Министерство образования и науки Российской Федерации
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет информатики
Кафедра теоретических основ информатики

УДК 681.03

ДОПУСТИТЬ К ЗАЩИТЕ В ГАК
Зав. кафедрой, доц., к.т.н.
_____ А. Л. Фукс
«___» _____ 2012 г.

ДИПЛОМНАЯ РАБОТА

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ EYE ON FILE ONLINE НА БАЗЕ VDOM ТЕХНОЛОГИЙ ДЛЯ ОРГАНИЗАЦИИ АРХИВОВ ДОКУМЕНТОВ

Андреева Алексея Петровича

Научный руководитель
асс. каф. ТОИ
_____М. А. Филонов
Исполнитель студ. Гр. 1471
_____ А.П. Андреев

Томск – 2012

РЕФЕРАТ

Дипломная работа, 9 рис.,10 источников, 1 приложение.

ПОИСКОВАЯ СИСТЕМА, WEB-ПРИЛОЖЕНИЕ, XAPIAN, PYTHON, VDOM ТЕХНОЛОГИИ, XML, WEB, APY, БАЗА ДАННЫХ.

Цель: разработка web-приложения Eye on file online на базе VDOM-технологий для организации архивов документов, с возможностью полнотекстового поиска.

Результат работы: разработана реляционная БД, слой бизнес-логики, web-интерфейс, а также API приложения, для интеграции со сторонними приложениями. Приложение принято заказчиком, внедрено и имеет дальнейшее развитие.

Область применения: для предприятий малого и среднего бизнеса, которые нуждаются в электронном ведении документа оборота с возможностью доступа через WEB.

Оглавление

ВВЕДЕНИЕ

Всемирная паутина (www) является основным и самым актуальным направлением развития Интернет-технологий и представляет собой мощный источник информации. Особую актуальность это направление приобретает в России – всѐ больше предприятий осознают удобство такого способа общения с клиентами и продвижения товаров и услуг. Для предприятий малого бизнеса создание собственного Интернет-представительства (сайта) становится одним из основных способов заявить о себе на рынке. Но для успешного поддержания сайта в актуальном состоянии необходимы системы, которые бы позволяли пользователям обновлять и изменять материалы, выложенные в сети.

Создание и изменение документов для публикации в Интернете осуществляется на тэговом языке разметки HTML, который интерпретируется браузером и отображается в виде, понятном для человека. Простые HTML-документы может создавать человек, не обладающий глубокими знаниями в web-технологиях, но чем сложнее требуемая структура документа и дизайнерские решения, тем больше проблем приходится решать автору для их реализации, поэтому для создания стандартных документов привлекаются специалисты по web-дизайну и информационному проектированию. Но так как часто нет возможности привлекать специалистов для полной поддержки, то необходимы инструменты, которые бы позволяли обычному пользователю публиковать материалы на сайте. Поэтому основа сайта, самая сложная часть с точки зрения реализации, создается специалистами в области web-технологий, а пользователям предоставляются специальные программные комплексы, которые позволяют осуществлять управление наполнением сайта.

Самыми востребованными инструментами, выполняющими данные задачи, являются так называемые редакторы в стиле WYSIWYG (What You

See Is What You Get), которые позволяют создавать публикуемый документ без использования специализированных знаний. WYSIWYG-редакторы делятся на два больших класса: специализированные локальные приложения и редакторы, встраиваемые в приложения web. Первые обладают более широким набором свойств и возможностей нежели вторые, но главное преимущество вторых – отсутствие привязки к машине пользователя и мгновенная публикация на сайте. Вы можете модифицировать, добавлять и удалять документы без использования специализированного ПО, пользуясь только браузером.

Еще один класс систем, которые следует упомянуть – системы управления содержимым (Content Management Systems, CMS). Такие системы представляют сайт в виде набора документов, упорядоченных в древовидную структуру. Документ обычно описывается с помощью набора полей, которые задают его содержание. Системы управления контентом предоставляют средства для создания, удаления и изменения документов, а так же механизмы управления структурой сайта. В большинстве случаев предоставляемый пользователю интерфейс для управления сайтом является web-приложением и состоит из набора определенных форм.

Система V.D.O.M. (Visual Dynamic Object Model / Визуальная Динамическая Объектная Модель) – клиент-серверная система, предназначенная для создания и управления сайтом при помощи специализированных визуальных средств и инструментария. Центральную роль в системе занимает объект пользовательского интерфейса. Система имеет несколько десятков объектов, которые позволяют построить полнофункциональные web-сайты, и была разработана с целью, помочь организациям и пользователям разрабатывать приложения для web без знания языков разметки и навыков программирования.

Поэтому французская компания BGSoft приняла решение сделать аналог своего настольного продукта Eye on file (EOF) на технологии VDOM.

Целью данной работы является разработка web-приложения EOF на базе VDOM-технологий для организации архивов документов, с возможностью полнотекстового поиска.

ГЛАВА 1. АНАЛИЗ

1.1 Основные требования

Согласно требованиям заказчика, разрабатываемое приложение должно:

Решать следующие задачи:

- Хранить документы
- Предоставлять доступ к документам
- Совершать поиск по документам.
- Разделять документы по группам.
- Предоставлять различный уровень прав доступа к ресурсам.

Быть разработана на платформе VDOM.

Обеспечивать возможность интеграции с другими приложениями.

1.2 Предметная область

На основе описания предметной области, предоставленного заказчиком, можно выделить некоторые ключевые сущности. Рассмотрим их.

Документ – составная сущность, представляющая из себя файл, который, как правило, должен содержать в себе текст и принадлежащие ему значения мета полей.

Документ должен находиться в файловой системе сервера, значение мета полей должны хранится в базе данных.

Документы должны группироваться по вида мета полей. Что определено **архивом** в котором находится документ.

У всех архивов должны быть обязательные мета поля: имя файла, дата создания, создатель, текстовое содержимое файла. Кроме того **мета поля** могут быть и дополнительные.

Должна быть возможность производить **поиск**. Поиск должен производится как по текстовому содержимому файла, так и его мета полям. Так же должно быть 2 вида поиска: быстрый и расширенный. Если пользователю приходится часто пользоваться каким-то шаблоном расширенного поиска, то должна быть возможность его сохранить.

У **пользователей** и **групп пользователей** должен различный уровень прав доступа к архивам документов, **папкам** в которых находятся архивы документов и сохраненным шаблонам поисков.

Итак, в рассматриваемой предметной области можно выделить следующие ключевые сущности:

- Документ
- Архив документов
- Мета поле.
- Поиск
- Пользователь.
- Группа пользователей

ГЛАВА 2. ОБЗОР СУЩЕСТВУЮЩИХ ПОИСКОВЫХ СИСТЕМЫ

2.1 История поисковых систем

Одним из первых инструментов поиска в интернете (до Всемирной паутины) был Archie.

Первой поисковой системой для Всемирной паутины был «Wandex», уже не существующий индекс, который создавал «World Wide Web Wanderer» — бот, разработанный Мэтью Грэем (англ. *Matthew Gray*) из Массачусетского технологического института в 1993. Также в 1993 году появилась поисковая система «Aliweb», работающая до сих пор. Первой полнотекстовой (т. н. «crawler-based», то есть индексирующей ресурсы при помощи робота) поисковой системой стала «WebCrawler», запущенная в 1994. В отличие от своих предшественников, она позволяла пользователям искать по любым ключевым словам на любой веб-странице — с тех пор это стало стандартом во всех основных поисковых системах. Кроме того, это был первый поисковик, о котором было известно в широких кругах. В1994 был запущен «Lycos», разработанный в университете Карнеги Мелона.

Вскоре появилось множество других конкурирующих поисковых машин, таких как «Excite», «Infoseek», «Inktomi», «Northern Light» и «AltaVista». В некотором смысле они конкурировали с популярными интернет-каталогами, такими, как «Yahoo!». Позже каталоги соединились или добавили к себе поисковые машины, чтобы увеличить функциональность. В 1996 году русскоязычным пользователям интернета стало доступно морфологическое расширение к поисковой машине Altavista и оригинальные российские поисковые машины «Рамблер» и «Апорт». 23 сентября 1997 была открыта поисковая машина Яндекс.

В последнее время завоёвывает всё большую популярность практика применения методов кластерного анализа и поиска по метаданным. Из

международных машин такого плана наибольшую известность получила «Clusty» компании Vivísimo. В 2005 году на российских просторах при поддержке МГУ запущен поисковик «Нигма», поддерживающий автоматическую кластеризацию. В 2006 году открылась российская метамашина Quintura, предлагающая визуальную кластеризацию в виде облака тегов. «Нигма» тоже экспериментировала с визуальной кластеризацией.

Помимо поисковых машин для Всемирной паутины, существовали и поисковики для других протоколов, такие как Archie для поиска по анонимным FTP-серверам и «Veronica» для поиска в Gopher.

2.2 Популярные поисковые системы

Согласно данным LiveInternet об охвате русскоязычных поисковых запросов:

Всеязычные:

- Google (37,2 %)
- Bing (0,8 %)
- Yahoo! (0,2 %) и принадлежащие этой компании поисковые машины:

Русскоязычные — большинство «русскоязычных» поисковых систем индексируют и ищут тексты на многих языках — украинском, белорусском, английском, татарском и др. Отличаются же они от «всеязычных» систем, индексирующих все документы подряд, тем, что в основном индексируют ресурсы, расположенные в доменных зонах, где доминирует русский язык или другими способами ограничивают своих роботов русскоязычными сайтами.

- Яндекс (48,1 %)
- Mail.ru (5,9 %)
- Рамблер (1,2 %)
- Нигма (0,3 %)

Некоторые из поисковых систем используют внешние алгоритмы поиска. Так, Qip.ru использует поисковый механизм Яндекса, а Nigma сочетает в себе как свой алгоритм, так и сборную выдачу от других поисковиков.

2.3 Необычные

- Koogle (с его помощью ортодоксальные иудеи могут найти контент, признанный раввинами удовлетворяющим религиозным требованиям).[3]
- Yauba (индийский поиск «для параноиков» — в отличие от привычных поисковиков, за пользователями не следят, а все записи о его действиях удаляются с сервера).[3]
- TinEye — поисковая система, специализирующаяся на поиске изображений в Интернете.
- Генон — поисковая система, собирающая и создающая контент у себя на сайте.
- ImHalal - первый исламский поисковик. Новый поисковик может отличить разрешённые для мусульман (халяльные) результаты поиска от запретных (харамных).

