Пермский филиал федерального государственного автономного
образовательного учреждение высшего образования
Национальный исследовательский университет
«Высшая школа экономики»

Факультет экономики, менеджмента и бизнес-информатики

Симонова Наталья Андреевна

# ИНДЕКСАЦИИ ХРАНИЛИЩА ТЕКСТОВ НА ОСНОВЕ ЕСТЕСТВЕННО-ЯЗЫКОВОЙ АДРЕСАЦИИ

*Выпускная квалификационная работа*

студента образовательной программы «Программная инженерия»
по направлению подготовки *09.03.04 Программная инженерия*

Рецензент
к.ф.-м.н., доцент, доцент кафедры
информационных технологий ФГБОУ ВО
«Пермский государственный
национальный исследовательский
университет»
Л.А. Залогова

Руководитель
к.ф.-м.н., доцент, доцент
кафедры
информационных
технологий в бизнесе
_____
Л.Н. Лядова

Пермь, 2018 год

# Аннотация

В данной работе представлены результаты реализации модуля программной системы для проведения лингвистических исследований, обеспечивающего хранение, поиск и получение текстов из корпусов с использованием индексации на основе естественно-языковой адресации в виде wcf-сервиса. Были проанализирован подход к хранению корпусов в существующих программных продуктах, а также документо-ориентированные СУБД для хранения текстовой информации. Анализ показал необходимость разработки специального модуля с применением индексации. Во второй главе приведен анализ проектирования баз данных и архитектуры, а также технико-экономическое обоснование. Далее описаны реализация, тестирование и публикация wcf-сервиса на Azure.

Данная работа будет интересна специалистам, которые занимаются разработкой документно-ориентированных систем, хранилищ неструктурированной или слабоструктурированной информацией большого объема.

Работа включает 48 страниц основного текста, 31 иллюстрацию, 17 таблиц, 4 приложения. Библиографический список – 27 наименований.

2018 г. Кафедра информационных технологий в бизнесе.

# Оглавление

# Введение

Соответствие научному стилю – один из показателей качества научных публикаций. Однако при предоставлении результатов исследований на иностранном языке данное требование может вызывать затруднение у авторов, не являющихся носителями языка. Изучение и формализация критериев оценки соответствия текста научному стилю может помочь упростить данную задачу для авторов. Изучением данной проблемы занимается лингвистика и её подраздел – стилистика, однако выявление критериев и проверка текста на соответствие им могут быть автоматизированы с помощью специально разработанных программных средств [1].

С развитием вычислительной техники появилась возможность автоматически обрабатывать большие массивы документов, поэтому своё развитие получила корпусная лингвистика, которая исследует особенности языка на основе корпусов – наборов текстов, собранных по определенному критерию, например, единому стилю или автору [6]. На основе таких наборов можно выявлять закономерности языка, собирая статистические данные. Такие закономерности или стандартные приемы были выявлены и для научного стиля в английском языке. Примером такой закономерности может быть более частое использование пассивного залога, чем в общей речи [26]. Однако данный вопрос только решается лингвистами и исследование не завершено.

Современные корпусные исследования требуют анализа большого объема текстов, поэтому лингвистам необходимы инструменты, которые, во-первых, позволяют хранить и обрабатывать большие объемы информации, так как при обработке текста необходимо хранить его первоначальное представление, аннотацию, плоский текст и т.д., а во-вторых, предоставляют совместный доступ к корпусам для разделения работы между несколькими лингвистами.

Существующие программные продукты (GATE, AntConc и др.) созданы для использования только специалистами-лингвистами и не пригодны для использования непрофессионалами. GATE представляет собой группу программных продуктов, включая настольные, облачные, серверные версии с возможностью расширения функциональности через плагины. Однако параллельная работа с корпусами возможна только, если пользователь устанавливает персональный сервер для хранения данных, а

облачная версия может использоваться только как механизм распределенной обработки в виде сервиса [10, 18].

Сотрудниками кафедры информационных технологий и департамента иностранных языков и студентами выполняется исследование в рамках научно-учебной группы «Разработка программного обеспечения для проведения корпусных исследований английского языка», одной из целей которой является создание подобного программного продукта. Разрабатываемый программный продукт может быть использован для количественного анализа научной речи, а также имеет web-интерфейс для обеспечения простого и совместного доступа, а все его компоненты является web-сервисами.

