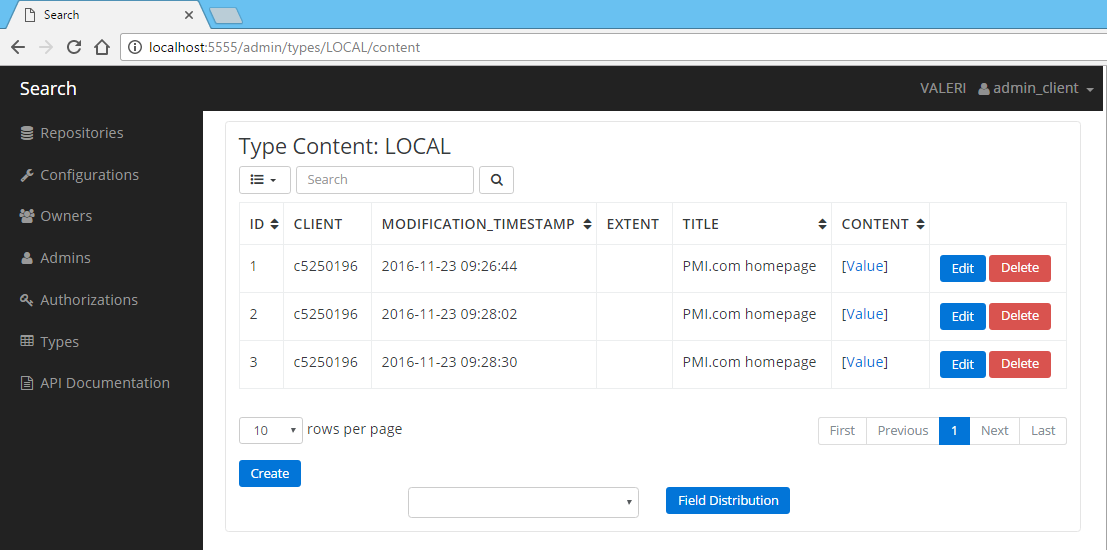


Рисунок 3.42 – Результат добавления содержимого в таблицу LOCAL

Находясь на странице с содержимым типа, можно перейти на страницу редактирования содержимого, есть возможность удалить содержимое, а также просмотреть распределение значений по выбранному полю (рисунок 3.43).

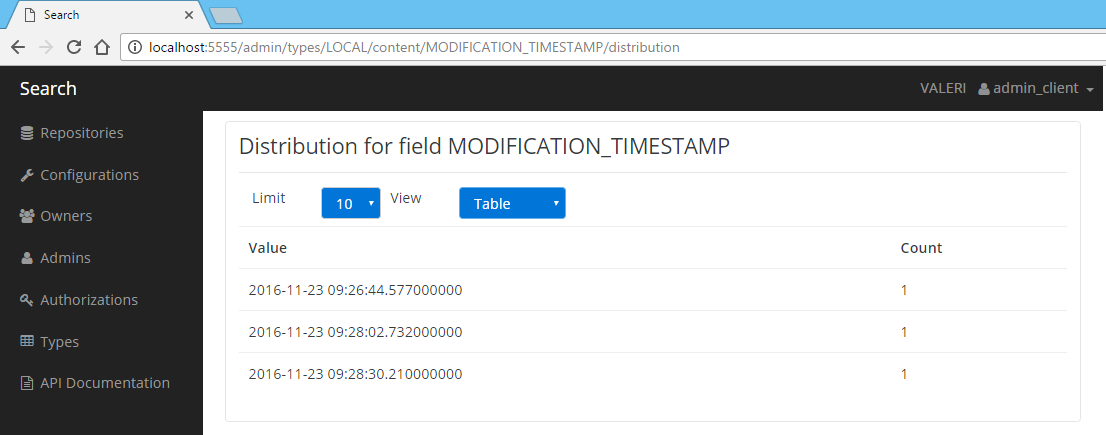


Рисунок 3.43 – Распределение значений в табличном формате

Чтобы просмотреть распределение по полю «MODIFICATION\_TIMESTAMP», выберем его в выпадающем списке внизу таблицы и нажмем кнопку «Field Distribution». Распределение значений можно просмотреть в одном из трех форматов: в виде таблицы, секторная и столбиковая диаграммы. Изменить представление распределения поля можно с помощью выпадающего списка. Страницы с разными форматами представлены на рисунках 3.43, 3.44 и 3.45.

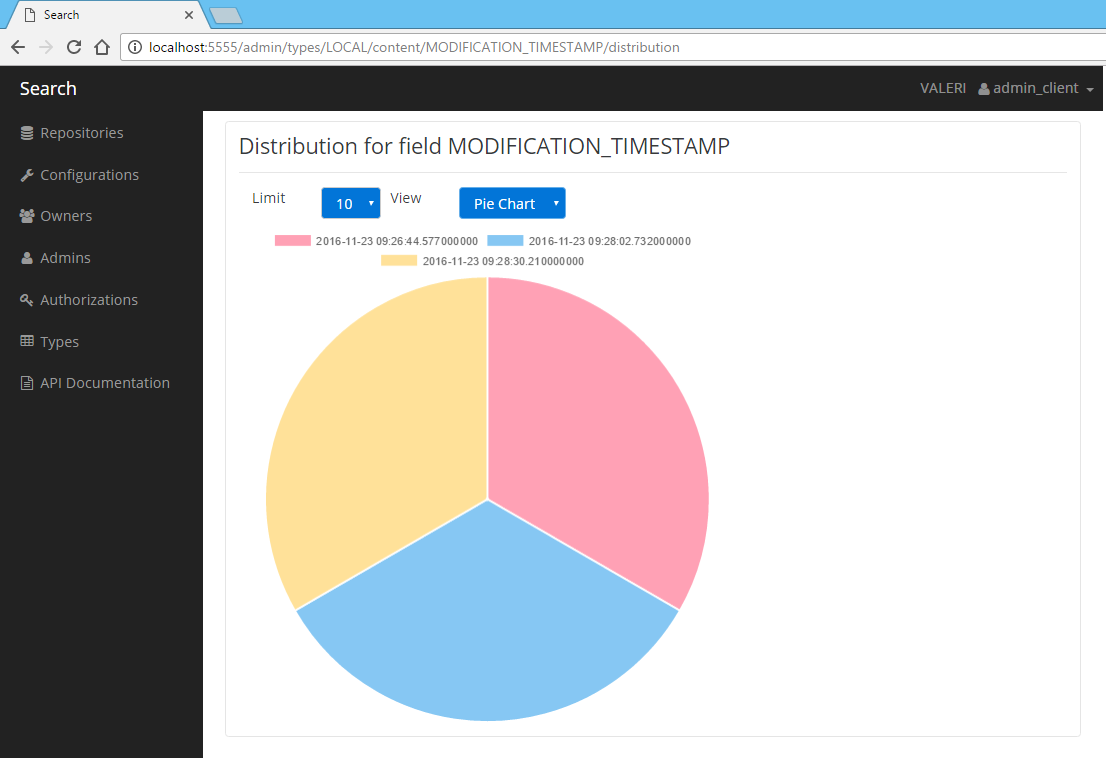


Рисунок 3.44 – Распределение значений в виде секторной диаграммы

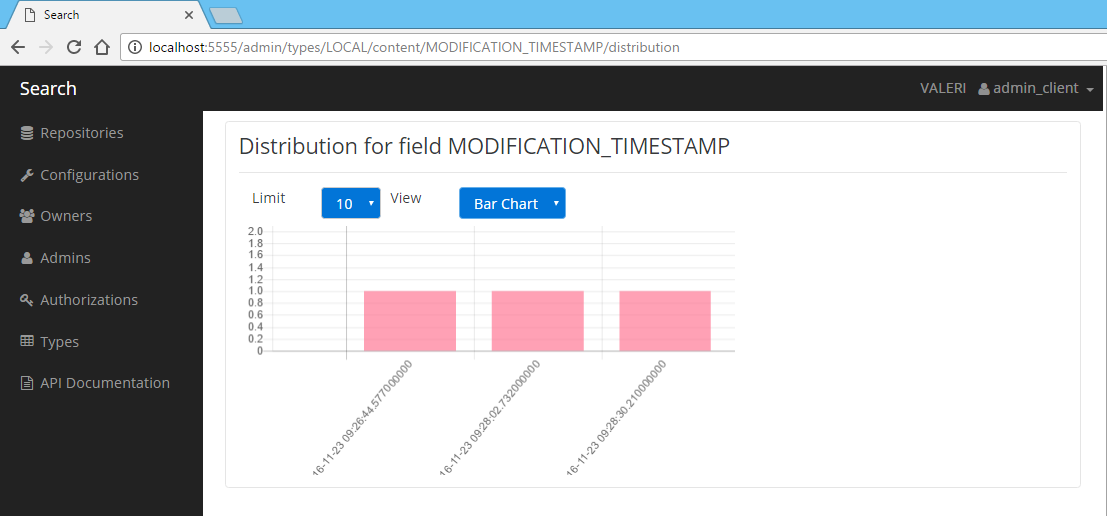


Рисунок 3.45 – Распределение значений в виде столбиковой диаграммы

* + 1. Результаты тестирования для сотрудника компании. Сотрудник компании открывает главную страницу поисковика, рисунок 3.46. Здесь ему предлагается ввести поисковый запрос. Введем любое слово, например, HANA, и нажмем кнопку «Search». После этого сотрудник компании будет перенаправлен на страницу с результатами поиска, как на рисунке 3.47.