2.4 Проблемы поисковых систем

KaZaA и Церковь Сайентологии использовали Закон об авторском праве в цифровую эпоху (DMCA), чтобы потребовать от Google удалить ссылки на материалы на их сайтах якобы защищённые авторским правом. Google по закону обязан удалить эти ссылки, но вместо того, чтобы убрать результаты поиска, предпочитает связать результаты с жалобами, которые подали эти организации.

New York Times жаловалась на то, что кэширование их содержания поисковым роботом — особенность, используемая поисковиками, в том числе Google Web Search, нарушает авторские права. Google соблюдает стандартные Интернет-приёмы для запросов об отключении кэширования посредством файла robots.txt — стандартного механизма, позволяющего администраторам веб-сайта потребовать исключения своего сайта или его части из результатов поиска — или через мета-теги, позволяющие редактору контента указать, можно ли индексировать или архивировать документ и можно ли проходить по ссылкам в документе. Окружной суд США штата Невада постановил, что кэши компании Google не нарушают авторских прав согласно американскому законодательству в делах Field v. Google и Parker v. Google.

ГЛАВА 3. ОБЗОР ПОИСКОВЫХ РЕШЕНИЙ ДЛЯ WEB

3.1 Требования к поисковым решениям

При выборе поискового механизма следует учитывать следующие параметры:

скорость индексирования - то есть, как быстро поисковый сервер обрабатывает документы и заносит их в свой индекс, делая доступным поиск по ним. Обычно измеряется в мегабайтах чистого текста в секунду.

скорость переиндексации - в процессе работы документы изменяются или добавляются новые, поэтому приходится переиндексировать информацию. Если сервер поддерживает инкрементное индексирование, то обрабатываются только новые документы, а обновление всего индекса оставляем на потом или даже вообще не делать. Другие сервера требуют полной перестройки индекса при добавлении новой информации, либо используют дополнительные индексы (дельта-индекс), в который включается только новая информация

поддерживаемые API - если используется поисковик в связке с веб-приложением, обратите внимание на наличие встроенного API к языку или платформе. Большинство поисковиков имеют API для всех популярных платформ - Java, PHP, Ruby, Phyton

поддерживаемые протоколы - кроме API важны и протоколы доступа, в частности, если необходимо получить доступ с другого сервера или приложения, к которому нет родного API. Обычно поддерживаются XML-RPC (или разновидности, вроде JSON-RPC), SOAP или доступ через http/socket.

размер базы и скорость поиска - эти параметры очень взаимосвязаны и если реализуется что-то уникальное и предусматривается, что могут быть миллионы и больше документов в базе, по которым нужно производить

мгновенный поиск, то посмотрите на известные реализации выбранной платформы. Хотя никто не заявляет явно про ограничения на количество документов в базах, и на небольших коллекциях (например, несколько тысяч или десятков тысяч документов) все поисковики будут примерно одинаковые, но если речь идёт о миллионах документов - это может стать проблемой. Кстати говоря, сам по себе этот параметр не всегда имеет значение, нужно смотреть на особенности каждой системы и алгоритмы работы с поиском, а также на другие параметры, например, скорость переиндексации или возможные типы индексов и системы их хранения.

поддерживаемые типы документов - конечно, любой сервер поддерживает обычный текст (хотя следует смотреть на возможность работы с многоязычными документами и кодировкой UTF-8), но если необходимо индексировать разные типы файлов, например, HTML, XML, DOC или PDF, то стоит посмотреть на те решения, где есть встроенный компонент для индексации различных форматов и извлечения из них текста. Конечно, все это можно сделать прямо в приложении, но лучше поиска готовые решения. Сюда же относится и поддержка индексирования и поиска информации, которая хранится в СУБД - не секрет, что такое хранение самое распространённое для веб-приложений, и лучше, чтобы поисковый сервер работал напрямую с базой данных, без необходимости вручную извлекать и "скармливать" ему документы для индексирования.

работа с разными языками и стемминг - для корректного поиска с использованием разных языков необходима родная поддержка не только кодировок, но и работа с особенностями языка. Все поддерживают английский язык, который для поиска и обработки достаточно простой, но для русского и других аналогичных необходимо использовать автоматические средства для обеспечения морфологии. Модуль стемминга позволяет склонять и разбирать слова в поисковом запросе для более корректного поиска. Если для критичен поиск на русском языке, необходимо

внимание на присутствие этого модуля и его особенности (конечно, ручной стемминг лучше автоматического, но сложнее, хотя и автоматические есть очень разные).

поддержка дополнительных типов полей в документах - кроме самого текста, который индексируется и в котором производится поиск, необходимо наличие возможности хранить в документе неограниченное количество других полей, которые могут хранить мета-информацию о документе, что необходимо для дальнейшей работы с результатами поиска. Очень желательно, чтобы количество и типы полей не ограничивались, а индексируемость полей можно было настраивать. Например: в одном поле хранится название, во втором - аннотация, в третьем - ключевые слова, в четвёртом - идентификатор документа в системе. Необходимо гибко настраивать область поиска (в каждом поле или в указанных), а также те поля, которые будут извлекаться с базы поисковика и выдаваться в результатах поиска.

платформа и язык - это так же важно, хотя и в меньшей степени. Если необходимо выделять поиск в отдельный от приложения модуль или сервер, или даже выносите его на отдельный сервер (железо в смысле), то роль платформы не такая и большая. Обычно это или C++ или Java, хотя есть варианты и для других языков (обычно порты решений на java).

наличие встроенных механизмов ранжирования и сортировки - особенно хорошо, если поисковик можно расширять (и он написан на известном вам языке) и написать нужные вам реализации этих функций, ведь существует очень много разных алгоритмов, и не факт, что используемый по умолчанию в поисковике вам подойдёт.

Конечно, есть ещё очень много разных параметров, да и сама область поиска данных достаточно сложная и серьёзная, но для наших применений этого вполне достаточно.

Теперь кратко о тех поисковых решениях, на которые следует обратить внимание. Намеренно не рассматривается встроенные в СУБД решение - FULLTEXT в MySQL и FTS в PostgreSQL (интегрирован в базу с версии 8.3). В MySQL решение не может применяться для серьёзного поиска, особенно по большим объёмам данных, поиск в PostgreSQL намного лучше, но только если вы уже применяете эту базу. Хотя, как вариант - ставить отдельный сервер БД и там использовать только хранение данных и поиск тоже вариант.

3.2. Обзор Sphinx search engine

Тип: отдельный сервер, MySQL storage engine

Платформа: C++/кросс-платформенный

Индекс: монолитный + дельта-индекс, возможность распределённого поиска

Поисковые возможности: булевый поиск, поиск по фразам и т.п. с возможностью группировки, ранжирования и сортировки результата

API и протоколы: SQL DB (а также встроенная поддержка MySQL и PostgreSQL), собственный XML-интерфейс, встроенные API для PHP, Ruby, Python, Java, Perl

Поддержка языков: встроенный английский и русский стемминг, soundex для реализации морфологии

Дополнительные поля: да, неограниченное количество

Типы документов: только текст или собственные XML-формат

Размер индекса и скорость поиска: очень быстрый, индексация около 10 Мб/сек (зависит от CPU), поиск около 0.1 сек/~2 - 4 Гб индексе, поддерживает размеры индекса в сотни Гб и сотни миллионов документов

(это если не использовать кластеризацию), однако есть примеры работ на терабайтных базах данных.

Лицензия: open source, GPL 2 или коммерческая.

Sphinx вероятно самый мощный и быстрый из всех открытых движков, которые будут рассмотрены. Особенно удобен тем, что имеет прямую интеграцию с популярными базами данных и поддерживает развитые возможности поиска, включая ранжирование и стемминг для русского и английского языка. Поддерживаются и нетривиальные возможности вроде распределённого поиска и кластеризации, однако фирменной особенностью является очень и очень высокая скорость индексации и поиска, а также способность отлично распараллеливаться и утилизировать ресурсы современных серверов. Известны очень серьёзные инсталляции, содержащие терабайты данных, поэтому Sphinx вполне можно рекомендовать как выделенный поисковый сервер для проектов любого уровня сложности и объёма данных. Прозрачная работа с самыми популярными базами данных MySQL и PostgreSQL позволяет его использовать в обычном для веб-разработки окружении, к тому же сразу "из коробки" есть API для разных языков, в первую очередь, для PHP без применения дополнительных модулей и библиотек расширения. Но сам поисковик необходимо компилировать и устанавливать отдельно, поэтому на обычном хостинге он неприменим - только VDS или собственный сервер, причём, желательно большие объемы памяти. Индекс у поисковика монолитный.

SphinxSE - это версия, функционирующая как движок хранения данных для MySQL (требует патча и перекомпиляции базы), **Ultrasphinx** - это конфигуратор и клиент для Ruby (кроме присутствующего в

дистрибутиве API), кроме этого есть плагины для многих известных CMS b блог-платформ, вики, которые заменяют стандартный поиск

3.3. Обзор Apache Lucene

Тип: отдельный сервер или сервлет, встраиваемая библиотека

Платформа: Java/кросс-платформенный (есть порты на множество языков и платформ)

Индекс: инкрементный индекс, но требующий операции слияния сегментов (можно параллельно с поиском)

Поисковые возможности: булевый поиск, поиск по фразам, нечёткий поиск и т.п. с возможностью группировки, ранжирования и сортировки результата

API и протоколы: Java API

Поддержка языков: отсутствует морфология, есть стемминг (Snowball) и анализаторы для ряда языков (включая русский)

Дополнительные поля: да, неограниченное количество

Типы документов: текст, возможно индексация базы данных через JDBC

Размер индекса и скорость поиска: около 20 Мб/минута, размер индексных файлов ограничен 2 Гб (на 32-bit ОС). Есть возможности параллельного поиска по нескольким индексам и кластеризация (требует сторонних платформ)

Лицензия: open source, Apache License 2.0

Lucene - самый известный из поисковых движков, изначально ориентированный именно на встраивание в другие программы. В частности,

его широко используют в Eclipse (поиск по документации) и даже в IBM (продукты из серии OmniFind). В плюсах проекта - развитые возможности поиска, хорошая система построения и хранения индекса, который может одновременно пополнятся, удалятся документы и проводится оптимизация вместе с поиском, а также параллельный поиск по множеству индексов с объединением результатов. Сам индекс построен из сегментов, однако для улучшения скорости рекомендуется его оптимизировать, что часто означает почти те же затраты, что и на переиндексацию. Изначально присутствуют варианты анализаторов для разных языков, включая русский с поддержкой стемминга (приведения слов к нормальной форме). Однако минусом является все же низкая скорость индексации (особенно в сравнении с Sphinx), сложность работы с базами данных и отсутствие API (кроме родного Java). И хотя для достижения серьёзных показателей Lucene может кластеризироваться и хранить индексы в распределённой файловой системе или базе данных, для этого требуется сторонние решения, так же как и для всех остальных функций - например, изначально он умеет индексировать только обычный текст. Но именно в плане использования в составе сторонних продуктов Lucene "впереди планеты всей" - ни для какого другого движка нет столько портов на другие языки и использования. Одним из факторов такой популярности является и очень удачный формат файлов индексов, который используют сторонние решения, поэтому вполне можно строить решения, работающие с индексом и производящие поиск, но не имеющие собственного индексатора (это легче реализовать и требования намного ниже).