С помощью разрабатываемой системы не только лингвисты смогут продолжать корпусные исследования загруженных корпусов, но и в перспективе студенты могут перед отправлением статьи на рецензию экспертам получить рекомендации на основе проверки соответствия статистическим данным.

Для обеспечения таких возможностей данным программам необходимо хранить как отдельные тексты, загружаемые студентами, так и целые корпусы для их дальнейшего исследования. Иногда корпусы могут достигать значительного объема, и поэтому необходимы эффективные алгоритмы поиска по документам, так как в неструктурированных данных поиск может обернуться последовательным перебором. Одним из возможных решений является использование индексации на основе естественно-языковой адресации, которая подразумевает использование цифровых кодов букв в качестве индекса [2, 24].

Для хранения текстовых документов может быть использовано одно из существующих решений – документно-ориентированная СУБД, обеспечивающая более простую работу с неструктурированными данными.

Объектом работы является хранение корпусов текстов. Предметом – индексация данных корпусов на основе естественно-языковой адресации.

Целью работы является разработка модуля для хранения, поиска и получения текстов из корпусов с использованием индексации на основе естественно-языковой адресации в виде web-сервиса.

В соответствии с целью были сформулированы следующие задачи:

1. Проанализировать существующие технологии хранения больших массивов текстовой информации, протоколы передачи, подходы к поиску и индексации, в том числе изучить принцип индексации на основе естественно-языковой адресации, разработать техническое задание.
2. Произвести проектирование web-сервиса для хранения корпусов:
   2.1. Произвести проектирование вариантов использования сервиса для выявления функциональных требований.
   2.2. Произвести моделирование статической структуры системы.
   2.3. Произвести проектирование архитектуры системы.
3. Реализовать web-сервис для хранения корпусов:
   3.1. Реализовать CRUD-функции для корпусов и документов.
   3.2. Реализовать индексацию на основе естественно-языковой адресации.
   3.3. Произвести функциональное и интеграционное тестирование.

Разрабатываемый программный продукт основывается на сервисно-ориентированной архитектуре. При решении поставленных задач при этом используются методы объектно-ориентированного проектирования и программирования, а также теоретико-множественные методы.

Теоретическая значимость данной работы заключается в исследовании моделей хранения текстов, а также нового применения индексации на основе естественно-языковой адресации.

Практическая значимость заключается в создании компонента-хранилища, который обеспечивает эффективную работу при анализе больших массивов информации, в качестве части исследовательского инструмента для количественного анализа научной речи.

Работа состоит из четырех глав. В первой главе описываются результаты анализа предметной области и существующих решений, требования к системе. Во второй главе приводится описание результатов моделирования компонента. В следующей главе описывается реализация, а в четвертой – результаты тестирования.

# Глава 1. Задача хранения корпусов текстов

На основе анализа проблемы хранения и обработки корпусов текстов, в том числе с помощью оценки существующих решений по их подходу к хранению информации, необходимо выбрать подходы и технологии для реализации сервиса, а также сформулировать требования к предлагаемому решению. Также необходимо оценить возможность применения технологии индексации текстов на основе естественно-языковой адресации.

Для сравнения инструментов по их подходу к хранению информации выбраны следующие критерии:

1. Возможность хранения больших объемов данных, которая становится всё более важной для задач корпусной лингвистики.
2. Скорость доступа к данным, которая должны быть высокой, так как корпусы и их метаинформация могут достигать большого объема.
3. Формат данных, так как открытость используемых структур является преимуществом для современных систем.
4. Объем дополнительной памяти, которая используется для хранения индексов. Помимо самой информации для ускорения доступа к ней необходимо хранить индексы, которые могут так же занимать большое количество памяти, что является явным недостатком в условиях ограниченности ресурсов.

## 1.1. Существующие решения для задач корпусной лингвистики

Современные задачи корпусной лингвистики требуют больших временных затрат, поэтому при хранении становится важным не только возможность хранения больших объемов неструктурированных текстовых данных, но и скорость доступа к ним. Также преимуществами системы для работы с корпусами можно назвать открытость используемых структур при хранении данных.

Существует большое количество программных систем для корпусных исследований. Некоторые из них, например AntConc [10], предоставляют различные возможности для исследования текста, однако являются настольными, не имеют встроенных корпусов и возможности многопользовательской работы. Другие приложения для корпусных исследований имеют web-интерфейс, однако большинство

из них закреплено за одним или несколькими корпусами (CQPweb [19]). Также существуют бесплатные библиотеки (Corsis [13]) с предоставляемыми программами с пользовательским графическим интерфейсом. Эти программы могут быть использованы только для работы с корпусами пользователя, расположенными на локальном компьютере.