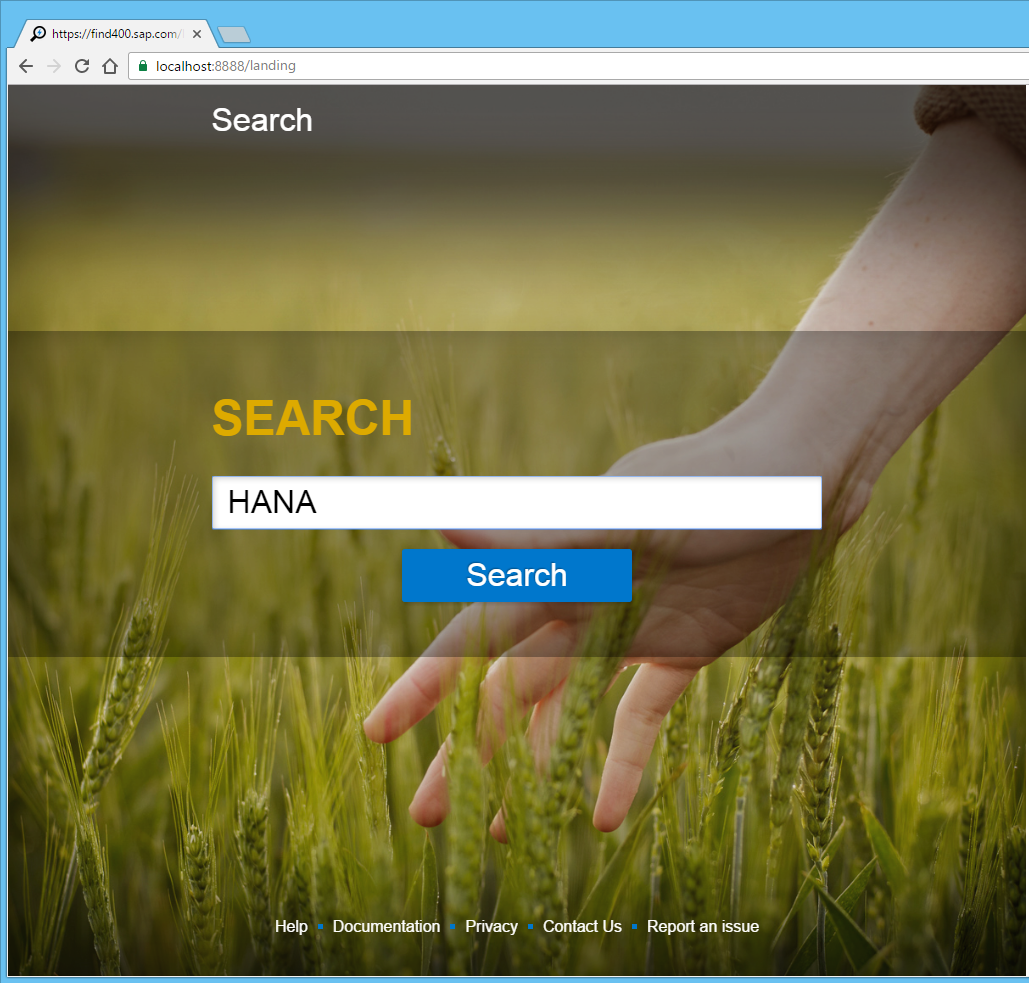


Рисунок 3.46 – Главная страница поиска для сотрудника компании

Рассмотрим подробнее выданные результаты. Страница на рисунке 3.47 условно разбита на 8 логических секций. Они выделены красными прямоугольниками на рисунке 3.48.

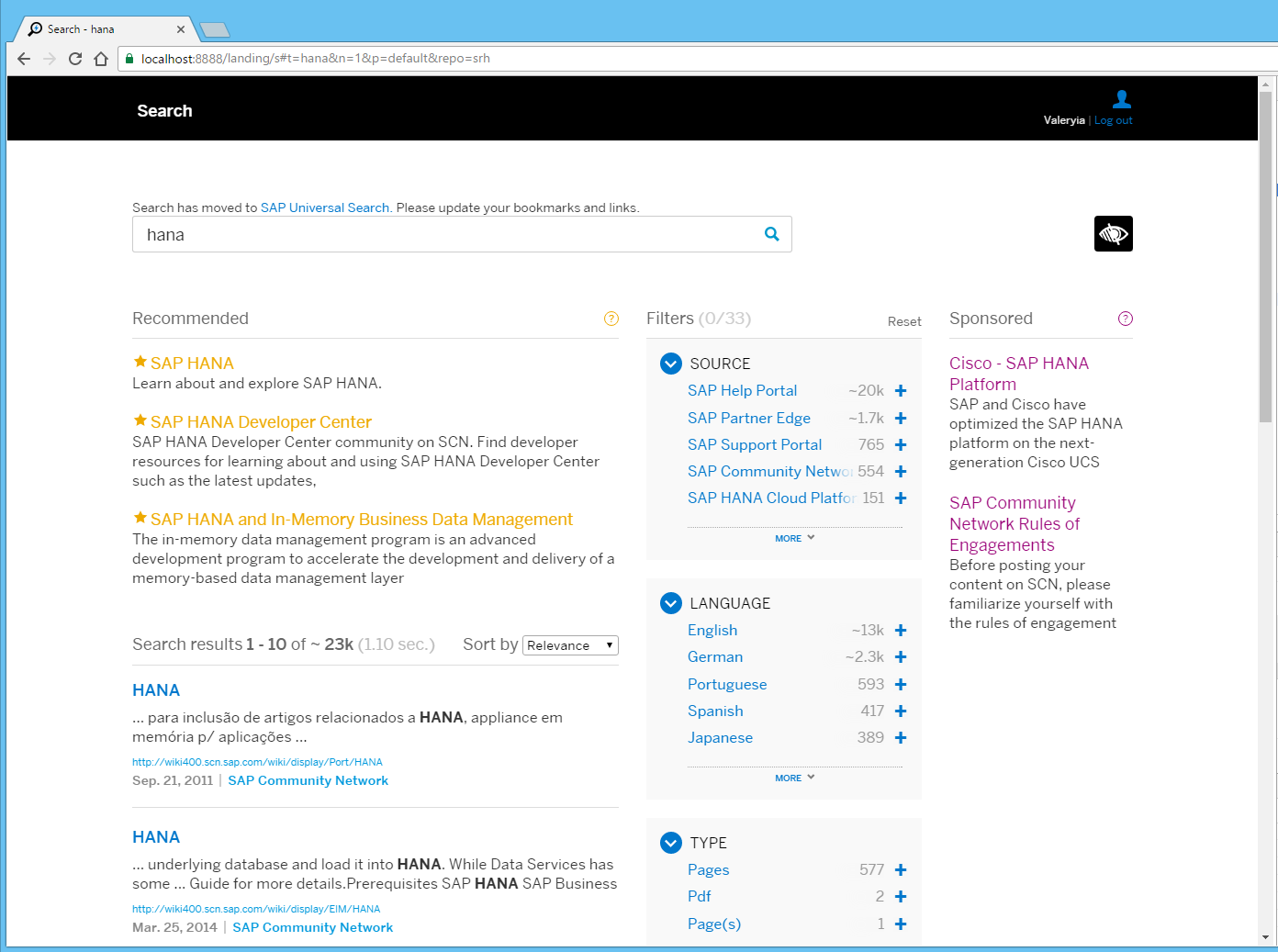


Рисунок 3.47 – Результаты поиска по запросу «HANA»

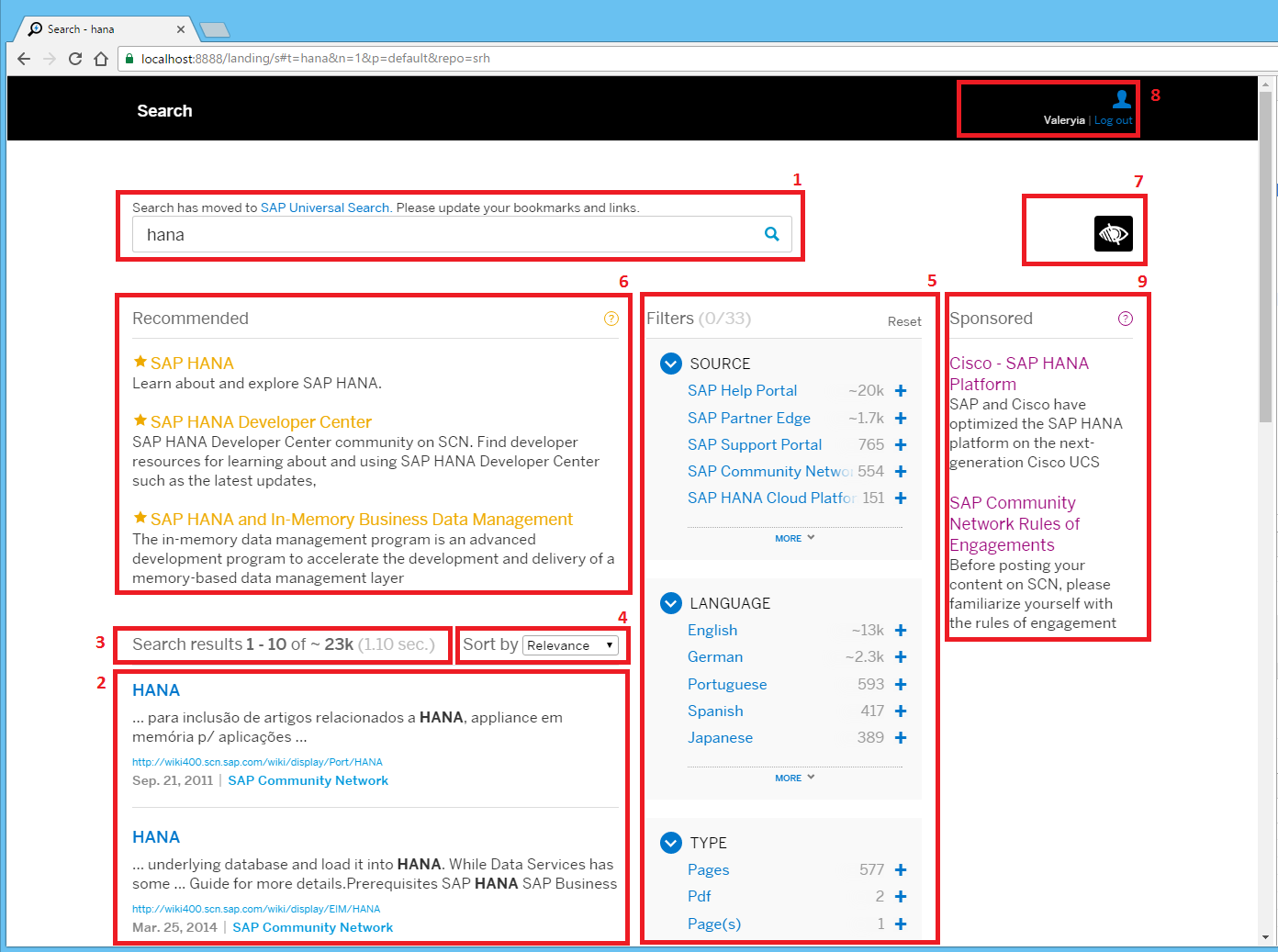


Рисунок 3.48 – Секции на странице результатов по запросу «HANA»

Секции содержат следующую информацию:

1. Строка с поисковым запросом.
2. Результаты по запросу.
3. Информация о результатах по запросу: количество найденных результатов, время, затраченное на поиск.
4. Сортировка результатов. Есть возможность сортировать по релевантности, или отображать сначала новые результаты.
5. Фильтры. Всего доступно несколько фильтров: источник данных, язык страницы, тип результата и другие.
6. Рекомендуемое.
7. Режим просмотра в темноте.
8. Информация о сотруднике.
9. Рекламные ссылки.

# ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ПРОЕКТА ПО РАЗРАБОТКЕ КОРПОРАТИВНОЙ ИНФОРМАЦИОННО-ПОИСКОВОЙ СИСТЕМЫ

## Описание функций, назначения и потенциальных пользователей корпоративной информационно-поисковой системы

Корпоративная информационно-поисковая система – программный продукт, который разрабатывается компанией «ЭПАМ Системз» по индивидуальному заказу для компании «SAP».

В настоящее время сотрудники компании «SAP» не имеют специализированной системы для автоматизации процесса поиска информации внутри компании. Для этих целей в компании используется каталог ресурсов, в котором добавление и обновление данных выполняется веб-мастерами вручную. Поиск данных в каталоге ресурсов, бесспорно, упрощает ежедневную работу сотрудников, однако по-прежнему занимает много времени. Такой каталог не может полностью решить все проблемы, возникающие перед сотрудниками.

Разрабатываемая информационно-поисковая система позволяет:

* облегчить процесс поиска информации сотрудниками внутри компании;
* сократить время, затрачиваемое сотрудниками на поиск документов и электронных ресурсов в компании;
* автоматизировать работу веб-мастеров по обновлению каталога всех веб-ресурсов компании, заменив ручное обновление каталога поисковыми роботами, способными добавлять данные, удалять несуществующие ссылки;
* улучшить показатели деятельности сотрудников, часто работающих с электронными документами;
* исключить случаи повторного создания существующих документов;
* управлять доступом персонала к веб-ресурсам компании, включая обеспечение секретности для закрытой информации.

Основными пользователями системы станет большинство сотрудников компании «SAP», ежедневно работающих с веб-ресурсами. Наибольшую пользу внедрение программного продукта принесет:

* бухгалтерам и аналитикам;
* менеджерам;
* программистам;
* персоналу отдела кадров.

Экономический эффект компании «ЭПАМ Системз» представляет собой прибыль, полученную от разработки программного продукта «Корпоративная информационно-поисковая система», заказчиком которого является компания «SAP». Если же рассматривать внедрение программного обеспечения, то для компании-заказчика можно обозначить неэкономический эффект.

Экономическая целесообразность инвестиций в разработку и использование автоматизированной системы управления задачами и ресурсами осуществляется на основе расчета и оценки уровня рентабельности при реализации программного продукта.

Таким образом, уникальный продукт разрабатывается по индивидуальному заказу и предназначен для использования внутри компании «SAP», являющейся организацией-заказчиком. Результатом внедрения станет замена неэффективных каталогов веб-страниц удобной поисковой системой, позволяющей сотрудникам в считанные секунды найти искомые документы и прочие веб-ресурсы, получив релевантные результаты.

## Расчет затрат на разработку корпоративной информационно-поисковой системы

Расчет затрат будет осуществлен в разрезе следующих статей:

* затраты на основную заработную плату разработчиков;
* затраты на дополнительную заработную плату разработчиков;
* отчисления на социальные нужды;
* прочие затраты (амортизация оборудования, расходы на электроэнергию, командировочные расходы, накладные расходы и т.п.).

Основная заработная плата исполнителей проекта определяется по следующей формуле:

, (4.1)

где *n* – количество исполнителей;

Тчi – часовая тарифная ставка *i*-го исполнителя, руб.;

Тч – количество часов работы в день (8 ч.);

Фэi – плановый фонд рабочего времени *i*-го исполнителя, дней;

К – коэффициент премирования (1,35).

В настоящий момент месячная тарифная ставка 1-го разряда в компании составляет 170 руб. Расчётная норма рабочего времени на 2017 год для пятидневной рабочей недели составляет 168 часов. Штат исполнителей проекта состоит из семи специалистов: руководителя проекта, ведущего программиста, двух инженеров-программистов первой категории, двух инженеров-программистов второй категорий и бизнес-аналитика.

Определим месячную заработную плату исполнителей, умножив месячную тарифную ставку 1-го разряда на тарифный коэффициент, соответствующий категории специалиста.

Месячная заработная плата руководителя проекта будет равна:

руб.

Месячная заработная плата ведущего программиста составит:

руб.

Месячная заработная плата программиста второй категории составит:

руб.

Месячная заработная плата программиста первой категории:

руб.

Месячная заработная плата бизнес-аналитика:

руб.

После того, как был составлен список работ для выполнения проекта, было определено, что программистам первой и второй категории понадобится 100 дней для выполнения своих задач. Руководитель проекта и ведущий программист будут работать 80 дней, а бизнес-аналитик 50 дней. Расчет основной заработной платы исполнителей проекта представлен в таблице 4.1.

Таблица 4.1 – Расчет основной заработной платы разработчиков ПП

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Исполнитель | Разряд | Месячный оклад, руб. | Дневная ставка, руб. | Плановый фонд рабочего времени, дн. | Основная заработная плата, руб. |
| 1 | 2 | 3 | 4 | 5 | 6 |
| Руководитель проекта | 13 | 676,60 | 32,24 | 80 | 2577,52 |
| Ведущий программист | 14 | 724,20 | 34,48 | 80 | 2758,86 |
| Программист 2-ой категории | 12 | 637,50 | 30,35 | 100 | 3035,71 |
| Программист 1-ой категории | 9 | 535,50 | 25,50 | 100 | 2550,00 |
| Бизнес-аналитик | 8 | 516,80 | 24,60 | 50 | 1230,48 |
| Итого | | | | | 12152,57 |
| Премии (35%) | | | | | 4253,40 |
| Итого затраты на основную заработную плату разработчиков | | | | | 16405,97 |

Дополнительная заработная плата исполнителей проекта определяется по формуле:

где – норматив дополнительной заработной платы, 10%.

После подстановки значений в формулу (4.2) дополнительная заработная плата составит:

Отчисления в фонд социальной защиты населения и на обязательное страхование () определяются в соответствии с действующими законодательными актами по формуле:

, (4.3)

где – норматив отчислений в фонд социальной защиты населения и на обязательное страхование, 34,6 %.

Размер отчислений в фонд социальной защиты населения и на обязательное страхование согласно формуле (4.3) составит:

руб.

Расходы по статье «Машинное время» () определяются по формуле:

, (4.4)

где  – цена одного машино-часа;

 – количество часов работы в день;

 – длительность проекта.

Стоимость машино-часа на предприятии составляет 1 руб. Как указано выше, разработкой проекта занимаются 7 специалистов, а общее машинное время займет 410 человеко-дней. Определим по формуле (4.4) затраты по статье «Машинное время»:

руб.

Затраты по статье «Накладные расходы» (РН), которые включают затраты на услуги связи (телефон, интернет), арендную плату за офис и содержание офиса, обучение работников, определяются по формуле:

 (4.5)

где – норматив накладных расходов, 50%.

После подстановки значений в формулу (4.5) накладные расходы составят:

Общая сумма расходов по всем статьям сметы () на ПО рассчитывается по формуле:

По формуле (4.6) получаем сумму расходов по всем статьям сметы:

руб.

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию ПС (), которые определяются по формуле:

 (4.7)

где – норматив расходов на сопровождение и адаптацию, 20%.

Затраты на сопровождение и адаптацию программного продукта по формуле (4.7) составят:

Общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) как полная себестоимость ПС () определяется по формуле:

(4.8)

Полная себестоимость разрабатываемого программного средства, рассчитанная по формуле (4.8) составит:

руб.

Все расчеты сметы затрат и себестоимости программного средства сведены в таблицу 4.2.

Таблица 4.2 – Смета затрат на разработку ПС

|  |  |  |
| --- | --- | --- |
| Наименование статей | Условное обозначение | Значение (руб.) |
| 1 | 2 | 3 |
| Основная заработная плата команды разработчиков |  | 16405,97 |
| Дополнительная заработная плата команды разработчиков |  | 1640,60 |
| Отчисления в фонд социальной защиты |  | 6244,11 |
| Машинное время |  | 3280 |
| Накладные расходы |  | 8202,99 |
| Затраты на сопровождение и адаптацию |  | 7154,73 |
| Полная себестоимость ПС |  | 42928,39 |

Как видно из таблицы 4.2, полная себестоимость программного продукта составит 46864,396 рублей. Полученную себестоимость можно использовать для определения экономического эффекта от разработки и продажи программного продукта.

## Определение экономического эффекта от разработки программного продукта

Так как программный продукт «Корпоративная информационно-поисковая система» разрабатывается по индивидуальному заказу компанией «ЭПАМ Системз» для компании «SAP», то экономический эффект рассчитывается для компании-разработчика, то есть для «ЭПАМ Системз».

Экономическим эффектом организации-разработчика выступает прибыль, полученная от реализации программного продукта под заказ сторонней организации.

Поскольку в данном случае разрабатываемый программный продукт «Корпоративная информационно-поисковая система» является уникальным и создаётся под нужды и требования конкретного заказчика, то его цена определяется в процессе переговоров между компанией-заказчиком и организацией-разработчиком.

Переговоры были проведены до начала разработки. Для определения цены продукта были проведены маркетинговые исследования и использована средняя цена на программные продукты, выполняющие аналогичные задачи.

В результате проведенных маркетинговых исследований было выяснено, что цена программного обеспечение со схожими выполняемыми функциями равна 50000 долларов. Примем 1 доллар = 1,89 руб. Получаем, что:

50000 \* 1,89 = 94500 руб.

Цена программного продукта состоит из трёх частей: себестоимости (суммы всех затрат на производство и реализацию продукта), прибыли и косвенных налогов. Так как компания ООО «ЭПАМ Системз» является резидентом Парка высоких технологий, она освобождена от уплаты НДС [17].

Следовательно, расчёт прибыли осуществляется по формуле (4.9):

(4.9)

где – цена реализации ПО заказчику, руб.;

– сумма расходов на разработку ПО, руб;

– прибыль, получаемая организацией-разработчиком от реализации данного ПО, руб.

Таким образом, подставив значения в формулу (4.9), получим прибыль:

П = 94500 – = 51571,61 руб.

Рентабельность от реализации программного продукта «Корпоративная информационно-поисковая система» можно рассчитать по формуле (4.10):

|  |  |
| --- | --- |
|  | (4.10) |

где – уровень рентабельности, %;

– прибыль, получаемая организацией-разработчиком от реализации данного ПО, руб.;

– полная себестоимость, руб.

Подставив значения в формулу (4.10), получим следующее:

В результате проведённых экономических расчётов были получены следующие показатели:

* себестоимость и отпускная цена программного продукта «Корпоративная информационно-поисковая система» равна руб.;
* договорная цена равна 94500 руб.;
* прибыль, получаемая от реализации программного продукта «Корпоративная информационно-поисковая система» составит 51571,61 руб.;
* уровень рентабельности равен 120%.

Таким образом, разработка данного программного продукта является эффективной с точки зрения организации-разработчика.