3.4. Обзор Solr

Solr - лучшее решение на базе Lucene, значительно расширяющее её возможности. Это самостоятельный сервер корпоративного уровня, предоставляющий широкие поисковые возможности в качестве веб-сервиса. Стандартно Solr принимает документы по протоколу HTTP в формате XML и

возвращает результат также через HTTP (XML, JSON или другой формат). Полностью поддерживается кластеризация и репликация на несколько серверов, расширена поддержка дополнительных полей в документах (в отличие от Lucene, для них поддерживаются различные стандартные типы данных, что приближает индекс к базам данных), поддержка фасетного поиска и фильтрации, развитые средства конфигурирования и администрирования, а также возможности бекапа индекса в процессе работы. Встроенное кеширование также повышает скорость работы. С одной стороны, это самостоятельное решение на базе Lucene, с другой - её возможности очень существенно расширены относительно базовых, поэтому если вам необходим отдельный поисковый сервер, обратите сперва внимание на Solr.

3.5. Обзор Nutch

Nutch - второй известнейший проект на базе Lucene. Это веб-поисковый движок (поисковый механизм + веб-паук для обхода сайтов) совмещённый с распределённой системой хранения данных Hadoop. Nutch "с коробки" может работать с удалёнными узлами в сети, индексирует не только HTML, но и MS Word, PDF, RSS, PowerPoint и даже MP3 файлы (мета-теги, конечно). Если стоит задача сделать небольшой локальный поисковик по местным ресурсам или заранее ограниченному набору сайтов, при этом нужен полный контроль над всеми аспектами поиска, или создаётся исследовательский проект для проверки новых алгоритмов, в таком случае Nutch станет лучшим выбором. Однако следует учесть, что его требования к аппаратной части и широком канале - для реального web-поисковика счёт трафика идёт на террабайты.

Но не только за счёт проектов-надстроек известен и популярен Lucene. Будучи лидером среди открытых решений и воплотив в себя множество отличных решений, Lucene первый кандидат для портирования на другие платформы и языки. Сейчас имеются следующие порты (я имею ввиду те, что более-менее активно развиваются и максимально полные порты):

3.6. Обзор Xapian

Тип: встраиваемая библиотека

Платформа: C++

Индекс: инкрементный индекс, прозрачно обновляемый параллельно с поиском, работа с несколькими индексами, in-memory индексы для небольших баз.

Поисковые возможности: булевый поиск, поиск по фразам, поиск с ранжированием, поиск по маске, поиск по синонимам и т.п. с возможностью группировки, ранжирования и сортировки результата

API и протоколы: C++, Perl API, Java JINI, Python, PHP, TCL, C# и Ruby, CGI интерфейс с XML/CSV форматом

Поддержка языков: отсутствует морфология, есть стемминг для ряда языков (включая русский), реализована проверка правописания в поисковых запросах

Дополнительные поля: отсутствует

Типы документов: текст, HTML, PHP, PDF, PostScript, OpenOffice/StarOffice, OpenDocument, Microsoft Word/Excel/Powerpoint/Works, Word Perfect, AbiWord, RTF, DVI, индексация SQL баз через Perl DBI

Размер индекса и скорость поиска: известны работающие инсталляции на 1.5 Тб индекса и количество документов в сотни миллионов.

Лицензия: open source, GPL

Пока это единственный претендент на конкуренцию с господством Lucene и Sphinx, выгодно отличается от них наличием "живого" индекса, не требующего перестройки при добавлении документов, очень мощным языком запросов, включая встроенный стемминг и даже проверку орфографии, а также поддержку синонимов. Эта библиотека будет лучшим выбором, если система на perl или же требуется развитые возможности остроения поисковых запросов и происходит очень частое обновление индекса, при этом новые документы должны быть сразу доступны для поиска. Однако нет информации о возможности добавлять к документам произвольные дополнительные поля и получать их с результатами поиска, поэтому связь системы поиска с собственной может представлять определённые трудности. В пакет входит **Omega** - надстройка над библиотекой, которая готова для использования в качестве самостоятельного поисковика и как раз она отвечает за возможности индексации разных типов документов и CGI интерфейс.

Вот наиболее значимые плюсы Xapian:

- Является свободным ПО с открытыми исходными кодами. Лицензируется по GPL.
- Поддерживает Юникод (включая коды символов, не попадающие в BMP) и хранит проиндексированные данные в UTF-8.
- Высокая переносимость. Запускается под Linux, Mac OS X, FreeBSD, NetBSD, OpenBSD, Solaris, HP-UX, Tru64, IRIX и, вероятно, под другими Unix-платформами, а также и в Microsoft Windows.
- Написан на C++. Обвязки для Perl дуступны в модуле Search::Xapian в CPAN. Обвязки для Java JNI включены в модуль xapian-bindings. Кроме того поддерживается SWIG, который

может генерировать обвязки для многих языков. В настоящее время это работает для Python, PHP, TCL, C# и Ruby.

- Ранжированный вероятностный поиск - важные слова получают больший вес, чем неважные слова. Этим достигается то, что более релевантные документы с большей вероятностью попадают в топ поисковой выдачи.
- Обратная релевантность - для данного документа или нескольких документов Xapian может предложить уточняющие термины для введенного запроса, дополнительные документы на смежные темы, категории документов и т.п.
- Поиск фраз и "поиск с зазором". Пользователи могут искать точные совпадения фраз или с указанием числа слов, которые могут размежать ключевые слова в заданном порядке или в произвольном порядке.
- Полный набор операторов для структурированного логического поиска ("ассортимент NOT рынок" и т.п.). Результаты логического поиска ранжируются с помощью вероятностных коэффициентов. Логические фильтры могут применяться также для ограничения вероятностного поиска.
- Поддержка стемминга поисковых запросов (например, по запросу "футбол" будут находиться также документы со словами "футбольный" и "футболист"). Это помогает находить релевантные документы, которые иначе были бы опущены. В настоящее время в библиотеку включены стеммеры для следующих языков: английский, венгерский, голландский, датский, испанский, итальянский, немецкий, норвежский, португальский, румынский, русский, турецкий, финский, французский и шведский.

- Поддерживаются синонимы. Как в явной форме (например, "~деньги"), так и путем автоматического расширения запроса.
- Xapian может опознавать запросы с орфографическими ошибками и предлагать пользователю исправленные запросы. Эта возможность основана на анализе слов, находящихся в проиндексированных данных, поэтому работает даже для тех слов, которые отсутствуют в словарях (например, слово "xapian" будет предложено в качестве поправки к запросу "xapain").
- Поддержка файлов баз данных, превышающих 2GB. Это необходимо для сканирования больших коллекций документов.
- Независимость от формата даты на используемой платформе. Вы можете создать базу данных на одной машине и искать по ней на другой машине.
- Позволяет одновременное обновление индекса и поиск по индексу. Новые документы моментально становятся доступными для поиска.
- Вместе с библиотекой мы предоставляем несколько маленьких образцов программ, которые используют нашу библиотеку, а также большое приложение, - Omega, - индексатор и CGI-скрипт для вывода результатов поиска.
- Индексатор умеет индексировать документы HTML, PHP, PDF, PostScript, OpenOffice/StarOffice, OpenDocument, Microsoft Word/Excel/Powerpoint/Works, Word Perfect, AbiWord, RTF, DVI, Perl POD, а также простой текст. Научить индексатор поддерживать другие форматы довольно легко при наличии доступного фильтра для преобразования. Этот индексатор работает с локальными файлами, но мы также предоставляем скрипт, позволяющий это изменить и создать веб-паука для

организации поиска по удаленным сайтам с использованием Omega.

- Вы также можете индексировать данные в какой-нибудь SQL- или иной РСУБД, поддерживаемой модулем Perl DBI. Таковыми являются MySQL, PostgreSQL, SQLite, Oracle, DB2, MS SQL, LDAP и ODBC.
- Поисковый CGI фронтенд с полностью настраиваемым внешним видом. Он может быть перенастроен для вывода результатов в XML или CSV, что может оказаться полезным, если вы хотите непосредственно использовать результаты поиска в программном коде для динамической генерации своих собственных страниц или для интеграции поиска в работающий на AJAX'e фронтент.

3.7. Вывод

Xapian достаточно хороший и качественный продукт, однако менее распространённый и гибкий, чем остальные. Для приложений на C++ и требованиями к широким возможностям языка запроса он будет лучшим выбором, однако требует ручной доводки и модификаций для встраивания в собственный код или использования как отдельного поискового сервера.

ГЛАВА 4. РЕАЛИЗАЦИЯ

4.1 Платформа VDOM

В данном подразделе приводится описание платформы VDOM, на которой разрабатывалось приложение EOF.

4.1.1 Технология VDOM

Web-сервер VDOM Box является развитием идей технологии VDOM (Visual Dynamic Object Model, визуальная динамическая объектная модель) [6] – клиент-серверной системы, предназначенной для создания и управления сайтом при помощи специализированных визуальных средств. Технология представляет сайт в виде объектов пользовательского интерфейса и, по сути, реализует компонентную модель разработки web-приложений. Технология VDOM разработана CyberTronique inc. с целью помочь организациям и пользователям разрабатывать web-приложения без глубоких знаний в web-технологиях.