### 1.1.1. Приложение AntConc

Настольное приложение AntConc не использует базы данных для хранения корпусов, а работает с неаннотированными текстами в текстовом формате, загружая их сразу в оперативную память. Для хранения некоторых результатов работы, например ключевые слова в формате KWIC [23], AntConc использует реляционную СУБД SQLite, без использования какого-либо индексирования. Позиционируя себя как приложения для работы с небольшими корпусами, AntConc не имеет возможности работать с корпусами большого объема из-за использования оперативной памяти, а поиск по документам не ускоряется за счет индексации [10].

С помощью AntConc можно составлять списки слов и словоформ, проводить построение конкорданса, искать коллокации, составлять частотные словари и др. Таким образом, AntConc является удобной настольной программой для работы с корпусами небольшого размера, однако не подразумевает совместную работу.

### 1.1.2. Приложение CQPweb

Приложение CQPweb построена на двух отдельных технологиях: IMS Open Corpus Workbench и реляционной базе данных MySQL. IMS Open Corpus Workbench (CWB) – это набор программных инструментов с открытым исходным кодом для управления большими корпусами и созданием запросов. CWB использует специальную модель данных (см. рис. 1.1) и основанный на ней открытый XML формат данных, а документы в данном формате, после обработки помещаются в базу данных MySQL [19].

Главным преимуществом CQPweb является его гибкость; его более обобщенная модель данных делает его совместимым с любым корпусом. Варианты анализа, доступные в CQPweb, включают: согласование; коллокации; таблицы и диаграммы распределения; списки частот; и ключевые слова или ключевые теги. Система CQPweb же позволяет делать различные запросы к 45 готовым корпусам через веб-интерфейс, однако она не предоставляет возможности работы с новыми корпусами.

| corpus position | word form | ID | part of speech | ID | lemma | ID |
|---|---|---|---|---|---|---|
| (0) | <text> value = “id=42 lang="English"” | | | | | |
| (0) | <text_id> value = “42” | | | | | |
| (0) | <text_lang> value = “English” | | | | | |
| (0) | <s> | | | | | |
| 0 | An | 0 | DET | 0 | a | 0 |
| 1 | easy | 1 | ADJ | 1 | easy | 1 |
| 2 | example | 2 | NN | 2 | example | 2 |
| 3 | . | 3 | PUN | 3 | . | 3 |
| (3) | </s> | | | | | |
| (4) | <s> | | | | | |
| 4 | Another | 4 | DET | 0 | another | 4 |
| 5 | very | 5 | ADV | 4 | very | 5 |
| 6 | easy | 1 | ADJ | 1 | easy | 1 |
| 7 | example | 2 | NN | 2 | example | 2 |
| 8 | . | 3 | PUN | 3 | . | 3 |
| (8) | </s> | | | | | |
| (9) | <s> | | | | | |
| 9 | Only | 6 | ADV | 4 | only | 6 |
| 10 | the | 7 | DET | 0 | the | 7 |
| 11 | easiest | 8 | ADJ | 1 | easy | 1 |
| 12 | examples | 9 | NN | 2 | example | 2 |
| 13 | ! | 10 | PUN | 3 | ! | 8 |
| (13) | </s> | | | | | |
| (13) | </text_lang> | | | | | |
| (13) | </text_id> | | | | | |
| (13) | </text> | | | | | |

***Рисунок 1.1. Пример модели данных CWB***

### 1.1.3. Библиотека Corsis

Corsis – это бесплатная библиотека для анализа корпусов с открытым исходным кодом, написанная на языке C#. Также бесплатно предоставляется кросс-платформенный графический интерфейс для работы с библиотекой. Для хранения документов данная библиотека использует XML-основанные файлы. Программа, основанная на данной библиотеке, не использует базу данных, а работает напрямую с XML-основанными файлами, загружая их сразу в оперативную память [13].

Библиотека и программа Corsis являются простыми и открытыми инструментами для простого анализа небольших корпусов, например, составления списка слов и конкорданса.