* 1. **Эффективность использования программного продукта**

Как было сказано выше, экономическим эффектом организации-разработчика выступает прибыль, полученная от реализации программного продукта заказчику. Компания-заказчик приобрела программное средство за 94500 руб. Если же рассматривать внедрение программного продукта в компании заказчика, то можно обозначить неэкономический эффект.

Так как использование поисковой системы будет оказывать косвенное влияние на показатели деятельности компании, данный эффект почти невозможно определить в стоимостном выражении. У компании-заказчика появятся выгоды иного характера: сократится время на поиск информации сотрудниками внутри компании, облегчится работа с документами.

Рассмотрим неэкономические выгоды компании-заказчика:

1. Значительное уменьшение времени на поиск документов. Быстрый доступ ко всей информации обеспечивает возможность оперативно принимать решения при выполнении сотрудниками своих обязанностей.
2. Снижение влияния человеческого фактора. Автоматизация поиска позволит сотрудникам сосредоточиться на решении более важных задач и обеспечит качественное и безошибочное выполнение операций.
3. Повышение доверия бизнеса к IT-компании. Если компания заявляет об использовании современных решений и продуктов для работы, это позволяет привлечь к себе внимание заказчиков.
4. Снижение вероятности появления рисков в сфере информационных технологий. Все документы для поиска собираются в общую базу знаний, которая в дальнейшем используется как источник лучших практик компании в управлении проектами и рисками.
5. Снижение рисков потери важной информации. Все электронные ресурсы компании будут проиндексированы, могут быть легко доступны сотрудникам. Изменения, внесенные сотрудниками в электронный документ, станут сразу доступны другим пользователям. В случае отсутствия такой общей системы, сотрудники сохраняют измененные документы на своих персональных компьютерах. Данные теряются и утрачиваются.
6. Эффективное внедрение в компанию новых сотрудников. Можно гораздо быстрее обучить нового сотрудника благодаря тому, что у него будет возможность просматривать любые необходимые ресурсы компании, не обращаясь за помощью к коллегам и не отрывая их от работы.

Поскольку разрабатываемый программный продукт выполняет весь заявленный функционал, является удобным способом оптимизации рабочего времени сотрудников, можно утверждать, что внедрение корпоративной информационно-поисковой системы является эффективным и полезным для заказчика.

# ВЫВОДЫ И ЗАКЛЮЧЕНИЯ

В результате выполнения дипломного проекта была разработана корпоративная информационно-поисковая система. Система разрабатывалась поэтапно, на каждом этапе были решены следующие задачи:

* 1. Изучена предметная область, рассмотрены уже существующие поисковые системы, их рейтинг в Беларуси, России и США. Были изучены особенности каждой системы.
  2. Рассмотрен и проанализирован процесс поиска информации в компании с точки зрения сотрудника компании, желающего найти электронные ресурсы. Результатом анализа стали модели «как-есть» и «как-должно-быть».
  3. Разработанные модели «как-есть» и «как-должно-быть» были проанализированы, изучена необходимая функциональность приложения и составлены варианты использования системы.
  4. Описана информационная модель системы.
  5. Рассмотрена последовательность взаимодействия компонентов системы при действиях пользователя. Выбраны компоненты и технологии для реализации системы. Составлены диаграммы компонентов и развертывания.
  6. Были составлены и описаны модели представления сервиса, включая диаграмму состояний UML.
  7. Разработана корпоративная информационно-поисковая система.
  8. Составлено руководство пользователя по развертыванию системы.
  9. Рассчитаны затраты на разработку корпоративной информационно-поисковой системы и рассчитан экономический эффект.
  10. Поисковая система была протестирована, произведена оценка выполненных задач.

Разработанная система уже внедрена в компании заказчика и находится на стадии тестирования. Использование системы позволяет сотрудникам компании снижать временные затраты на поиск электронных документов и информации в компании, сосредотачиваясь на выполнении своих прямых обязанностей.

Разработка и успешное тестирование корпоративной информационно-поисковой системы свидетельствует о том, что в результате выполнения данного дипломного проекта цель была достигнута. Поставленные задачи полностью выполнены.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Поисковая система - Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Поисковая\_система.

[2] Как работает Google Поиск? / Хабрахабр [Электронный ресурс.].– Режим доступа: https://habrahabr.ru/company/ua-hosting/blog/277819/.

[3] Поисковый индекс — Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Поисковый\_индекс.

[4] Рейтинг популярности поисковых систем на Март 2016 года [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.gs.seo-auditor.com.ru/sep/2016/03/.

[5] Самые популярные поисковые системы на 2015 год [Электронный ресурс]. – Режим доступа: http://revolweb.ru/prodvizhenie-sajtov/samye-populyarnye-poiskovye-sistemy-na-2015-god.

[6] ТОП-5 поисковых систем в Украине, России, Беларуси, Казахстане [Электронный ресурс]. – Электронные данные. – Режим доступа: http://uaweb.ua/publication/top\_5\_search\_engine\_2015.html.

[7] Корпоративные поисковые системы – Компьютер пресс [Электронный ресурс]. – Электронные данные. – Режим доступа: http://compress.ru/article.aspx?id=16576.

[8] Рынок предложения корпоративных поисковых систем [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.asknet.ru/Analytics/cnews1-3.htm.

[9] Инфопарк: Иностранное общество с ограниченной ответственностью «ЭПАМ Системз» [Электронный ресурс]. – Электронные данные. – Режим доступа: http://infopark.by/node/4680.

[10] Открытые системы: EPAM получила статус Золотого партнера SAP [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.osp.ru/resources/releases/?rid=3794.

[11] SAP – Википедия [Электронный ресурс]. – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/SAP.

[12] Администратор баз данных – Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Администратор\_баз\_данных.

[13] Enterprise Search: Enhancing Business Performance / O'Reilly Media - Martin White, Sebastopol, CA, 2015.

[14] Content Management Bible, Second Edition / Wiley Publishing, Inc. - Boiko Bob. – Indianapolis, Indiana, 2005. – 1168 с.

[15] Саттон М. Корпоративный документооборот: принципы, технологии, методологии внедрения. Перевод с англ. / Саттон М. – М.: Азбука, 2002. – 448 с.

[16] Палицын, В.А. Технико-экономическое обоснование дипломных проектов: методическое пособие для студ. всех спец. / В.А. Палицын. – Минск.: БГУИР, 2014. – 51 с.

[17] Горовой В.Г., Грицай А.В., Пархименко В.А. (кафедра экономики БГУИР, 2014), Экономическое обоснование проекта по разработке программного обеспечения.

# ПРИЛОЖЕНИЕ А (обязательное) Диаграмма вариантов использования информационно-поисковой системы



Рисунок А.1 – Диаграмма вариантов использования информационно-поисковой системы

# ПРИЛОЖЕНИЕ Б (обязательное) Схема алгоритма работы поискового робота

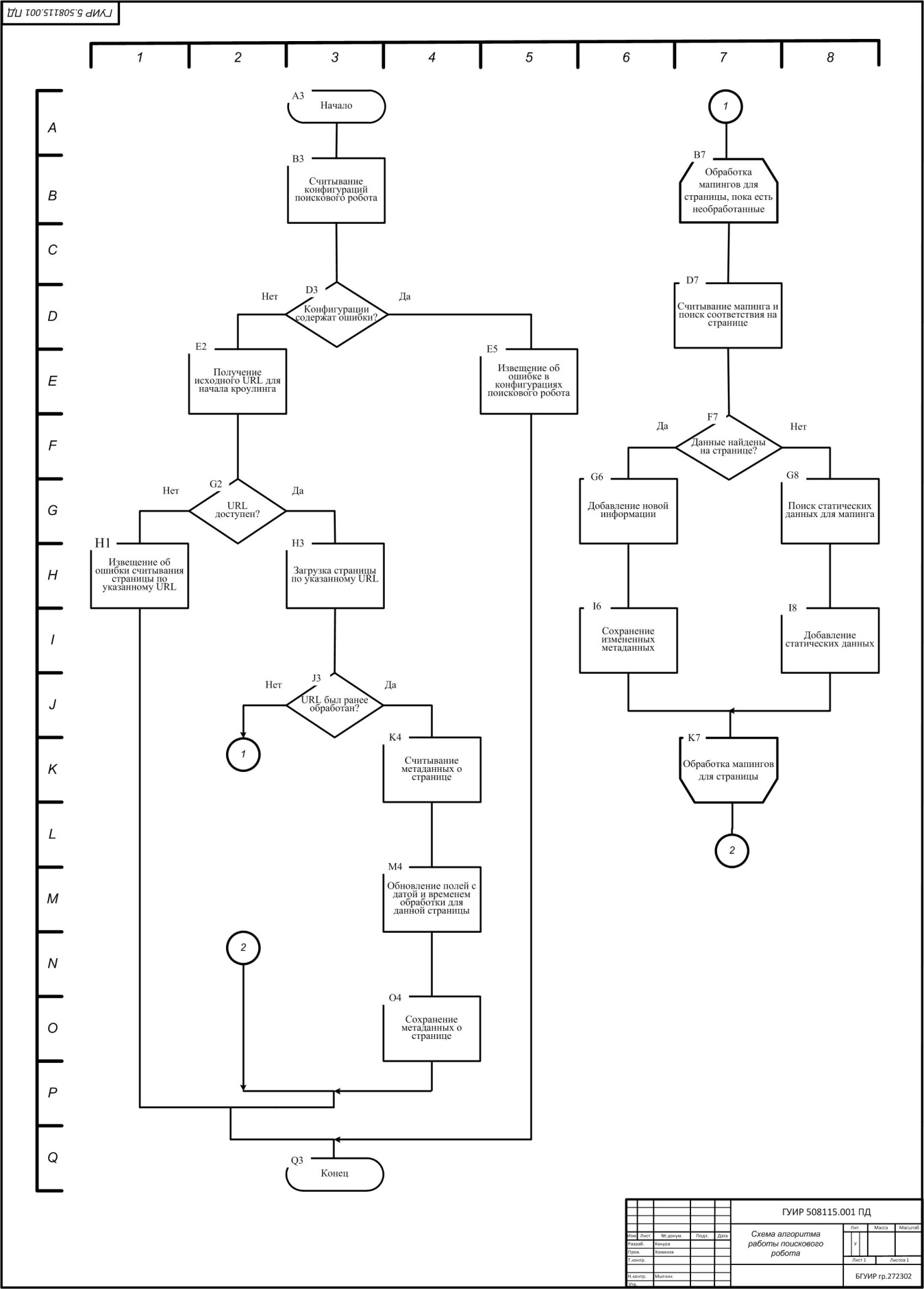


Рисунок Б.1 – Схема обобщенного алгоритма работы поискового робота при сохранении страницы, найденной поисковым роботом

# ПРИЛОЖЕНИЕ В (обязательное) Листинг основных элементов программы

package com.sap.scp.srch.api.search.controller;

/\*\*

\* REST API Controller class for the search API.