Для создания сайта в системе VDOM пользователь использует специальное web-приложение, которое включает в себя такие инструменты, как WYSIWYG-редактор для страниц (рисунок 5), специальные средства для проектирования схемы приложения и интерфейс для разработки сценариев, встраиваемых в страницы сайта.

Рисунок 1- Интерфейс визуального редактора VDOM

На сегодняшний день версия технологии VDOM существует в форме аппаратно-программного комплекса – устройства, выполняющего функцию web-сервера – VDOM Box Server (рисунок 6).

Рисунок 2 - Внешний вид VDOM Box Server

4.1.2 Технология E2VDOM

Система E2VDOM [7] является вторым поколением системы VDOM. В ней была представлена возможность управлять не только визуальной составляющей страниц, но и также логикой их работы. Логика реализуется за счет использование системы **event->action**. Каждый объект помимо атрибутов имеет определенный набор событий, на которые он может

реагировать. Например, для кнопки (тип Button) таким событием может быть OnClick, генерирующееся, когда пользователь щелкает по кнопке. Конкретное событие может быть связано с определенным обработчиком (action-ом).

Различают *клиентские* и *серверные* action-ы. Клиентские action-ы выполняются на стороне клиента и не требуют какого-либо вмешательства со стороны сервера. Особенность серверных action-ов заключается в том, что они выполняются на VDOM сервере. Сначала сигнал о том, что на стороне клиента сработало какое-либо событие (например, пользователь нажал на кнопку) отправляется на сервер, где выполняется его обработка. Если в ходе обработки события (выполнения скрипта на языке Python), у каких-либо объектов изменились значения атрибутов, сервер отправляет клиенту HTML код, соответствующий этим измененным объектам. То есть, во-первых, обработку всей логики осуществляет сервер, что снижает нагрузку на стороне клиента. Во-вторых, при изменении значений атрибутов отдельных объектов (т.е. их визуальной составляющей) выполняется повторная отрисовка только этих объектов, а не всей страницы целиком. Подробная схема работы системы **event->action** приведена на рисунке 7.

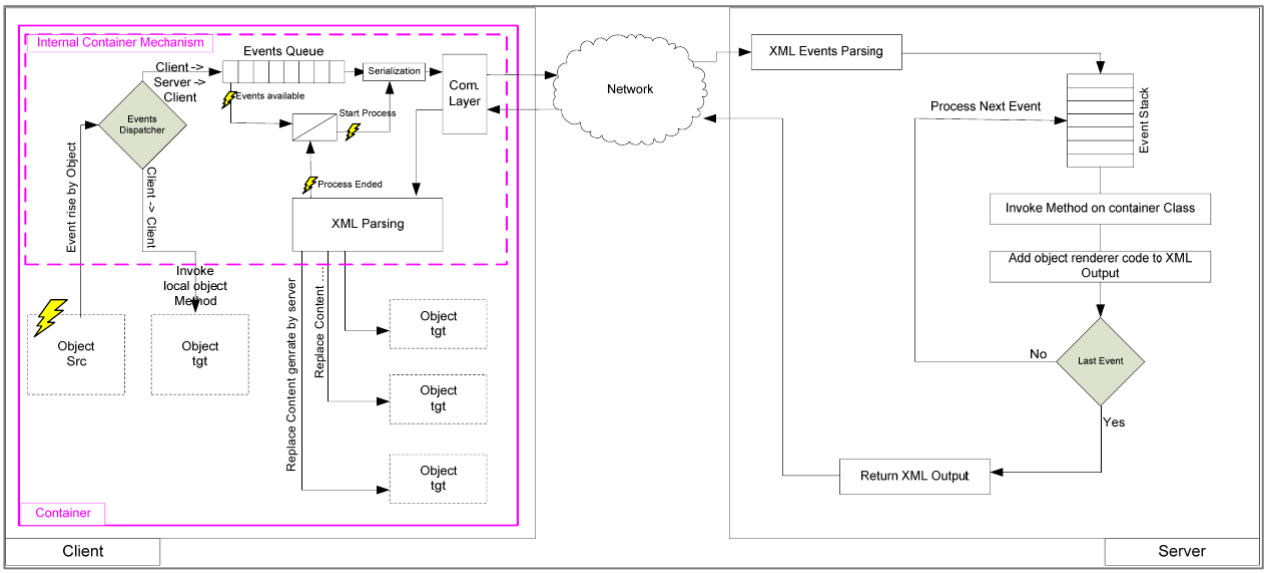

Рисунок 3 - Схема работы системы event->action платформы VDOM

4.1.3 Аппаратная и программная части

В качестве основы для построения аппаратной части VDOM Box Server был выбран класс миниатюрных материнских плат формата mini-itx. Такие платы обладают компактными размерами, но в то же время обеспечивают нужную производительность для работы web-сервера, обслуживающего сайты предприятий малого бизнеса. С учетом относительно небольшой цены, итоговое решение получается приемлемым по затратам. Аппаратная часть сервера состоит из двух основных частей:

Материнская плата формата mini-itx, на которой происходит работа VDOM Box сервера и реализуется взаимодействие с посетителями сайта.

Электронная плата, которая отвечает за настройку системы и управление лицензиями.

Вся программная часть хранится на внешней флеш-карте, доступной в системе только для чтения. Такое решение предотвращает возможность «поломки» системы в результате эксплуатации. Также такой способ хранения позволяет очень просто устанавливать систему на новые аппаратные платформы и обновлять программное обеспечение.

Пользовательские данные хранятся на жестком диске и могут быть скопированы на внешний носитель при помощи системы резервного копирования. Эта система также позволяет переносить web-приложения между различными устройствами.

Электронная смарт-карта хранит на себе данные о пользователе и лицензии. Она определяет лицензионные ограничения, такие, как максимальное количество объектов, которое может быть создано в системе.

Программная часть VDOM Box реализована на платформо-независимом языке Python, что обеспечило переносимость и простоту

поддержки решения. В качестве операционной системы была выбрана FreeBSD, так как она обеспечивает нужную функциональность и нетребовательна к ресурсам.

4.1.4 Объектная модель

В объектной модели выделяется два основных класса объектов: объекты-контейнеры и простые объекты. Основная задача контейнера – задавать логическую структуру страницы web-приложения, а также определять тип вывода содержимого. Предполагается реализация следующих типов контейнеров:

- HTML container: отвечает за вывод объектов в HTML-поток.
- Flash container: отвечает за вывод объектов во flash-анимацию.
- PDF container.
- PNG container.

Также при помощи определенных типов контейнеров можно создавать составные объекты, такие как «таблица». Для этих целей введены следующие типы контейнеров:

- Table container: отвечает за вывод таблицы целиком, содержит набор объектов типа row-container.
- Row container: отвечает за отображение одного ряда таблицы.
- Cell container: отвечает за отображение одной ячейки таблицы.

Такое разделение типов позволяет очень гибко конструировать страницы web-приложения. На рисунке 4 показан пример структуры страницы: верхний контейнер –это HTML-container, который содержит несколько объектов внутри себя: два других контейнера (Flash container, PNG container) и пять простых объектов (два Picture object, Menu object, два RichText Object).

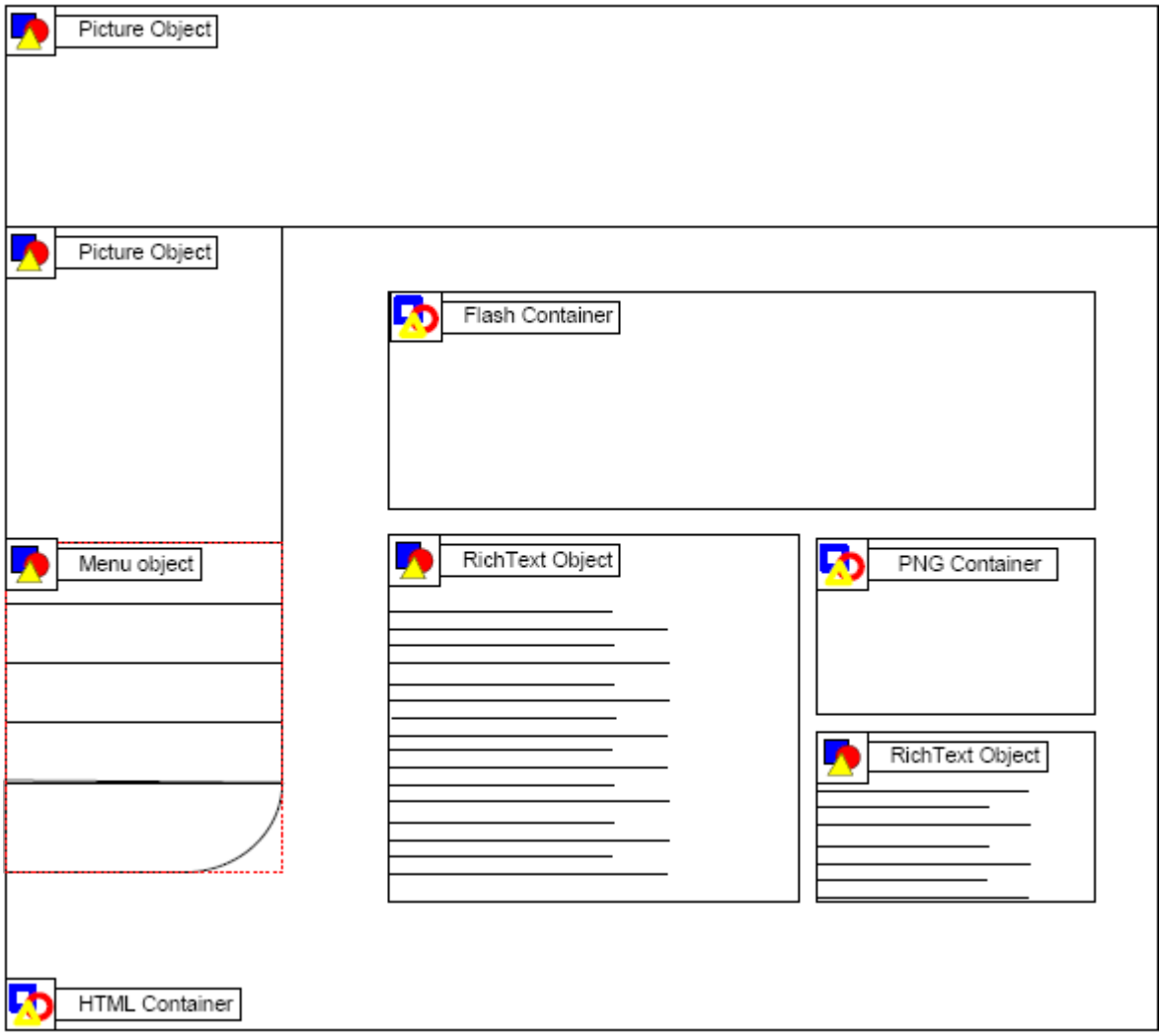

Рисунок 4 - Пример структуры страницы web-приложения с использованием объектов разных типов

Для представления объектов в системе используется XML-формат описания типов. Это позволяет легко создавать пользовательские объекты и переносить их из одной системы в другую. Также такое представление данных обеспечивает нужную динамичность для работы сервера. Описание типа имеет следующую структуру:

```
<Type>
<Information>...</Information>
<Attributes>...</Attributes>
<Resources>...</Resources>
<SourceCode>...</SourceCode>
</Type>
```

Секция **Information** описывает общую информацию типа: имя, описание, идентификатор, типы поддерживаемых контейнеров.