### 1.1.4. Семейство GATE

Описанные выше приложения, не считая библиотек, не имеют интерфейсов прикладного уровня, то есть не могут быть настроены под определенные задачи. Таким функционалом, помимо широких встроенных функций для обработки текстов, обладает GATE, один из существующих инструментов для корпоративных исследований. Семейство GATE – это группа программного обеспечения, включающая настольные,

облачные, серверные версии с возможностью расширения функциональности через плагины [14, 18].

В продуктах GATE вся информация о документах хранится в открытых форматах XML или OWL в специальных базах данных с реализацией индексов MG4J [15]. MG4J является еще одним решением для полнотекстового поиска в больших документах с использованием методов сжатия с инвертированным индексом [12]. Основным ограничением такого подхода по-прежнему можно назвать память, необходимую для индексов, хотя в современных версиях продуктов GATE она не столь критична [15]. Тем не менее, необходимый объем памяти для реализации MG4J больше, чем гипотетический необходимый объем памяти для индексации на основе естественно-языковой адресации.

GATE Teamware (рис. 1.2) – сетевая программная система с открытым исходным кодом для совместной работы над корпусными исследованиями. Однако параллельная работа с корпусами требует установки пользователем персонального сервера для хранения данных и дополнительного администрирования, а облачная версия используется только как сервис распределенной обработки [11].

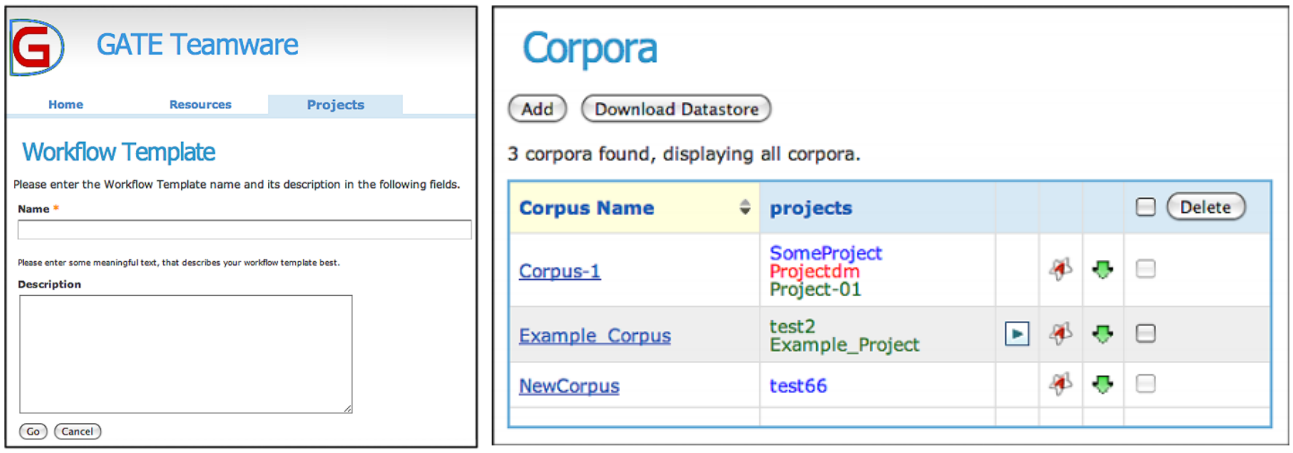


***Рисунок 1.2. GATE Teamware***

Таким образом, семейство GATE включает в себя многофункциональные мультиплатформенные инструменты для разноплановой работы с корпусами, однако из-за универсальности программы пользовательский интерфейс не является достаточно удобным и интуитивно понятным.

### 1.1.5. Сравнение инструментов

В таблице 1.1 приведено сравнение подходов к хранению корпусов и текстов у существующих решений по ранее выделенным критериям.

*Таблица 1.1. Сравнение существующих решений*

| **Критерий** | **AntConc** | **CQPweb** | **Corsis** | | **Gate** |
|---|---|---|---|---|---|
| | | | **Библиотека** | **Программа** | |
| Возможность хранения больших объемов данных | Нет, так как не использует БД, загружает в оперативную память | Да | Является библиотекой | Нет, так как не использует БД, загружает в оперативную память | Да |
| Скорость доступа к данным | Высокая, так как работа происходит в оперативной памяти | Достаточно высокая за счет использования CWB | | Высокая, так как работа происходит в оперативной памяти | Достаточно высокая за счет использования MG4J |
| Формат данных | Только txt | XML | XML | | XML или OWL |
| Объем дополнительной памяти | Дополнительная память не нужна | Нужна дополнительная память для модели CWB | Является библиотекой | Дополнительная память не нужна | Нужна дополнительная память для индексов MG4J |

Приложение CQPweb может быть использовано только для работы со встроенными, заранее обработанными корпусами, поэтому самым эффективным решением является семейство GATE. Однако и GATE, и CQPweb задействуют дополнительную память для ускорения доступа к данным.