\* Handles the GET and POST requests to the search API.

\*/

@RestController

@RequestMapping("/api/v1/search")

public class SearchApiController {

private static final String APPLICATION\_EXECUTION\_TIME = "Application Execution Time";

private static final String COUNT = "count";

private static final String DEBUG = "debug";

private static final String DISTRIBUTIONS = "distributions";

private static final String FIELD = "field";

private static final String RESULTS = "results";

private static final String SUCCESS = "success";

private static final String STATUS = "status";

private static final String UNKNOWN\_ERROR = "Unknown Error";

public static final String SUPER\_ADMIN\_CLIENT = "admin\_client";

public static final double MIN\_WEIGHT = 0.0;

public static final double MAX\_WEIGHT = 1.0;

public static final double MIN\_FUZZINESS = 0.0;

public static final double MAX\_FUZZINESS = 1.0;

@Autowired private Token token;

@Autowired

NewSearchService ss;

@Autowired

DomainFactory df;

/\*\*

\* Request mapping handler method for POST requests with JSON data

Продолжение приложения В

\* to the search API.

\* @param sr the JSON POST request body data

\* @param request the HttpServletRequest

\* @param response the HttpServletResponse

\* @return REST API JSON response map

\*/

@SuppressWarnings("unchecked")

@RequestMapping(method = RequestMethod.POST, headers = "Accept=application/json", produces={"application/json; charset=UTF-8"})

@ResponseBody

public Map<String, Object> search(@RequestBody Map<String, Object> sr, HttpServletRequest request, HttpServletResponse response) {

// result

Map<String, Object> resultMap = new LinkedHashMap<>();

// status

int errors = 0;

int successes = 0;

int overallErrorCode = 0;

CaseInsensitiveMap searchRequestInsensitiveMap = new CaseInsensitiveMap(sr);

// technical debug

Boolean techDebug = Boolean.valueOf(request.getParameter("techdebug"));

/\*\*

\* count

\*/

Map<String, ?> countMap = (Map<String, ?>) searchRequestInsensitiveMap.get("returncount");

if( countMap != null) {

Stopper countApplicationExecutionTime = new Stopper(APPLICATION\_EXECUTION\_TIME).start();

CaseInsensitiveMap countMapInsensitiveMap = new CaseInsensitiveMap(countMap);

Map<String, Object> countResultMap = new LinkedHashMap<>();

DocumentCountResult documentCountResult = null;

Продолжение приложения В

try {

BaseRequest countRequest = RequestBuilder.buildCountRequest(sr, token, ApiUtil.getType(request), techDebug);

Boolean debug = (Boolean) countMapInsensitiveMap.get(DEBUG);

// execute count

documentCountResult = ss.getNumberOfDocuments(countRequest);

countResultMap.put(STATUS, SUCCESS);

if(documentCountResult != null) {

countApplicationExecutionTime.stop();

if(debug != null && debug) {

countResultMap.put("time", countApplicationExecutionTime.getElapsedTime());

}

if(techDebug != null && techDebug) {

documentCountResult.addDebugInto(getApplicationExecutionTimeInfo(countApplicationExecutionTime, documentCountResult.getDebugInfo()));

countResultMap.put(DEBUG, documentCountResult.getDebugInfo());

}

countResultMap.put(COUNT, documentCountResult.getTotalCount());

}

successes++;

} catch (Exception e) {

String errmsg = "";

String logerrmsg = "";

int errCode = 0;

if( e instanceof UncategorizedSQLException ) {

logerrmsg = e.getMessage();

errmsg = UNKNOWN\_ERROR;

}

errCode = errCode!=0?errCode:getHttpErrorCode(e);

errmsg = Str.ok(errmsg) ? errmsg : e.getMessage();

ApiError error = ApiError.create(e, getHttpErrorCode(e), "Error counting documents: " + errmsg);

error.logmessage(logerrmsg);

error.doLogIfAllowed();

countResultMap.putAll(error.getJsonResponseMap());

Продолжение приложения В

errors++;

overallErrorCode = calculateOverallErrorCode(overallErrorCode, errCode);

}

resultMap.put(COUNT, countResultMap);

}

/\*\*

\* search

\*/

Map<String, ?> searchMap = (Map<String, ?>) searchRequestInsensitiveMap.get("returnresults");

if(searchMap != null) {

Stopper searchApplicationExecutionTime = new Stopper(APPLICATION\_EXECUTION\_TIME).start();

SearchRequest searchRequest = new SearchRequest();

Map<String, Object> searchResultMap = new LinkedHashMap<>();

SearchResult searchResult = null;

try {

CaseInsensitiveMap searchInsensitiveMap = new CaseInsensitiveMap(searchMap);

searchRequest = RequestBuilder.buildSearchRequest(sr, token, ApiUtil.getType(request), techDebug);

Boolean debug = (Boolean) searchInsensitiveMap.get(DEBUG);

// execute search

searchResult = ss.search(searchRequest);

searchResultMap.put(STATUS, SUCCESS);

if(searchResult != null) {

searchApplicationExecutionTime.stop();

if(debug != null && debug) {

searchResultMap.put("time", searchApplicationExecutionTime.getElapsedTime());

}

if(techDebug != null && techDebug) {

searchResult.addDebugInto(getApplicationExecutionTimeInfo(searchApplicationExecutionTime, searchResult.getDebugInfo()));

Продолжение приложения В

searchResultMap.put(DEBUG, searchResult.getDebugInfo());

}

searchResultMap.put(RESULTS, searchResult.getSearchResult());

}

successes++;

} catch (Exception e) {

String errmsg = "";

String logerrmsg = "";

int errCode = 0;

if( e instanceof UncategorizedSQLException ) {

logerrmsg = e.getMessage();

errmsg = UNKNOWN\_ERROR;

}

if ( e instanceof ApiError ) {

errCode = ((ApiError) e).getStatus();

}

errCode = errCode!=0?errCode:getHttpErrorCode(e);

errmsg = Str.ok(errmsg) ? errmsg : e.getMessage();

ApiError error = ApiError.create(e, errCode, "Error searching documents: " + errmsg);

error.logmessage(logerrmsg);

error.doLogIfAllowed();

searchResultMap.putAll(error.getJsonResponseMap());

errors++;

overallErrorCode = calculateOverallErrorCode(overallErrorCode, errCode);

}

resultMap.put(RESULTS, searchResultMap);

}

/\*\*

\* distribution

\*/

Map<String, ?> distributionsMap = (Map<String, ?>) searchRequestInsensitiveMap.get("returndistributions");

Продолжение приложения В

if( distributionsMap != null) {

CaseInsensitiveMap distributionsInsensitiveMap = new CaseInsensitiveMap(distributionsMap);

DistributionSearchRequest distributionRequest = new DistributionSearchRequest();

// base request

Exception baseRequestException = null;

try {

setBaseRequest(distributionRequest, (List<Map<String, ?>>) searchRequestInsensitiveMap.get("filters"), request);

} catch (Exception e) {

baseRequestException = e;

}

// debug

Boolean debug = (Boolean) distributionsInsensitiveMap.get(DEBUG);

if((debug != null && debug) || (techDebug != null && techDebug)) {

distributionRequest.setDebug(true);

} else {

distributionRequest.setDebug(false);

}

// distributions

List<Map<String, ?>> distributions = (List<Map<String, ?>>) distributionsInsensitiveMap.get(DISTRIBUTIONS);

if(distributions != null) {

List<Object> distributionResults = new ArrayList<>();

for (Map<String, ?> distribution : distributions) {

Stopper distributionApplicationExecutionTime = new Stopper(APPLICATION\_EXECUTION\_TIME).start();

CaseInsensitiveMap distInsensitiveMap = new CaseInsensitiveMap(distribution);

Map<String, Object> distributionMap = new LinkedHashMap<>();

DistributionSearchResult distributionSearchResult = null;

try {

if(baseRequestException != null) {

Продолжение приложения В

throw baseRequestException;

}

String distributionFilter = toUppercase((String) distInsensitiveMap.get(FIELD));

distributionRequest.setDistributionFilter(distributionFilter);

Object distributionLimit = distInsensitiveMap.get("limit");

if(distributionLimit != null) {

distributionRequest.setNumberOfValuesPerFilter(getAndValidateInteger("Distribution Limit", distributionLimit));

}

// execute distribution

distributionSearchResult = ss.getDistribution(distributionRequest);

distributionMap.put(STATUS, SUCCESS);

if(distributionSearchResult != null) {

distributionApplicationExecutionTime.stop();

if(debug != null && debug) {

distributionMap.put("time", distributionApplicationExecutionTime.getElapsedTime());

}

if(techDebug != null && techDebug) {

distributionSearchResult.addDebugInto(getApplicationExecutionTimeInfo(distributionApplicationExecutionTime, distributionSearchResult.getDebugInfo()));

distributionMap.put(DEBUG, distributionSearchResult.getDebugInfo());

}

List<Filter> distributionResult = distributionSearchResult.getDistributionSearchResult();

if(distributionResult != null) {

distributionMap.put(FIELD, distributionFilter);

distributionMap.put(DISTRIBUTIONS, convertDistributions(distributionResult));

}

}

successes++;

} catch (Exception e) {

String errmsg = "";

Продолжение приложения В

String logerrmsg = "";

int errCode = 0;

if( e instanceof UncategorizedSQLException ) {

logerrmsg = e.getMessage();

errmsg = UNKNOWN\_ERROR;

}

errCode = errCode!=0?errCode:getHttpErrorCode(e);

errmsg = Str.ok(errmsg) ? errmsg : e.getMessage();

ApiError error = ApiError.create(e, errCode, "Error getting distributions: " + errmsg);

error.logmessage(logerrmsg);

error.doLogIfAllowed();

distributionMap.putAll(error.getJsonResponseMap());

errors++;

overallErrorCode = calculateOverallErrorCode(overallErrorCode, errCode);

}

distributionResults.add(distributionMap);

}

resultMap.put(DISTRIBUTIONS, distributionResults);

}

}

/\*\*

\* global result

\*/

Map<String, Object> globalResultMap = new LinkedHashMap<>();

if ((countMap != null || searchMap != null || distributionsMap != null) && successes == 0) {

if (overallErrorCode == 0) { // if there are no successes and no error code was set, then set 500!

overallErrorCode = 500;

}

globalResultMap.putAll(ApiError.create(overallErrorCode, "All requests failed. Please check individual errors below.").getJsonResponseMap());

response.setStatus(overallErrorCode);

} else {

if (errors > 0) {

globalResultMap.put(STATUS, "partial");

Продолжение приложения В

} else {

globalResultMap.put(STATUS, SUCCESS);

}

}

globalResultMap.put("result", resultMap);

return globalResultMap;