Секция **Attributes** содержит в себе описание атрибутов объектов данного типа и используется при трансляции XML-описания во внутреннее представление.

Секция **Resources** описывает различные файловые ресурсы, используемые типом, например, это могут быть картинки или иконки.

Секция **SourceCode** содержит в себе код на языке Python, реализующий функциональность данного класса. Здесь содержаться функции по выводу представления объекта в различные типы контейнеров и дополнительные функции, которые объект предоставляет.

Данный XML-документ обрабатывается на этапе загрузки сервера, и в результате формируется внутреннее представление объекта в памяти, которое используется при формировании объектов web-приложения.

4.1.5 Web-сервисы

Чтобы манипулировать данными на сервере, созданы web-сервисы, которые позволяют производить базовые операции над объектами в памяти. На базе этого механизма можно создавать сторонние приложения, которые будут производить администрирование системы либо расширять её функциональность. Например, IDE-редактор, разрабатываемый как основное средства разработки дизайна для приложений на VDOM Box Server, основан на этих web-сервисах.

Web-сервисы – это программные модули, работающие по протоколу SOAP, которые предоставляют удаленным программам функциональность, определенную на языке XML. Взаимодействие веб-сервисов и сторонних продуктов происходит по протоколу SOAP (надстройка над протоколом HTTP).

Для облегчения создания приложений для VDOM Box реализована библиотека, которая прозрачно для программиста представляет работу с объектами на сервере.

Рассмотрим основные принципы взаимодействия с сервером. Стандартный механизм веб-сервисов не имеет механизма сохранения данных о пользователе между отдельными транзакциями. По этой причине, чтобы обеспечить безопасность и удобство общения клиента с сервером, реализован механизм «сессий» наподобие механизма, реализованного во многих браузерах. При открытии соединения при помощи сервиса open_session текущему клиенту присваивается номер сессии, и дальнейшее общение происходит с его использованием. Окончание сессии производится при помощи сервиса close_session. Для предотвращения перехвата сессий предусмотрен механизм динамического ключа – строки, которая идентифицирует текущую транзакцию в диалоге. Данный ключ вычисляется на основании пользовательского пароля и номера сообщения и обеспечивает базовую защиту от перехвата сессии.

3.2 Реляционная база данных

В данном подразделе речь пойдет о разработке самой базы данных на платформе VDOM на основе полученной схемы.

Для начала рассмотрим технические возможности VDOM по разработке БД. Как уже было сказано ранее, в системе VDOM в качестве СУБД используется вариация SQLite – embedded SQLite 3. Данная СУБД является «встраиваемой», то есть разработанная с ее помощью база данных интегрируется непосредственно в само приложение.

Такой подход к организации БД хорош на этапе разработки. Он позволяет, не отвлекаясь на технические детали, сократить время разработки базы данных и соответственно самого приложения. Кроме того, при таком

подходе не требуется разворачивать отдельный сервер с СУБД и налаживать связь приложения с базой данных. Достаточно просто установить приложение на VDOM сервер, и оно будет полностью готово к работе.

Однако за данные удобства приходится расплачиваться относительно низким быстродействием базы данных, а также урезанной функциональностью самой СУБД. К наиболее значимым ее ограничениям относятся:

- Отсутствие поддержки составных первичных ключей.
- Отсутствие поддержки внешних ключей.
- Отсутствие поддержки каскадных операций.
- Отсутствие поддержки триггеров.
- Ограниченный набор типов данных.

Рассмотрим, как проходит разработка базы данных на платформе VDOM. Для создания и редактирования базы данных используется интегрированная среда разработки VDOM IDE, являющаяся основным средством разработчика на VDOM-платформе. Поскольку база данных встраивается непосредственно в приложение, то для начала необходимо создать как минимум пустое приложение. Далее в нем создается так называемый Database container, представляющий локальную базу данных текущего приложения. Создание таблиц осуществляется путем добавления в контейнер экземпляров VDOM-типа Database table. Редактирование структуры таблицы (то есть ее столбцов) осуществляется путем редактирования атрибута schema посредством специальной формы. Для каждого столбца можно задавать название, тип данных, а также ограничения целостности типа UNIQUE и NOT NULL. Заметим, что для каждой созданной таблицы автоматически генерируется суррогатный автоинкрементный столбец с именем «id», являющийся первичным ключом таблицы. Схема редактируемой базы данных отображается в главном окне

среды разработки. Изменение данных, хранящихся в таблице, осуществляется путем редактирования атрибута data через соответствующую форму.

В результате проделанной работы на основе построенной реляционной схемы была разработана база данных. Соответствующую реляционную диаграмму можно увидеть на рисунке 5. Первое, что бросается в глаза, отсутствие на диаграмме связей между таблицами (не поддерживаются внешние ключи).

В целом, несмотря на описанные ограничения используемой СУБД, получившаяся база данных оказалась вполне пригодной для приложения EOF. Проблему малого числа типов данных удалось решить сведением требуемых типов к имеющимся в наличие. Например, можно свести Boolean к Integer (0 – false, 1 - true), datetime – к string и так далее. Задачу контроля ограничений целостности, специфичных для предметной области (отсутствие поддержки триггеров), а также ограничений ссылочной целостности (отсутствие поддержки внешних ключей) удалось полностью решить на уровне слоя бизнес-логики. Что же касается проблемы низкого быстродействия, то, по словам заказчика, она не является критичной, по крайней мере, на начальном этапе эксплуатации системы.

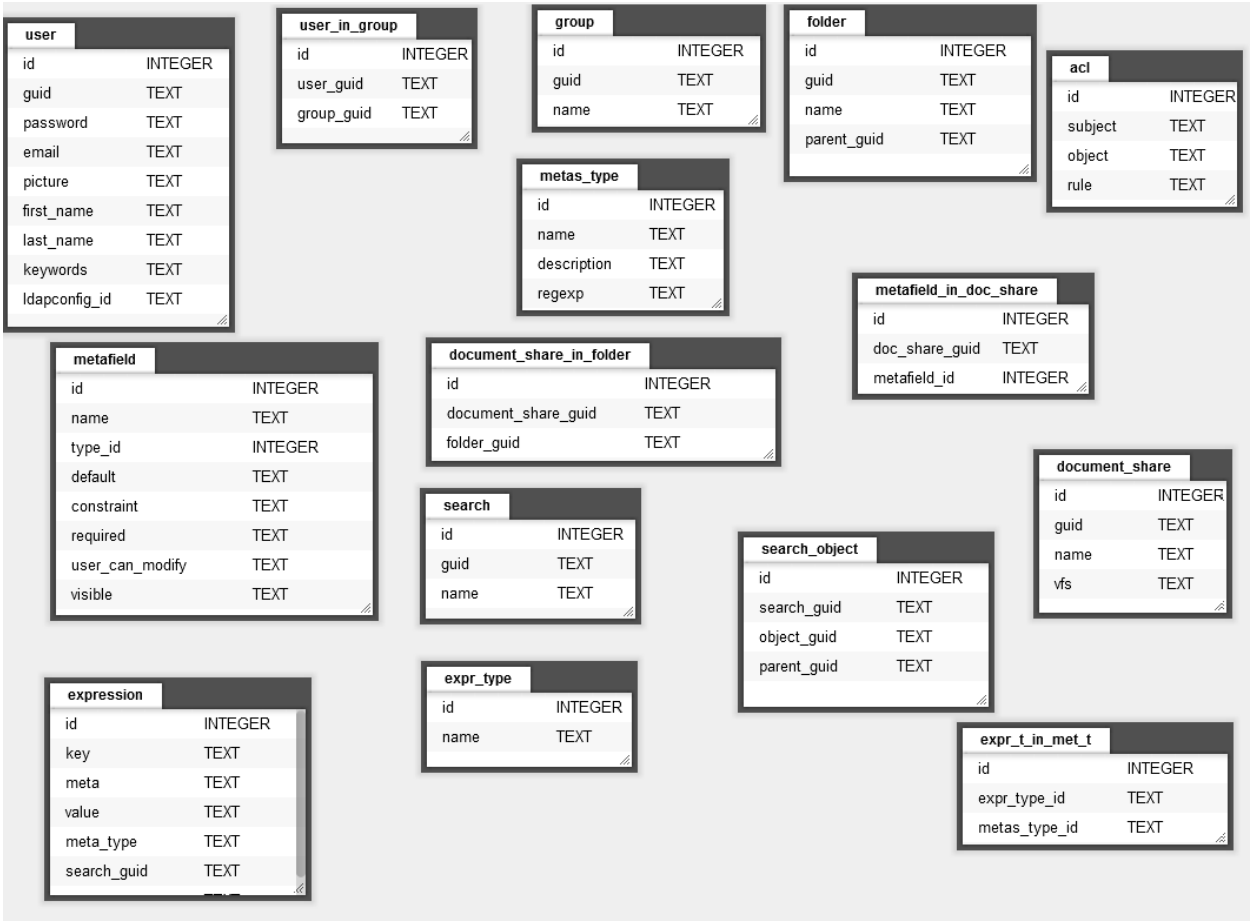

Рисунок 5 - Реляционная диаграмма, отображаемая средой VDOM IDE

4.2 Архитектура

В данном подразделе рассматривается реализация архитектуры системы VBI Partners, состоящей из слоя бизнес-логики и API, необходимого для интеграции с другими приложениями.