## 1.2. Инструменты для решения задач хранения текстовых документов

В связи с необходимостью хранить и обрабатывать большое количество текстовой информации, было принято решение использовать документо-ориентированную СУБД (один их NoSQL подходов), так как она обеспечивает более простую работу с неструктурированными текстовыми данными, а также многие NoSQL базы специально разработаны для хранения крупных объемов данных с более простой поддержкой масштабирования.

Технология NoSQL описывает несколько нереляционных подходов к базам данных, одним из которых является и документо-ориентированный. Преимуществами данных подходов являются гибкость схемы, легкая масштабируемость и эффективность разработки [9].

Документо-ориентированные базы данных хранят и обрабатывают документы – иерархические самоописываемые структуры данных, обычно хранящиеся в форматах XML, JSON, BSON и др. Такие базы данных хранят в себе агрегаты данных, состоящих из скалярных значений, ассоциативных массивов и коллекций. Документы в таких базах данных хранятся в качестве значений, доступных по уникальному ключу, однако

открытость структуры позволяет создавать запросы и к части содержимого документа, не загружая его полностью [9].

Основными представителями таких СУБД можно назвать MongoDB, OrientDB, а также предоставляемую порталом Azure CosmosDB и другие.

### 1.2.1. СУБД MongoDB

Одной из самых популярных документно-ориентированных СУБД является MongoDB, предоставляющая широкие возможности формирования запросов с помощью специального языка, хранения документов в формате JSON или BSON и индексацию. Данная СУБД поддерживает работу на кластерах благодаря специальному механизму асинхронной синхронизации (репликация), а так же обеспечивает масштабируемость, в том числе шардинг [9]. MongoDB является бесплатным продуктом с открытым исходным кодом, написанным на языке C++.

Для работы с базой MongoDB предоставляет специальный язык запросов, позволяющий добавлять, удалять и изменять документы, а также искать документы по некоторым условиям [9].

### 1.2.2. СУБД OrientDB

OrientDB – это свободно распространяемая СУБД, объединяющая в себе документо- и графо-ориентированные СУБД. OrientDB, как и любая другая документо-ориентированная СУБД, позволяет хранить и обрабатывать документы, но также OrientDB поддерживает хранение отношений между документами. Данная СУБД хранит документы в формате JSON и дает возможность проводить некоторые запросы к базе на SQL и с помощью Gremlin.

Данная СУБД имеет малый размер и обеспечивает простую установку, а новый подход к связям (с помощью указателей) позволяет увеличить скорость доступа к данным. Также OrientDB СУБД имеет REST-архитектуру, поддерживает трензакции и некоторую другую функциональность [9].

### 1.2.3. СУБД Azure Cosmos DB

В мае 2017 года появилось новое решение Azure Cosmos DB – база данных как сервис, предоставляемая Azure для студентов бесплатно по специальной подписке, которая расширяет предыдущий проект DocumentDB. Данная СУБД поддерживает

различные варианты NoSQL, в том числе и документно-ориентированный. С помощью предоставляемого API с данными в Cosmos DB можно работать как посредством SQL и LINQ, так и посредством языка MongoDB и др.

Cosmos DB обеспечивает высокую масштабируемость, гарантируя при этом предсказуемый уровень производительности без дополнительного управления, а также может прозрачно реплицировать данные с настраиваемым уровнем согласованности для нахождения нужного компромисса между согласованностью и производительностью [3].

### 1.2.4. Сравнение документо-ориентированных СУБД

Так как все рассмотренные СУБД имеют все необходимые функции , то их сравнение было проведено по нефункциональным критериям:

1. Поддержка различных языков запросов.
2. Простота развертывания на сервере.
3. Цена доступа.
4. Простота взаимодействия с СУБД.

Результаты сравнения представлены в таблице 1.2.