}

private int calculateOverallErrorCode(int overallErrorCode, int subQueryErrorCode) {

if (overallErrorCode > 0 && subQueryErrorCode != overallErrorCode) {

return 500; // different sub-query error codes

} else {

return subQueryErrorCode;

}

}

private List<Map<String, Object>> convertDistributions(List<Filter> distributionResult) {

if(distributionResult == null) {

return Collections.emptyList();

} else {

List<Map<String, Object>> distributions = new ArrayList<>();

for(Filter filter : distributionResult) {

Map<String, Object> distributionMap = new LinkedHashMap<>();

distributionMap.put("value", filter.getFilterValue());

distributionMap.put(COUNT, filter.getCount());

distributions.add(distributionMap);

}

return distributions;

}

}

private long getHanaExecutionTime(List<DebugInfo> debugInfo) {

long elapsedTime = 0;

if(debugInfo != null) {

for(DebugInfo di : debugInfo) {

if("SQL Query".equalsIgnoreCase(di.getTitle()) || "Retrieved From Cache".equalsIgnoreCase(di.getTitle())) {

elapsedTime = di.getStopper().getElapsedTime();

}

}

}

Продолжение приложения В

return elapsedTime;

}

private String toUppercase(String str) {

if(str != null) {

return str.toUpperCase();

} else {

return str;

}

}

private void setBaseRequest(BaseRequest backendSearchRequest, List<Map<String, ?>> filters, HttpServletRequest request) {

// client

backendSearchRequest.setClient(token.getClient());

// repository

if(token.getRepository() != null) {

backendSearchRequest.setRepository(token.getRepository().toUpperCase());

}

// type

String type = ApiUtil.getType(request);

if(type != null) {

backendSearchRequest.setType(NamingUtil.canonicalizeType(type));

}

Map<String,ArrayList<String>> specialCharacters;

HashMap<String,ArrayList<String>> specialCharacterParameters;

// filters

if(filters != null && !filters.isEmpty()) {

Map<String, ?> filterInLastIndex = filters.get(filters.size() - 1);

if(filterInLastIndex.get(RequestBuilder.LOGICAL\_OPERATOR) != null){

filterInLastIndex.remove(RequestBuilder.LOGICAL\_OPERATOR);

filters.remove(filters.size() -1);

filters.add(filters.size(), filterInLastIndex);

}

List<SearchField> searchFields = new ArrayList<>();

for (Map<String, ?> filter : filters) {

Продолжение приложения В

specialCharacterParameters = new HashMap<>();

String filterName = (String)filter.get("type");

if(filter.get(RequestBuilder.SPECIAL\_CHARACTERS) != null && (filterName.equalsIgnoreCase(Type.FUZZY.name()) || filterName.equalsIgnoreCase(Type.EXACT.name()))){

specialCharacters = (Map<String, ArrayList<String>>) filter.get(RequestBuilder.SPECIAL\_CHARACTERS);

ArrayList<String> removeArr = specialCharacters.get(RequestBuilder.REMOVE\_OBJECT\_NAME);

ArrayList<String> escapeArr = specialCharacters.get(RequestBuilder.ESCAPE\_OBJECT\_NAME);

specialCharacterParameters.put(RequestBuilder.REMOVE\_OBJECT\_NAME,removeArr);

specialCharacterParameters.put(RequestBuilder.ESCAPE\_OBJECT\_NAME,escapeArr);

}

SearchField searchField = RequestBuilder.buildSearchFilterFromMap(filter,specialCharacterParameters);

searchFields.add(searchField);

}

backendSearchRequest.setSearchFields(searchFields);

}

}

private int getAndValidateInteger(String parameterName, Object integerValue) {

Integer value;

if (integerValue instanceof Integer) {

value = (Integer) integerValue;

} else {

try {

String temp = String.valueOf(integerValue);

value = Integer.parseInt(temp);

} catch (NumberFormatException e) {

throw new IllegalArgumentException("Parameter " + parameterName + " must be an integer value.");

}

}

return value.intValue();

}

Продолжение приложения В

/\*\*

\* Request mapping handler method for GET requests with request parameters

\* to the search API (URL search).

\* @param outputFields list of output fields to return (separated by colon)

\* @param page result page number

\* @param pageSize result page size (max. number of results per page)

\* @param sort sort settings (<sort attribute>:<sort direction>)

\* @param snippet snippet field

\* @param boost boost settings (<boost field>:<boost limit>)

\* @param distribution distribution settings (<distribution attribute>:<max. number of results>)

\* @param filter list of search filters

\* @param debug debug flag

\* @param techdebug technical debug flag

\* @param returnResults return results flag

\* @param returnCount return count flag

\* @param returnDistribution return distribution flag

\* @param request HttpServletRequest

\* @param response HttpServletResponse

\* @return REST API JSON response map

\*/

@RequestMapping(method = RequestMethod.GET)

@ResponseBody

public Map<String, Object> search(@RequestParam(required = false) String outputFields,

@RequestParam(required = false) Integer page,

@RequestParam(required = false) Integer pageSize,

@RequestParam(required = false) String sort,

@RequestParam(required = false) String snippet,

@RequestParam(required = false) String boost,

@RequestParam(required = false) String distribution,

@RequestParam(required = false) String[] filter,

@RequestParam(required = false, defaultValue = "false") Boolean debug,

Продолжение приложения В

@RequestParam(required = false, defaultValue = "false") Boolean techdebug,

@RequestParam(required = false, defaultValue = "false") Boolean returnResults,

@RequestParam(required = false, defaultValue = "false") Boolean returnCount,

@RequestParam(required = false, defaultValue = "false") Boolean returnDistribution,

HttpServletRequest request,

HttpServletResponse response) {

Map<String, Object> resultMap = new LinkedHashMap<>();

// status

int errors = 0;

int successes = 0;

/\*\*

\* count

\*/

if(returnCount != null && returnCount) {

Stopper countApplicationExecutionTime = new Stopper(APPLICATION\_EXECUTION\_TIME).start();

BaseRequest baseRequest = new BaseRequest();

// execute count

Map<String, Object> countResultMap = new LinkedHashMap<>();

DocumentCountResult documentCountResult = null;

try {

baseRequest = RequestBuilder.buildGetCountRequest(filter, token, ApiUtil.getType(request), debug, techdebug);

documentCountResult = ss.getNumberOfDocuments(baseRequest);

countResultMap.put(STATUS, SUCCESS);

if(documentCountResult != null) {

countApplicationExecutionTime.stop();

if(debug != null && debug) {

Продолжение приложения В

countResultMap.put("time", countApplicationExecutionTime.getElapsedTime());

}

if(techdebug != null && techdebug) {

documentCountResult.addDebugInto(getApplicationExecutionTimeInfo(countApplicationExecutionTime, documentCountResult.getDebugInfo()));

countResultMap.put(DEBUG, documentCountResult.getDebugInfo());

}

countResultMap.put(COUNT, documentCountResult.getTotalCount());

}

successes++;

} catch (Exception e) {

String errmsg = "";

String logerrmsg = "";

int errCode = 0;

if( e instanceof UncategorizedSQLException ) {

logerrmsg = e.getMessage();

errmsg = UNKNOWN\_ERROR;

}

errCode = errCode!=0?errCode:getHttpErrorCode(e);

errmsg = Str.ok(errmsg) ? errmsg : e.getMessage();

ApiError error = ApiError.create(e, errCode, "Error counting documents: " + errmsg);

error.logmessage(logerrmsg);

error.doLogIfAllowed();

countResultMap.putAll(error.getJsonResponseMap());

errors++;

}

resultMap.put(COUNT, countResultMap);

}

/\*\*

\* search

\*/

if(returnResults != null && returnResults) {

Продолжение приложения В

Stopper searchApplicationExecutionTime = new Stopper(APPLICATION\_EXECUTION\_TIME).start();

SearchRequest searchRequest = new SearchRequest();

Map<String, Object> searchResultMap = new LinkedHashMap<>();

SearchResult searchResult = null;

try {

SearchParameters params = new SearchParameters(outputFields, page, pageSize, sort, snippet, boost, filter);

searchRequest = RequestBuilder.buildGetSearchRequest(params, token, ApiUtil.getType(request), debug, techdebug);

// execute search

searchResult = ss.search(searchRequest);

searchResultMap.put(STATUS, SUCCESS);

if(searchResult != null) {

searchApplicationExecutionTime.stop();

if(debug != null && debug) {

searchResultMap.put("time", searchApplicationExecutionTime.getElapsedTime());

}

if(techdebug != null && techdebug) {

searchResult.addDebugInto(getApplicationExecutionTimeInfo(searchApplicationExecutionTime, searchResult.getDebugInfo()));

searchResultMap.put(DEBUG, searchResult.getDebugInfo());

}

searchResultMap.put(RESULTS, searchResult.getSearchResult());

}

successes++;

} catch (Exception e) {

String errmsg = "";

String logerrmsg = "";

int errCode = 0;

if( e instanceof UncategorizedSQLException ) {

logerrmsg = e.getMessage();

errmsg = UNKNOWN\_ERROR;

}

Продолжение приложения В

if ( e instanceof ApiError ) {

errCode = ((ApiError) e).getStatus();

}

errCode = errCode!=0?errCode:getHttpErrorCode(e);

errmsg = Str.ok(errmsg) ? errmsg : e.getMessage();

ApiError error = ApiError.create(e, errCode, "Error searching documents: " + errmsg);

error.logmessage(logerrmsg);

error.doLogIfAllowed();

searchResultMap.putAll(error.getJsonResponseMap());

errors++;

}

resultMap.put(RESULTS, searchResultMap);

}

/\*\*

\* distribution

\*/

if(returnDistribution != null && returnDistribution && distribution != null) {

Stopper distributionApplicationExecutionTime = new Stopper(APPLICATION\_EXECUTION\_TIME).start();

DistributionSearchRequest distributionRequest = new DistributionSearchRequest();

Map<String, Object> distributionMap = new LinkedHashMap<>();

DistributionSearchResult distributionSearchResult = null;

// distributions

List<Object> distributionResults = new ArrayList<>();

try {

distributionRequest = RequestBuilder.buildGetDistributionSearchRequest(distribution, filter, token, ApiUtil.getType(request), debug, techdebug);

// execute distribution

distributionSearchResult = ss.getDistribution(distributionRequest);

Продолжение приложения В

distributionMap.put(STATUS, SUCCESS);

if(distributionSearchResult != null) {

distributionApplicationExecutionTime.stop();

if(debug != null && debug) {

distributionMap.put("time", distributionApplicationExecutionTime.getElapsedTime());

}

if(techdebug != null && techdebug) {

distributionSearchResult.addDebugInto(getApplicationExecutionTimeInfo(distributionApplicationExecutionTime, distributionSearchResult.getDebugInfo()));

distributionMap.put(DEBUG, distributionSearchResult.getDebugInfo());