4.2.1 Слой бизнес-логики

Задача реализации данного слоя заключается в написании классов на языке Python [8, 9] в соответствии с полученной диаграммой. В данном пункте рассмотрены особенности реализации общей структуры классов, а также решения таких задач, как контроль ограничений целостности, преобразование типов данных между БД и слоем бизнес-логики, ограничение области видимости «закрытых getter-ов».

Значительную часть типичного класса слоя бизнес-логики составляют методы, выполняющие ORM-преобразования. Структура подобных методов предполагает определенную обработку данных в соответствии с их логикой и, естественно, обращение к базе данных. Взаимодействие с БД происходит за счет построения запросов на языке SQL и передаче их СУБД через стандартный интерфейс.

Рассмотрим интерфейс платформы VDOM для работы с базой данных. Он включает в себя три основные функции:

vdom_sql_fetch_one(dbase_guid, sql_query, params)

vdom_sql_fetch_all(dbase_guid, sql_query, params)

vdom_sql_commit(dbase_guid, sql_query, params)

Первые две функции предназначены для выборки данных из БД и принимают в качестве параметра select-запрос. Отличаются же они тем, что **vdom_sql_fetch_one** возвращает лишь первую строку результата переданного SQL-запроса в виде Python-списка, тогда как **vdom_sql_fetch_all** возвращает все строки запроса в виде списка списков. Третья функция, **vdom_sql_commit**, предназначена для внесения изменений в БД и принимает insert-, update- и delete-запросы. Для insert- и update-запросов возвращается значение первичного ключа последней вставленной/обновленной записи. Для delete-запроса возвращается **None.**

Заметим, что упомянутые функции имеют аналогичный набор параметров. Рассмотрим их подробнее:

dbase_guid – GUID локальной базы данных приложения.

sql_query – строка запроса на языке SQL.

params – кортеж значений параметров указанного SQL-запроса.

Строка SQL-запроса может содержать произвольное число параметров отмеченных знаком «?», список значений этих параметров указывается в кортеже **params**. Данный механизм работы с параметрами призван исключить возможность такой известной уязвимости приложений как SQL-инъекция. Приведем пример обращения к базе данных через стандартный интерфейс.

```
db_rows = vdom_sql_fetch_one("426f6f74-0000-11aa-aa11-00306543ecac",  "SELECT * FROM users WHERE id=?", (45, ))
```

Данный метод вернет Python-список значений атрибутов записи таблицы **users** со значением id, равным 45.

Заметим, что, база данных в типичном VDOM-приложении, как правило, одна, а значит, каждый раз указывать ее GUID при вызове стандартных функций не очень удобно. Данное неудобство можно преодолеть за счет использования класса (назовем его **Database**), который бы хранил ссылку на локальную БД. Интерфейс данного класса практически полностью повторяет стандартный VDOM-интерфейс, но устраняет необходимость каждый раз указывать GUID базы данных. Вообще говоря, введение подобного класса является стандартным приемом, используемым разработчиками на VDOM-платформе. Приведем пример обращения к базе данных через промежуточный интерфейс.

```
db_rows = Database.maindb().fetch_all("SELECT id, name FROM user")
```

Рассмотрим теперь решение такой важной задачи, как контроль ограничений целостности. Контроль ОЦ происходит внутри метода **save()** каждого класса. Сначала выполняется проверка всех условий целостности, которые, могут быть нарушены при сохранении текущего объекта в БД, после чего выполняется уже непосредственно сохранение объекта.

При нарушении какого-либо ограничения целостности генерируется исключение типа **EOFLogicError** с соответствующим описанием ошибки. Данный класс исключений (унаследованный от класса **Exception**) был введен для разграничения ошибок, порождаемых нарушением ОЦ предметной области, и ошибок другого плана.

Предполагается, что при необходимости сохранения объекта клиент слоя бизнес-логики будет помещать вызов метода **save()** внутрь блока **try…except** и должным образом обрабатывать исключительную ситуацию типа **EOFLogicException**, например, выводя полученное описание ошибки пользователю во всплывающем окне.

Заметим, что хотя СУБД Embedded SQLite 3 и поддерживает ограничения целостности типа UNIQUE и NOT NULL, к сожалению, при обращении к базе данных через стандартный интерфейс, клиенту не выдается информация о том, какой именно тип ОЦ был нарушен. Так как клиенту слоя бизнес-логики такая информация может быть интересна, проверки упомянутых ограничений целостности также были реализованы на уровне слоя.

Напомним, что СУБД Embedded SQLite имеет ограниченный набор типов данных, поэтому некоторые столбцы таблиц содержат данные в «неправильном» формате. Например, столбец с данными «дата/время» хранит их в виде строки, данные типа boolean хранятся как число и так далее. В классах бизнес-логики хотелось бы видеть в качестве значений атрибутов данные «правильных» типов. Таким образом, встает задача преобразования типов данных между БД и слоем бизнес-логики. Теперь рассмотрим, в каких методах необходимо выполнять данные преобразования. Изменение объектов бизнес-логики происходит внутри методов **insert()** и **update()**, а их создание по строке выборки из БД – внутри метода **__init__(row)**. Таким образом, выполнение преобразований данных необходимо включить именно в эти три

метода, что и было сделано. Например, метод **__init__(row)** класса **User** выглядит следующим образом:

```
    @classmethod
def __init__ (self, row):
    self.id             = row[0]
    self.guid           = row[1]
    self._password      = row[2]

    self._email         = row[3]
    self.picture        = row[4]
```

Теперь поговорим о реализации области видимости «закрытых getter-ов» класса (напомним, что к ним относятся методы выборки объектов данного класса, предназначенные для использования в методах других классов бизнес-логики). Здесь необходимо было решить задачу ограничения области видимости данных методов, сделав их доступными для классов слоя бизнес-логики и недоступными за его пределами. К сожалению, язык Python не располагает средствами, аналогичными, например, механизму дружественных классов в языке C++. Ввиду данных обстоятельств, пришлось сделать рассматриваемые методы открытыми, а также ввести условное соглашение об их неиспользовании за пределами слоя бизнес-логики.

Рассмотрим задачу «размещения» классов слоя бизнес-логики. На платформе VDOM существует понятие «библиотеки» (library). Отдельно взятая библиотека представляет собой обычный Python-модуль, то есть файл исходного кода на Python, который может быть использован как в других VDOM-библиотеках, так и в серверных скриптах (являющихся, как правило, обработчики определенных VDOM-событий на странице) с помощью стандартного механизма импорта языка Python. Таким образом, механизм

библиотек на платформе VDOM реализует возможность повторного использования кода.

Реализация каждого класса слоя бизнес-логики была оформлена в виде отдельной библиотеки. Была использована следующая схема именования: class_<название класса>. Например библиотека, реализующая класс **User**, называется class_user.

Непосредственно для написания классов использовался редактор скриптов VDOM Script Editor, являющийся неким дополнением к основной среде разработки – VDOM IDE, которая, конечно же, тоже поддерживает возможность редактирования Python-кода.

Что касается тестирования работоспособности разработанных классов, то в условиях отсутствия на данном этапе пользовательского web-интерфейса оно выполнялось на простой тестовой странице (содержащей несколько кнопок и визуальных компонентов для ввода/вывода данных).

В результате проделанной работы были реализованы 6 классов бизнес-логики: **ACL**, **User**, **Document**, **Group**, **Document_share** и **Search**. Фрагмент кода реализации одного из классов можно увидеть на рисунке 6.

```
320     def update_email(self):
321         query = """SELECT *
322                         FROM `user`
323                         WHERE email= ? """
324         row = Database.maindb().fetch_one( query, (self.email,) )
325         if row and row['guid']!=self.guid :
326             raise Exception("USER_EXISTS")
327 
328         query = """UPDATE user
329                     SET email=?
330                      WHERE guid=?    """
331         row = Database.maindb().commit( query,( self.email, self.guid))
332 
333         if not self.is_admin(): return
334 
335         query = """UPDATE config
336                     SET `value`=?
337                      WHERE item='admin'    """
338         row = Database.maindb().commit( query,( str(self.email),))
339 
```

Рисунок 6 - Фрагмент реализации класса User слоя бизнес-логики

4.2.2 API

В данном пункте рассмотрена реализация API на платформе VDOM.

Рассмотрим для начала средства, предоставляемые используемой платформой для решения поставленной задачи. Платформа VDOM предоставляет возможность удаленного вызова процедур посредством библиотеки, работающей по протоколу SOAP. Каждая процедура оформляется в виде отдельного файла исходного кода на языке Python и помещается в приложение, из которого может быть в последствие вызвана. Ввод и вывод данных осуществляется в формате XML. Вызываемая процедура получает данные через сессию, через переменную с фиксированным именем (**xml_data**). Выходные данные отправляются аналогично – сохраняются в переменную с фиксированным именем (**response**).

Теперь поговорим непосредственно о реализации API. Разработка каждой отдельной API-операции подразумевает последовательное решение трех задач, а именно:

- Анализ входных XML-данных.
- Реализация логики операции.
- Формирование выходных XML-данных.

Логика операций API сводилась к использованию слоя бизнес-логики и некоторой дополнительной обработке, специфичной для каждой отдельной операции.

Задача формирования выходных XML-данных решалась посредством такой структуры Python, как шаблоны строк (template strings). Например:

```
template_response = "<success>\n\t<user_id>%s</user_id>\n<success>"
request.session.value("response", template_response % user.id)
```

Задача анализа XML, решалась с помощью стандартной библиотеки Python – minidom, предоставляющей все необходимые для решения данной задачи средства.

Рассмотрим упомянутую библиотеку более детально. Анализ XML-данных начинается с вызова функции **parseString**, которая возвращает экземпляр класса **Document**, представляющий некую внутреннюю модель XML-документа. Согласно данной модели документ имеет иерархическую структуру вложенных *узлов* (экземпляров класса **Node**). Существуют два основных вида узлов: *узлы-элементы* (elements) и *текстовые узлы* (text nodes). Узлы-элементы представляют тег XML-документа и соответственно могут содержать произвольное число дочерних узлов различного вида. Текстовые узлы представляют текст, заключенный внутри тегов XML-документа.