***Таблица 1.2. Сравнение документо-ориентированных СУБД***

| | **Поддержка различных языков запросов** | **Простота развертывания на сервере** | **Цена доступа** | **Простота взаимодействия с СУБД** |
|---|---|---|---|---|
| **MongoDB** | - | - | + | - |
| **OrientDB** | ± | - | + | ± |
| **Cosmos DB** | + | + | ± | + |

Для разрабатываемого сервиса была выбрана Cosmos DB в качестве СУБД для хранения документов, так как данная СУБД уже является сервисом, не требует сложных настроек, поддерживает различные языки, хоть и предоставляется только по подписке.

## 1.3. Индексация на основе естественно-языковой адресации

Несмотря на то, что современные документо-ориентированные СУБД имеют инструменты для эффективного поиска данных, доступных по ключам, полнотекстовый поиск может иметь недостаточную производительность. Последовательное чтение текстового файла (чтобы найти конкретное ключевое слово) является медленной операцией, и каждый индексированный подход будет быстрее [5, 24].

Таким образом, применение индексации на основе естественной языковой адресации должно повысить производительность СУБД для полнотекстного поиска, однако, как и для любой индексации, это потребует дополнительных ресурсов: памяти для хранения дополнительной информации и времени на обработку. Главным преимуществом данного подхода является отсутствие необходимости хранения дополнительных индексов, так как индексами служат сами ключевые слова, то есть если стандартная схема поиска информации выглядит как «Имя-Адрес-Маршрут» (Что мы ищем? Где мы ищем? Как туда попасть?), то при использовании индексации на основе естественной языковой хранить адрес нет необходимости (так как он совпадает с именем), и схема сокращается до «Имя-Маршрут» [24].

При использовании индексации на основе естественно-языковой адресации необходима специальная модель данных, которой была выбрана многодоменная информационная модель, основанная на многоуровневых множествах. Доступ до информации, хранимой для слова “beer” представлен на рисунке 1.3 [24].

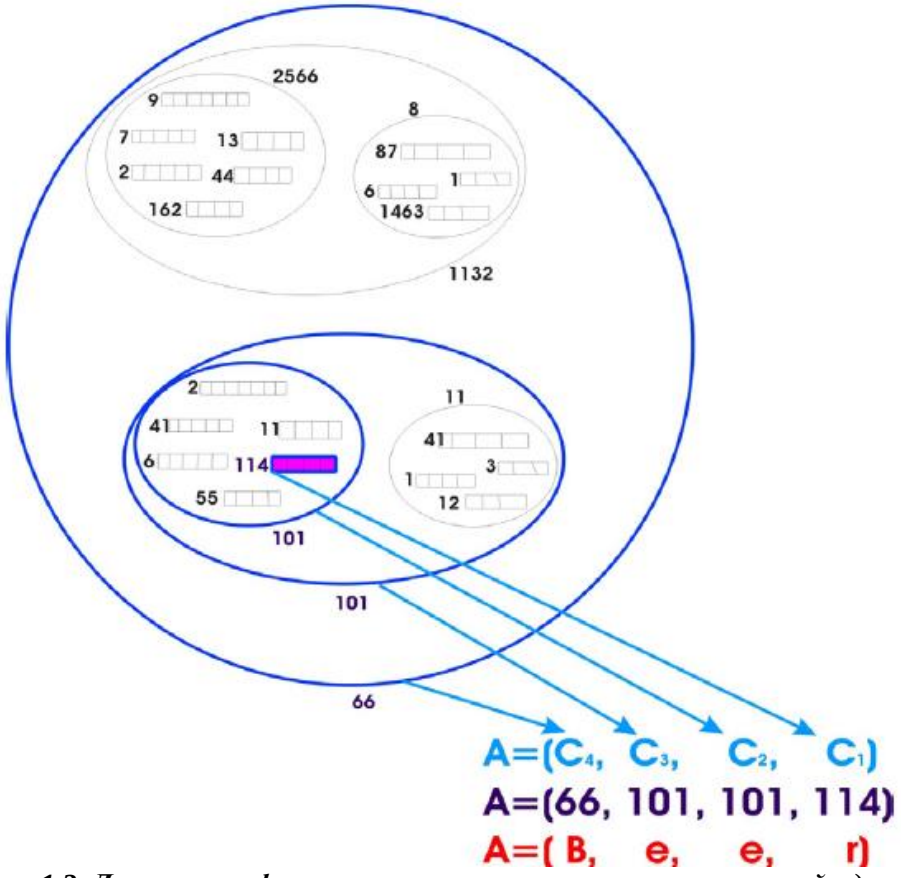


***Рисунок 1.3. Доступ к информации на основе естественно-языковой адресации***

Индексации на основе естественно-языковой адресации подразумевает использование цифровых кодов букв в качестве индекса. При этом каждый символ

строки используется как путь в специальной многомерной структуре на основе дерева, где каждый узел содержит массив указателей на каждую букву алфавита (на каждый цифровой код буквы) или на подстроку. Указатели же содержат либо искомое значение, либо ссылку на другое дерево по цифровому коду соответствующего символа.