}

List<Filter> distributionResult = distributionSearchResult.getDistributionSearchResult();

if(distributionResult != null) {

distributionMap.put(FIELD, distributionRequest.getDistributionFilter());

distributionMap.put(DISTRIBUTIONS, convertDistributions(distributionResult));

}

}

successes++;

} catch (Exception e) {

String errmsg = "";

String logerrmsg = "";

int errCode = 0;

if( e instanceof UncategorizedSQLException ) {

logerrmsg = e.getMessage();

errmsg = UNKNOWN\_ERROR;

}

errCode = errCode!=0?errCode:getHttpErrorCode(e);

errmsg = Str.ok(errmsg) ? errmsg : e.getMessage();

ApiError error = ApiError.create(e, errCode, "Error getting distributions: " + errmsg);

error.logmessage(logerrmsg);

error.doLogIfAllowed();

distributionMap.putAll(error.getJsonResponseMap());

errors++;

Продолжение приложения В

}

distributionResults.add(distributionMap);

resultMap.put(DISTRIBUTIONS, distributionResults);

}

/\*\*

\* global result

\*/

Map<String, Object> globalResultMap = new LinkedHashMap<>();

if(successes == 0) {

globalResultMap.putAll(ApiError.create(500, "All requests failed. Please check individual errors below.").getJsonResponseMap());

response.setStatus(500);

} else {

if (errors > 0) {

globalResultMap.put(STATUS, "partial");

} else {

globalResultMap.put(STATUS, SUCCESS);

}

}

globalResultMap.put("result", resultMap);

return globalResultMap;

}

private int getHttpErrorCode(Throwable throwable) {

// in future extend here the mapping of exceptions to http error codes

return (throwable instanceof IllegalArgumentException) ? 400 : 500;

}

private DebugInfo getApplicationExecutionTimeInfo(Stopper applicationExecutionTime, List<DebugInfo> debugInfo) {

DebugInfo applicationExecutionTimeInfo = new DebugInfo(APPLICATION\_EXECUTION\_TIME);

long hanaExecutionTime = getHanaExecutionTime(debugInfo);

long noneHanaExecutionTime = applicationExecutionTime.getElapsedTime() - hanaExecutionTime;

Продолжение приложения В

applicationExecutionTimeInfo.setStopper(applicationExecutionTime);

applicationExecutionTimeInfo.setData("Execution Time inside Hana: " + hanaExecutionTime + "ms. Execution Time Outside Hana: " + noneHanaExecutionTime + "ms.");

return applicationExecutionTimeInfo;

}

}

package com.sap.scp.srch.api.search.service;

@Service

public class NewSearchService {

@Autowired

private NewSearchRepository searchRepository;

@Autowired

private RepositoryTypeService repositoryTypeService;

@Autowired

private RepositoryTypeHelper typehelper;

@Autowired

private RepositoryService repositoryService;

public static DateFormat DATE\_FORMAT = new SimpleDateFormat("yyyy-MM-dd");

public static DateFormat TIME\_FORMAT = new SimpleDateFormat("HH:mm:ss");

public static DateFormat TIMESTAMP\_FORMAT = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

public static final int MAX\_PAGE\_SIZE = 600;

public static final int MAX\_NUMBER\_OF\_VALUES\_PER\_FILTER = 50;

public static final int MAX\_SEARCH\_FIELD\_VALUE\_SIZE = 1000;

public static final int MAX\_BOOSTING\_LIMIT = 1000;

public static final List<String> SPECIAL\_FIELDS = Arrays.asList("FACTOR", "SCORE", "BOOST", "SNIPPET");

/\*\*

\* main

Продолжение приложения В

\*/

@Autowired

private Token token;

/\*\*

\* Checks if the automaticSearchFilterException is true or not

\* @param automaticSearchFilterException

\* @return

\*/

public boolean isAutomaticSearchFilterExceptionTrue(String automaticSearchFilterException){

if (automaticSearchFilterException == null || automaticSearchFilterException.isEmpty()) {

return false;

}

return LogicalExpression.parse(automaticSearchFilterException).evaluate(token);

}

@SuppressWarnings("unchecked")

public SearchResult search(SearchRequest searchRequest) throws DataAccessException, IllegalArgumentException {

List<DebugInfo> changedSearchFields = new ArrayList<>();

String repository = searchRequest.getRepository();

String type = searchRequest.getType();

List<SearchField> searchFields = searchRequest.getSearchFields();

searchFields = this.addAutomaticSearchFilter(repository, type, searchFields);

// validate search fields + mask multi-values + include client if not full permission

List<SearchField> extendedSearchFields = getExtendedSearchFields(searchRequest.getRepository(), searchRequest.getType(), searchFields, changedSearchFields);

// search

NewSearchRepository.SearchSqlParameters searchSqlParameters =

searchRepository.createSearchSqlParameters(repository, type, extendedSearchFields);

Продолжение приложения В

SearchSettings searchSettings = searchRepository.createSearchSettings(searchRequest.getPage(),

validatePageSize(searchRequest.getPageSize()),

validateAndFormatOrderByField(repository, type, searchRequest.getOrderBy()),

validateAndFormatOutputFields(repository, type, searchRequest.getOutputFields()),

validateSnippetField(repository, type, searchRequest.getSnippetField()),

validateBoostingField(repository, type, searchRequest.getBoostingField()),

validateBoostingLimit(searchRequest.getBoostingLimit()));

SearchResult result = searchRepository.search(searchSqlParameters,

searchRequest.isDebug(),

searchSettings,

searchRequest.getParameters());

// handle debug information

if (searchRequest.isDebug() && !changedSearchFields.isEmpty() && result != null) {

result.addDebugInto(changedSearchFields);

}

return convertSearchResults(repository, type, result);

}

private SearchResult convertSearchResults(String repository, String type, SearchResult result) {

if(result != null && result.getSearchResult() != null) {

List<Map<String, Object>> originalResults = result.getSearchResult();

Продолжение приложения В

convertedResult.put(name, repositoryClient.getClientName());

} else {

switch(fieldInfo.getDataType()) {

case DATE:

convertedResult.put(name, DATE\_FORMAT.format( (Date) value ));

break;

case TIME:

convertedResult.put(name, TIME\_FORMAT.format( (Time) value ));

break;

case TIMESTAMP:

convertedResult.put(name, TIMESTAMP\_FORMAT.format( (Timestamp) value ));

break;

case BOOLEAN:

convertedResult.put(name, ((Number) value).intValue() == 1);

break;

default:

convertedResult.put(name, value);

break;

}

}

}

}

}

convertedResults.add(convertedResult);

}

result.setSearchResult(convertedResults);

return result;

} else {

return null;

}

}

public DocumentCountResult getNumberOfDocuments(BaseRequest searchRequest) throws DataAccessException, IllegalArgumentException {

Продолжение приложения В

List<DebugInfo> changedSearchFields = new ArrayList<DebugInfo>();

String repository = searchRequest.getRepository();

String type = searchRequest.getType();

List<SearchField> searchFields = searchRequest.getSearchFields();

searchFields = this.addAutomaticSearchFilter(repository, type, searchFields);

// validates search fields + mask multi-values + include client if not full permission

List<SearchField> extendedSearchFields = getExtendedSearchFields(searchRequest.getRepository(), searchRequest.getType(), searchFields, changedSearchFields);

// search

DocumentCountResult docCountResult = searchRepository.getNumberOfDocuments( searchRequest.getRepository(),

searchRequest.getType(),

extendedSearchFields,

searchRequest.isDebug());

// handle debug information

if (searchRequest.isDebug() && !changedSearchFields.isEmpty() && docCountResult != null) {

docCountResult.addDebugInto(changedSearchFields);

}

return docCountResult;

}

public DistributionSearchResult getDistribution(DistributionSearchRequest searchRequest) throws IllegalArgumentException {

List<DebugInfo> changedSearchFields = new ArrayList<DebugInfo>();

String repository = searchRequest.getRepository();

String type = searchRequest.getType();

Продолжение приложения В

String distributionFilter = searchRequest.getDistributionFilter();

// type is mandatory

if (type == null) {

throw new IllegalArgumentException("Type is mandatory and can not be null in getDistribution.");

}

// distributionFilter is mandatory

if(distributionFilter == null) {

throw new IllegalArgumentException("DistributionField is mandatory and can not be null in getDistribution.");

}

//automatic Search Filter stuff

List<SearchField> searchFields = searchRequest.getSearchFields();

searchFields = this.addAutomaticSearchFilter(repository, type, searchFields);

// validates search fields + mask multi-values + include client if not full permission

List<SearchField> extendedSearchFields = getExtendedSearchFields(repository, type, searchFields, changedSearchFields);

// validates repository, type and distribution field

Field distributionFieldInfo = typehelper.getFieldInfo(repository, type, distributionFilter);

// validate field

if (DataType.BINARY == distributionFieldInfo.getDataType() || DataType.TEXT == distributionFieldInfo.getDataType() || DataType.BINTEXT == distributionFieldInfo.getDataType()) {

throw new IllegalArgumentException("Sorting not supported on field " + distributionFieldInfo.getFieldName() + ". Sorting can not be used on " + distributionFieldInfo.getDataType() + " fields.");

}

// distribution search

DistributionSearchResult distributionResult = searchRepository.getDistribution( repository,

Продолжение приложения В

validateNumberOfValuesPerFilter(searchRequest.getNumberOfValuesPerFilter()), searchRequest.isDebug());

// handle debug information

if (searchRequest.isDebug() && !changedSearchFields.isEmpty() && distributionResult != null) {

distributionResult.addDebugInto(changedSearchFields);

}

return distributionResult;

}

/\*\*

\* validation + utilities

\*/

private String validateAndFormatOutputFields(String repository, String type, List<String> outputFields) throws IllegalArgumentException {

String formattedOutputFields;

if (outputFields == null || outputFields.size() == 0) {

formattedOutputFields = "\*";

} else {

formattedOutputFields = "";

int numberOfFields = outputFields.size();

for (int i = 0; i < numberOfFields; i++) {

String currField = outputFields.get(i);

if (currField.equals("\*")) {

formattedOutputFields = "\*";

break;

} else {

// validates field

Field fieldInfo = typehelper.getFieldInfo(repository, type, currField);

// formats

formattedOutputFields += fieldInfo.getDbName();

if (i < numberOfFields - 1) {

Продолжение приложения В

formattedOutputFields += ", ";

}

}

}

}

return formattedOutputFields;

}

private String validateAndFormatOrderByField(String repository, String type, List<String> orderByFields) throws IllegalArgumentException {

String formattedOrderByFields = "";

if (orderByFields != null) {

for(String orderByField : orderByFields) {

String[] orderByElements = orderByField.split(" ");