Заметим, что любая последовательность символов, заключенных между границами тегов (в том числе содержащая пробелы, символы перехода на новую строку и другие), образует текстовый узел. В следующем примере тег **<info>** содержит три узла: текстовый (**“\n\t”**), элемент **<name>**, и еще один текстовый (**“\n”**).

```
<info>
  <name>John</name>
</info>
```

Получить корневой элемент XML-документа можно, вызвав метод **getDocumentElement()** экземпляра класса **Document.**

Так как объекты класса **Node** являются ключевыми в структуре XML-документа, рассмотрим вкратце его интерфейс. Объект класса **Node** имеет следующие атрибуты:

- **nodeType** – тип узла (имеет значение **ELEMENT_NODE** для узла-элемента и **TEXT_NODE** для текстового узла).

- **tagName** – название тега, представляемого элементом (для текстового узла всегда **None**).
- **attributes** – словарь атрибутов соответствующего тега, ключами которого являются названия атрибутов, а значениями – значения этих атрибутов (для текстового узла всегда **None**).
- **childNodes** – список дочерних узлов текущего элемента (для текстового узла всегда пустой список).
- **nodeValue** – строковое значение текстового узла (для элемента всегда **None**).

Также объект класса **Node** имеет метод **getElementsByTagName(tagName),** выдающий список его потомков (как прямых, так и не прямых) с указанным названием тега.

Можно условно выделить две стратегии анализа XML-документа. Первая из них предполагает рекурсивный обход XML-дерева, начиная от корневого элемента, и получение для текстовых узлов их строковых значений, а для элементов значений их атрибутов и списка их потомков. Если же в документе заведомо отсутствуют теги с одинаковым названием, то задача значительно упрощается. Можно просто вызывать метод **getElementsByTagName** от корневого элемента и анализировать полученные элементы – в этом заключается идея второго подхода.

В силу особенностей спецификации разрабатываемого API (отсутствие повторяющихся тегов в структуре входных данных каждой операции) при анализе XML-данных был использован второй подход (то есть предполагающий использование метода **getElementsByTagName**).

Рассмотрим теперь вопрос «размещения» операций внешнего API. Для обеспечения возможности удаленного вызова операции она должна быть реализована в виде отдельного Python-скрипта и помещена в определенный HTML-контейнер. В соответствии с этим условием в приложении был создан HTML-контейнер с именем «API», внутрь которого были помещены

скрипты, реализующие соответствующие операции. Именование скриптов производилось в соответствии с названиями реализуемых операций.

Для тестирования разрабатываемого API было написано небольшое VDOM-приложение, которое позволяло устанавливать связь с конкретным сервером, запускать с него удаленные процедуры и отслеживать возвращаемый ими результат. При разработке тестирующего приложения использовался стандартный Python-класс **VDOMService**, представляющий собой некую «обертку» над упомянутой SOAP-библиотекой. Рассмотрим интерфейс данного класса. Он включает в себя два метода, а именно:

```
@staticmethod
connect(url, login, md5_hexpass, application_id)
```

Данный метод устанавливает соединение с указанным VDOM сервером, а точнее с конкретным приложением, установленным на сервере, и возвращает экземпляр класса **VDOMService**, готовый к работе. Метод имеет следующие параметры:

- **url** – IP адрес VDOM сервера.
- **login** – логин для входа на сервер.
- **md5_hexpass** – md5-хеш пароля для входа на сервер.
- **application_id** – GUID приложения, с которым выполняется соединение.

- **call**(self, container_id, action_name, xml_data)

Данный метод осуществляет вызов удаленной процедуры и возвращает XML данные, являющиеся результатом ее вывода. Рассмотрим его параметры:

- **container_id** – GUID контейнера, содержащего удаленную процедуру.
- **action_name** – имя удаленной процедуры.

- **xml_data** – XML данные, подаваемые на вход удаленной процедуры.

В соответствии со спецификацией заказчика в результате проделанной работы было реализовано и протестировано 7 удаленных процедур, а именно:

- **login** – выполнить авторизацию пользователя в системе.
- **create_user**– создать пользователя.
- **get_acl** – взять права доступа.
- **set_acl** – установить права доступа.
- **create_folder** – создать папку.
- **list** – получить все доступные папки и архивы.
- **put_document** – добавить документ в архив документов.

На рисунке 7 приведен фрагмент исходного кода одной из операций.

```
18
19 if "xml_data" in args:
20     try:
21         user = User.current()
22         if user:
23
24             data = args["xml_data"]
25
26             from xml.dom.minidom import parseString
27             dom_param = parseString(data.encode("UTF-8"))
28
29             share_guid = filename = base64_data = metafields = ""
30
31             guid = dom_param.getElementsByTagName("share_guid")
32             if guid and guid[0].firstChild:
33                 share_guid = guid[0].firstChild.nodeValue
34
35             fname = dom_param.getElementsByTagName("filename")
36             if fname and fname[0].firstChild:
37                 filename = fname[0].firstChild.nodeValue
38
39
40             text = dom_param.getElementsByTagName("text")
41             if text and text[0].firstChild:
42                 text = text[0].firstChild.nodeValue
43             else:
44                 text = None
45
46
47             data = dom_param.getElementsByTagName("data")
48             if data and data[0].firstChild:
49                 base64_data = data[0].firstChild.nodeValue
```

Рисунок 7 - Фрагмент исходного кода операции put_document

4.3 Пользовательский интерфейс

В данном подразделе приводится описание процесса реализации web-интерфейса для приложения. В пункте 4.3.1 описан общий поход к решению данной задачи; в пункте 4.3.2 рассмотрен пример разработки одной из страниц web-интерфейса.

4.3.1 Web-интерфейс

Итак, для каждой отдельной страницы необходимо было разработать как *визуальную*, так и *поведенческую* части. Рассмотрим решение двух данных задач.

Визуальная часть разрабатывалась за счет размещения на каждой странице объектов и задания значений их атрибутам. В некоторых местах для задания изощренного оформления, использовались стили CSS [10]. Различные, чисто декоративные элементы оформления, такие, как например, логотип, иконки, рисунки, шапка сайта, использованные при оформлении пользовательского интерфейса, были разработаны дизайнером компании VDOM Box Research. Для задания визуальной составляющей страниц, использовалась интегрированная среда разработки VDOM IDE.

Поведенческая часть разрабатывалась за счет использования технологии event->action. Для построения схемы взаимодействия размещенных на страницах объектов также использовалась VDOM IDE. Для написания логики обработчиков событий (action-ов) использовалась VDOM Script IDE, так как функциональность редактора скриптов VDOM IDE оказалась довольно-таки ограниченной.

Всего в результате проделанной работы в соответствии со спецификацией заказчика было разработано 17 web-страниц:

- Administration
- Custom_search
- Custom_search_edit
- Custom_searches_acl
- Document_share
- Document_share_edit
- Documents_share
- Folder

- Folder_edit
- Group
- Home_Page
- Reset_password
- Retrieve_password
- Search
- Searches
- User
- Users_and_groups.

В пункте 3.3.2 рассматривается пример разработки одной из страниц web-интерфейса.

4.3.2 Пример реализации одной из web-страниц

Рассмотрим процесс разработки web-интерфейса на примере страницы Document_share_edit, внешний вид которой можно увидеть на рисунке 8. Данная страница позволяет пользователю осуществлять резервирование товаров в соответствии с конкретным заказом. Рассмотрим процесс разработки визуальной и поведенческой частей web-страницы.

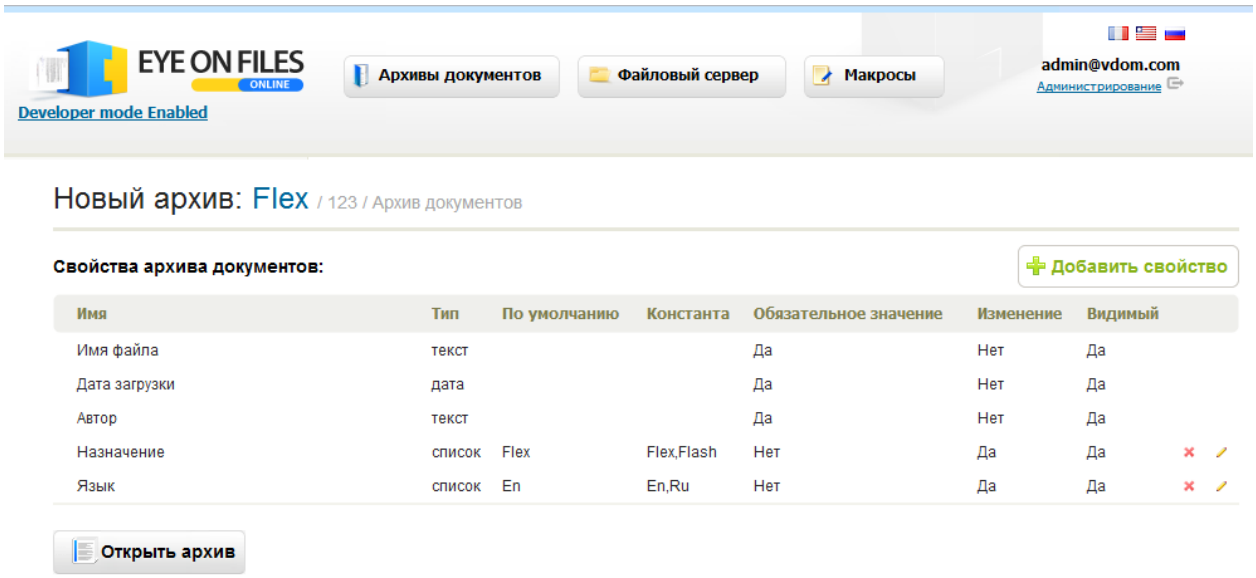

Рисунок 8 - Внешний вид страницы Document_share_edit

Для задания визуальной составляющей страницы были использованы следующие типы объектов:

- **RichText**. Используется для отображения текстовых данных. Способен распознавать HTML теги.
- **Button**. Представляет собой такой стандартный элемент управления как кнопка.
- **Container**. Предназначен для логического и визуального объединения нескольких объектов.
- **Copy**. Позволяет «копировать» определенный объект, расположенный на другой странице. Используется в случаях, когда необходимо визуально продублировать один объект на многих страницах (например, заголовок, общий для нескольких страниц).
- **Datatable**. Используется для вывода данных в табличной форме. Способен отслеживать нажатия пользователя на строки таблицы.
- **Growl**. Предназначен для выдачи уведомлений в виде всплывающего окна.