Рассматривались различные структуры для основы реализации индексации на основе естественно-языковой адресации, в том числе префиксные деревья и многоуровневые хеш-таблицы. Самым быстрым и удобным решением было выбрано использование многоуровневой хэш-таблицы с ключом фиксированной длины.

Так как средняя длина самых употребляемых слов в английском языке составляет около 5 букв, а максимальная – 15 букв [24], то вложенность таблиц будет не большая при длине ключа 3-5 символов.

Одним из отличий данной структуры является то, что деревья разработаны для хранения информации в ограниченной оперативной памяти, а для реализации естественно-языковой адресации была создана подобная структура для хранения на постоянном запоминающем устройстве [24].

Естественно-языковая адресация применялась в графовых базах данных или RDF-репозиториях, основанных на тройках данных (объект-связь-субъект) [24].

Применение индексации в системе ICON на основе естественно-языковой адресации показало следующие преимущества:

- Линейная алгоритмическая сложность.
- Уменьшение необходимой памяти, так как нет необходимости хранить индексы отдельно [22].

## 1.4. Технология Windows Communication Foundation

Модуль для хранения корпусов и текстов реализован с помощью технологии Windows Communication Foundation (WCF), которая является основой для создания web-сервисов. В отличие от обычного сервиса, WCF позволяет иметь несколько точек доступа, поддерживает сессии на уровне сервиса, а не метода, позволяет использовать различные протоколы и предоставляет некоторые другие расширенные возможности. В качестве хостинга для WCF-сервиса может выступать любое приложение [27].

Для работы с данными используются пользовательские классы, отражающие предметную область, поэтому при создании сервиса необходимо определить не только

контракты операции, но и контракты данных для передачи информации в качестве объектов.

## 1.5. Технология Entity Framework

При работе с реляционной базой данных используется технология ORM Entity Framework, которая позволяет работать с облачной базой данных SQL Azure [17], как с локальной. Использование Entity Framework обусловлено рядом преимуществ:

1. Независимость базы данных, то есть нет необходимости писать код, специфичный для конкретной базы данных.
2. Поддержка LINQ (строгая типизация, проверка запросов на этапе компиляции).
3. Быстрота написания запросов.
4. Поддержка транзакций и параллелизма [20].

Также Entity Framework сохраняет достойную производительность на небольших и средних моделях при выполнении определенных рекомендаций [7]. Так как реляционная модель данных является небольшой, то возможная потеря производительности не является критически важным фактором.

## 1.6. Требования к сервису-хранилищу текстов

На основе проведенного анализа сформированы требования к разрабатываемой программе:

1. Сервис должен быть опубликован на облачной платформе Azure.
2. Сервис должен обеспечивать взаимодействие с другими сервисами посредством сети Интернет с помощью разработанных стандартов взаимодействия.
3. Для индексации хранимых документов должны использоваться индексация на основе естественно-языковой адресации.
4. В качестве нереляционной СУБД должна быть использована документо-ориентированная Azure Cosmos DB.
5. В качестве нереляционной СУБД должна быть использована SQL Azure с помощью технологии Entity Framework.

6. Источниками входных данных являются другие сервисы, реализованные в программной системе. Должны быть разработаны стандарты взаимодействия между сервисами.
7. Выходные данные должны посылаться так же в другие сервисы согласно разработанным стандартам взаимодействия, реализованные в программной системе.
8. Данная программа должна быть совместима с другими сервисами программной системы, посредством поддержки стандартов взаимодействия.
9. Функциональные требования представлены на диаграмме прецедентов (рис.1.4).

Полное описание прецедентов приведено в приложении А.