String sortingDirection = "DESC";

if (orderByElements.length == 2) {

sortingDirection = orderByElements[1];

}

if( sortingDirection.equalsIgnoreCase("ASC") || sortingDirection.equalsIgnoreCase("DESC")) {

if(!formattedOrderByFields.equals("")) {

formattedOrderByFields += ", ";

}

if(orderByElements[0].equalsIgnoreCase("score") || orderByElements[0].equalsIgnoreCase("boost")) {

formattedOrderByFields += orderByElements[0].toUpperCase() + " " + sortingDirection;

} else {

// validates field

Field fieldInfo = typehelper.getFieldInfo(repository, type, orderByElements[0]);

if (DataType.BINARY == fieldInfo.getDataType() || DataType.TEXT == fieldInfo.getDataType() || DataType.BINTEXT == fieldInfo.getDataType()) {

throw new IllegalArgumentException("Sorting not supported on field " + fieldInfo.getFieldName() + ". Sorting can not be used on " + fieldInfo.getDataType() + " fields.");

} else if (fieldInfo.isMultiValued() || fieldInfo.getFieldName().equalsIgnoreCase("CLIENT")) {

Продолжение приложения В

throw new IllegalArgumentException("Snippet Field: " + snippetField + " must be a full text index field or of type TEXT.");

}

}

}

private int validateNumberOfValuesPerFilter(int numberOfValuesPerFilter) throws IllegalArgumentException {

if (numberOfValuesPerFilter > MAX\_NUMBER\_OF\_VALUES\_PER\_FILTER) {

throw new IllegalArgumentException("Number of values per filter exceeded allowed limit: " + numberOfValuesPerFilter + " . Maximum value allowed is: " + MAX\_NUMBER\_OF\_VALUES\_PER\_FILTER);

} else {

return numberOfValuesPerFilter;

}

}

private int validatePageSize(int pageSize) throws IllegalArgumentException {

if (pageSize > MAX\_PAGE\_SIZE) {

throw new IllegalArgumentException("Page size exceeded allowed limit: " + pageSize + " . Maximum value allowed is: " + MAX\_PAGE\_SIZE);

} else {

return pageSize;

}

}

private int validateBoostingLimit(int boostingLimit) throws IllegalArgumentException {

if (boostingLimit > MAX\_BOOSTING\_LIMIT) {

throw new IllegalArgumentException("Boosting limit exceeded allowed limit: " + boostingLimit + " . Maximum value allowed is: " + MAX\_BOOSTING\_LIMIT);

} else {

return boostingLimit;

}

}

Продолжение приложения В

private String validateSearchFieldValue(Field fieldInfo , String searchFieldValue ) throws IllegalArgumentException {

if ((searchFieldValue == null || searchFieldValue.length() > MAX\_SEARCH\_FIELD\_VALUE\_SIZE) ) {

throw new IllegalArgumentException("Illegal search field value: " + searchFieldValue + ". Null is not permitted and maximum allowed size is: " + MAX\_SEARCH\_FIELD\_VALUE\_SIZE);

} else {

if (fieldInfo != null && fieldInfo.isCaseInsensitive()){

return searchFieldValue.toLowerCase();

}

return searchFieldValue;

}

}

private String validateAndConvertClientValue(String repository, String clientName) throws IllegalArgumentException {

RepositoryClient repositoryClient = repositoryService.findRepositoryClientByName(clientName, repository);

if (repositoryClient == null) {

throw new IllegalArgumentException("Client name " + clientName + " does not exist.");

} else {

return String.valueOf(repositoryClient.getClientId());

}

}

private void validateSearchField(SearchField searchField, Field fieldInfo) throws IllegalArgumentException {

switch (searchField.getType()) {

case FUZZY:

if(!fieldInfo.isSearchable() && !fieldInfo.getDataType().isTypeBinaryText()) {

throw new IllegalArgumentException("FUZZY filter only allowed on fields with full text index. Search Field " + searchField.getName() + " does not have full text index.");

}

break;

case EXACT:

Продолжение приложения В

if(fieldInfo.isMultiValued() && !fieldInfo.isSearchable()) {

throw new IllegalArgumentException("EXACT filter not allowed on multivalue fields without full text index. Search Field " + searchField.getName() + " is multivalue and does not have a full text index.");

}

break;

case LIKE:

if(fieldInfo.getDataType().isTypeBinaryText()) {

throw new IllegalArgumentException("LIKE filter not allowed on BINTEXT datatype.");

}

break;

case IN:

if(fieldInfo.getDataType().isTypeBinaryText()) {

throw new IllegalArgumentException("IN filter not allowed on BINTEXT datatype.");

}

if(fieldInfo.isMultiValued()) {

throw new IllegalArgumentException("IN filter not allowed on multivalue fields. Search Field " + searchField.getName() + " is multivalue.");

}

break;

case OPERATOR:

if(fieldInfo.getDataType().isTypeBinaryText()) {

throw new IllegalArgumentException("OPERATOR filter not allowed on BINTEXT datatype.");

}

if(fieldInfo.isMultiValued() && (Operator.IS\_NOT\_NULL != searchField.getOperator() && Operator.IS\_NULL != searchField.getOperator())) {

throw new IllegalArgumentException("OPERATOR filter not allowed on multivalue fields. Search Field " + searchField.getName() + " is multivalue.");

}

break;

}

}

private Type getDefaultSearchType(Field fieldInfo) {

Продолжение приложения В

boolean singleOperator = searchField.getOperator().isSingleOperator();

if((searchField.getValues() == null || searchField.getValues().isEmpty() ) && singleOperator == false) {

throw new IllegalArgumentException("SearchField values can not be null or empty.");

}

if(searchField.getValues()!=null && (searchField.getOperator().toString().equalsIgnoreCase(Operator.IS\_NOT\_NULL.toString()) || searchField.getOperator().toString().equalsIgnoreCase(Operator.IS\_NULL.toString()))){

throw new IllegalArgumentException("Depricated to use SearchField values with FilterOperator equal "+searchField.getOperator().toString());

}

// validates repository, type and field

Field fieldInfo = typehelper.getFieldInfo(repository, type, searchField.getName());

if(searchField.getType() == null) {

// set default search type according to field info

searchField.setType(getDefaultSearchType(fieldInfo));

} else {

// validate search type against underlying field

validateSearchField(searchField, fieldInfo);

}

extendedSearchField.setName(fieldInfo.getDbName());

if (singleOperator == false){

if (fieldInfo.isMultiValued()) {

// only one value supported for multi-value fields

if (searchField.getType() == Type.LIKE) {

extendedSearchField.setValue("%~" + validateSearchFieldValue(fieldInfo,searchField.getValue()) + "~%");

} else {

if (singleOperator == false){

extendedSearchField.setValue(validateSearchFieldValue(fieldInfo,searchField.getValue()));

}

}

} else {

Продолжение приложения В

List<String> validatedValues = new ArrayList<String>();

if(searchField.getName().equalsIgnoreCase("CLIENT")) {

validatedValues.add(validateAndConvertClientValue(repository,searchField.getValue()));

} else {

// maybe we should also limit the number of values allowed ???

for(String value: searchField.getValues()) {

validatedValues.add(validateSearchFieldValue(fieldInfo,value));

}

}

extendedSearchField.setValues(validatedValues);

}

}

// operator

extendedSearchField.setOperator(searchField.getOperator());

// type

if(searchField.getType() == Type.FUZZY && (StringUtils.countOccurrencesOf(searchField.getValue(), "\"") > 1 || Str.numericPercent(searchField.getValue()) > 60 )) {

// change search type from fuzzy to exact if more than one quote is found in value

extendedSearchField.setType(Type.EXACT);

// create corresponding debug info

DebugInfo chengedSearchFieldInfo = new DebugInfo("Changed Filter Information");

chengedSearchFieldInfo.setData("FUZZY filter [" + searchField.getName() + "] with value [" + extendedSearchField.getValue() + "] was changed to EXACT filter with same value.");

changedSearchFields.add(chengedSearchFieldInfo);

} else {

extendedSearchField.setType(searchField.getType());

}

// fuzzyness

extendedSearchField.setFuzzyLevel(searchField.getFuzzyLevel());

Продолжение приложения В

extendedSearchField.setFuzzyParameters(searchField.getFuzzyParameters()); // not validated

// weight

extendedSearchField.setWeight(searchField.getWeight());

// validates logical operator

extendedSearchField.setLogicalOperator(searchField.getLogicalOperator());

extendedSearchFields.add(extendedSearchField);

}

}

return extendedSearchFields;

}

private List<SearchField> addAutomaticSearchFilter(String repository, String type, List<SearchField> origSearchFields) {

List<SearchField> searchFields = origSearchFields;

if (searchFields == null) {

searchFields = new ArrayList<>();

}

com.sap.scp.srch.api.repository.domain.Type typeObj = repositoryTypeService.getType(type, repository);

List<Field> fieldList = typeObj.getFields();

String automaticSearchFilter;

String automaticSearchFilterException;

for (int i = 0; i < fieldList.size(); i++) {

Field f = fieldList.get(i);

if (f.hasAutomaticSearchFilter()) {

automaticSearchFilter = f.getAutomaticSearchFilter();

if (f.hasAutomaticSearchFilterException()) {

automaticSearchFilterException = f.getAutomaticSearchFilterException();

} else {

automaticSearchFilterException = null;

}

if (!f.hasAutomaticSearchFilterException() || !isAutomaticSearchFilterExceptionTrue(automaticSearchFilterException)) {

if (automaticSearchFilter.startsWith("$")) {

Продолжение приложения В

Object data = token.getUserInfo().get(automaticSearchFilter.substring(1));

if (data instanceof String) {

SearchField searchField = new SearchField();

searchField.setName(f.getFieldName());

searchField.setValue((String) data);

searchField.setType(SearchField.Type.EXACT);

searchFields.add(searchField);

} else if (data instanceof Collection) {

SearchField searchField = new SearchField();

for(String parameter :(Collection<String>) data){

searchField = new SearchField();

searchField.setName(f.getFieldName());

searchField.setType(SearchField.Type.EXACT);

searchField.setLogicalOperator(SearchField.LogicalOperator.OR);

searchField.setValue(parameter);

searchFields.add(searchField);

}

}else{

SearchField searchField = new SearchField();

searchField.setName(f.getFieldName());

searchField.setType(SearchField.Type.EXACT);

searchField.setLogicalOperator(SearchField.LogicalOperator.OR);

searchField.setValue("p");

searchFields.add(searchField);

}

} else {

SearchField searchField = new SearchField();

searchField.setName(f.getFieldName());

searchField.setValue(automaticSearchFilter);

searchField.setType(SearchField.Type.EXACT);

searchFields.add(searchField);

}

}

}

}

return searchFields;

}

}