Объект Copy используется для размещения заголовка вверху страницы (заголовок заранее скомпонован и хранится на отдельной странице Design). Объекты типа RichText используются для нанесения декоративных и пояснительных надписей. Верхнее окно с информацией реализовано с помощью объекта Container, содержащего некоторые другие объекты.

ЗАКЛЮЧЕНИЕ

В ходе дипломной работы было разработано Web приложение EOF, которое имеет возможность хранить Архивы документов, совершать поиск, давать различные права доступа пользователям и группам к папкам и архивам. В кроме того приложение EOF доступно для использования в других приложениях. Для реализации этого продукта было выбрана поисковое решение Xapian и разработаны VDOM компоненты позволяющие приложению взаимодействовать с Xapian, так же разработана база данных, слой бизнес логики, Web-интерфейс и API приложения. Кроме этого были разработаны VDOM компоненты, которые используются не только в этом приложении, а в других приложениях на базе VDOM.

ЛИТЕРАТУРА

1. Бабанов А.М. – Технология разработки программного обеспечения: структурный подход. – Томск: Изд-во НТЛ, 2006. – 220 с.
2. Дейт К.Дж. Введение в системы баз данных. – Вильямс, 2006. 1328 с.
3. Fowler M. Patterns of Enterprise Application Architecture. – Addison Wesley, 2004. –P. 523
4. ORM. Коротко и ясно. [Электронный ресурс]. – 2009. – Режим доступа: http://brotkin.ru/2009/02/01/orm/, свободный
5. Ларман К. Применение UML и шаблонов проектирования. – Издательский дом «Вильямс», 2004. 620 с.
6. Korboulewsky N. V.D.O.M. BOX. Technology V.D.O.M. / V.D.O.M. v2 – W.H.O.L.E. v1- EI.V.D.O.M v1. – 7 rue Saint Henri 31000 Toulouse, 2007. – P. 76
7. Korboulewsky N. V.D.O.M. Technology / Event System. – 7 rue Saint Henri 31000 Toulouse, 2008. – P. 21
8. Python v2.7.2 documentation. [Электронный ресурс]. – 2011. – Режим доступа: http://docs.python.org/, свободный
9. Rossum G., Drake F.L. An Introduction to Python. - Network Theory Ltd., 2003. – P. 454
10. Справочник по CSS. [Электронный ресурс]. – 2010. – Режим доступа: http://css.manual.ru/, свободный

ПРИЛОЖЕНИЕ А. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

При открытии приложение необходимо ввести логин и пароль. (рисунок 9)

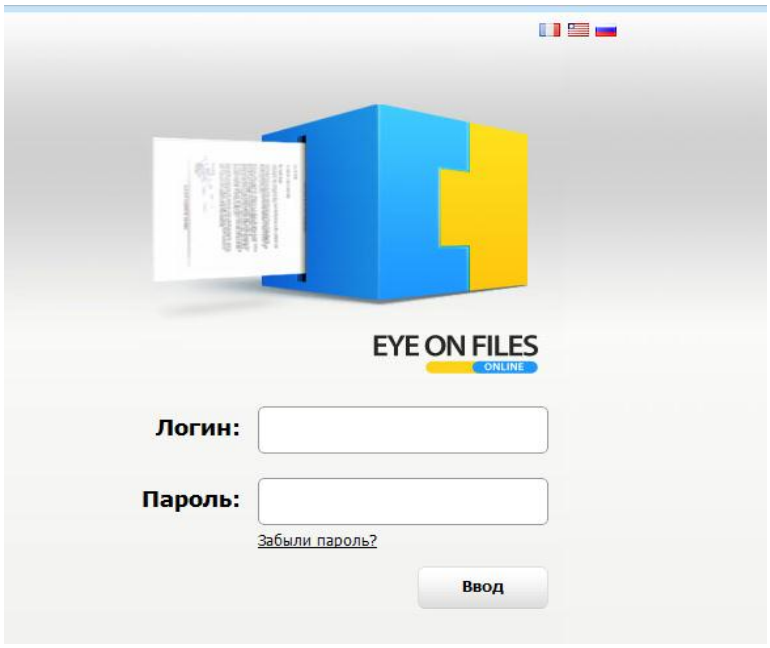

Рисунок 9 - Вход в приложение

На странице Архив документов представлены доступные пользователю папки. (рисунок 10)

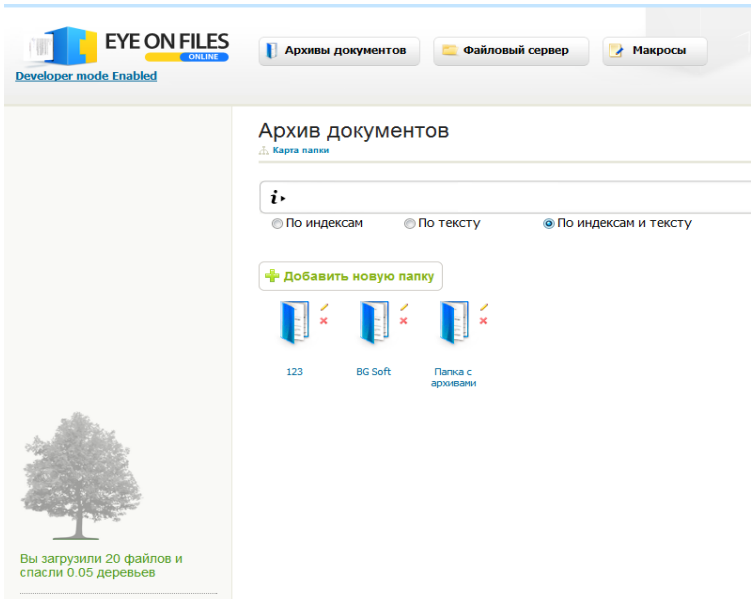

Рисунок 10 - Вид Архивов документа

В из папки есть возможность выбрать доступный архив, рисунок 11.

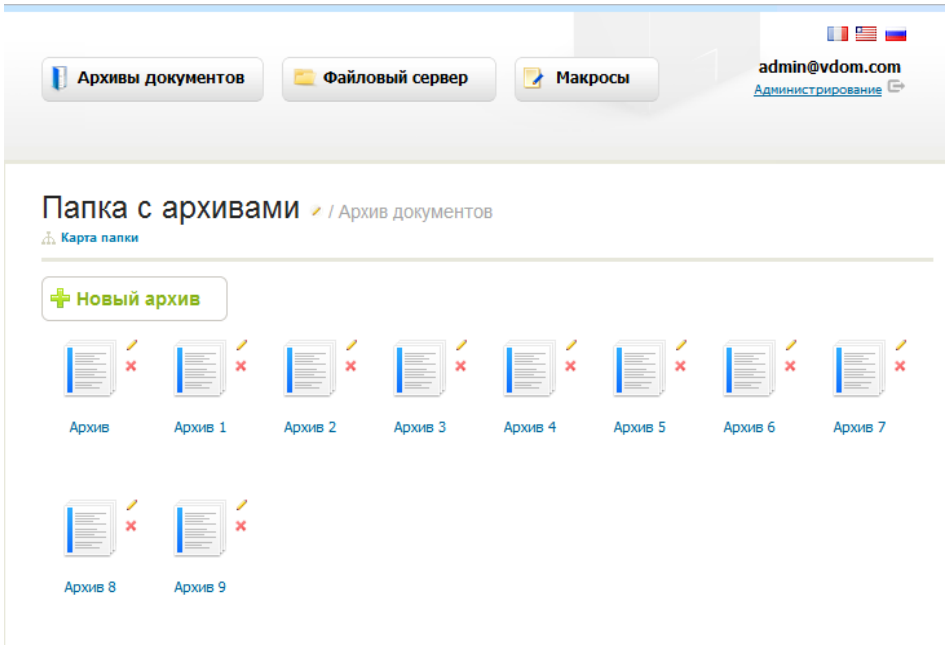

Рисунок 11 - вид архива документа

В архиве можно добавлять файл, удалять, скачивать и отправлять по электронной почте, рисунок 12.

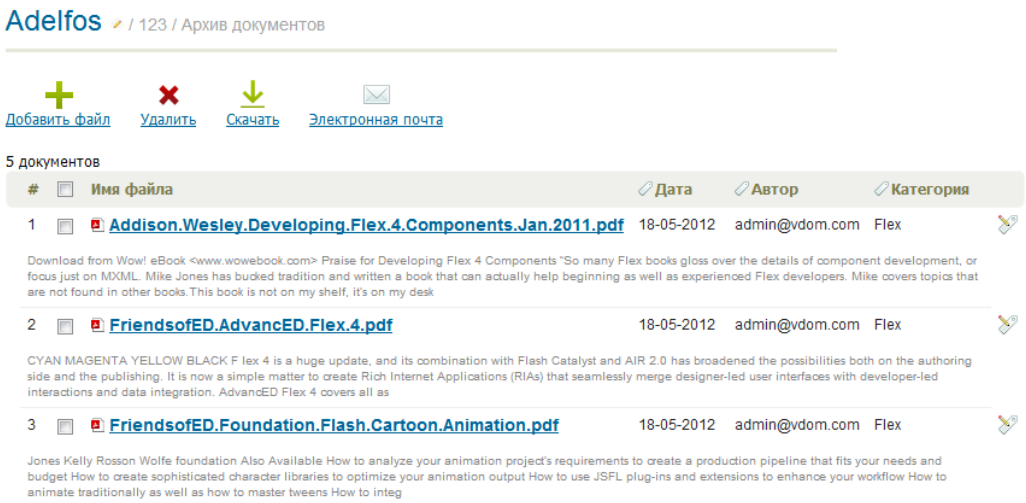

Рисунок 12 - вид архива

Также доступен поиск по содержимому рисунок 13

Рисунок 13 - Вид результата поиска