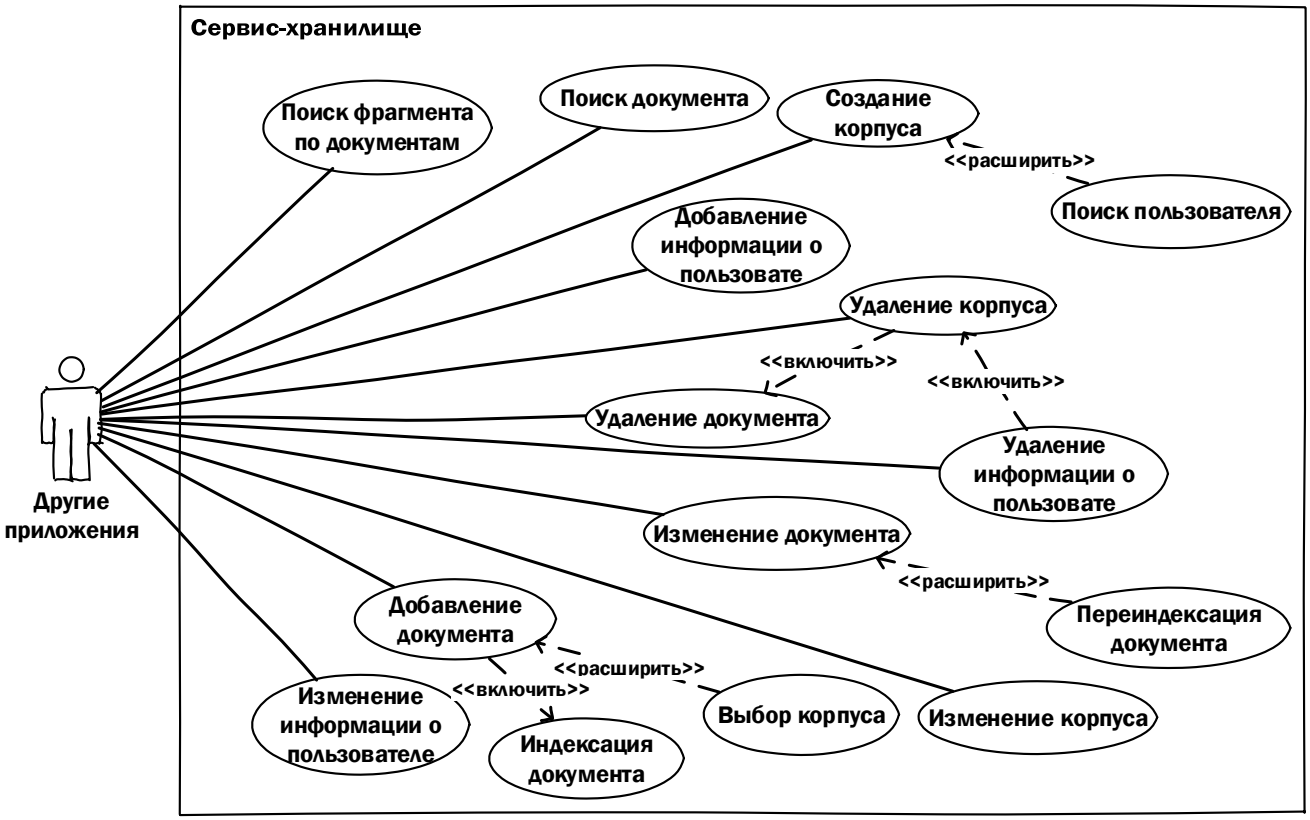


***Рисунок 1.4. Диаграмма прецедентов***

Сервис должен обеспечить возможность выполнения перечисленных ниже функций:

1. Создание/удаление/изменение корпусов текстов.
2. Добавление/удаление/изменение документов в корпусы.
3. Добавление информации о пользователе (регистрация пользователя).
4. Удаление/изменение информации о пользователе.

5. Получение информации о пользователе.
6. Получение информации о документе.
7. Получение информации о корпусе.
8. Поиск документов в базе.
9. Поиск документов по ключевым словам.

Полные требования к системе приведены в приложении В.

# Глава 2. Проектирование хранилища текстов

На этапе проектирования сервиса описываются алгоритмы всех функций, разрабатывается архитектура системы, проектируются реляционная и нереляционная базы данных, а также описываются спроектированные классы. На данном этапе выбираются инструменты разработки сервиса для хранения корпусов.

## 2.1. Проектирование работы сервиса

При проектировании работы сервиса были созданы диаграммы активностей для каждой функции контракта сервиса.

Добавление информации о пользователе (регистрация нового пользователя) представлено на рисунке 2.1.

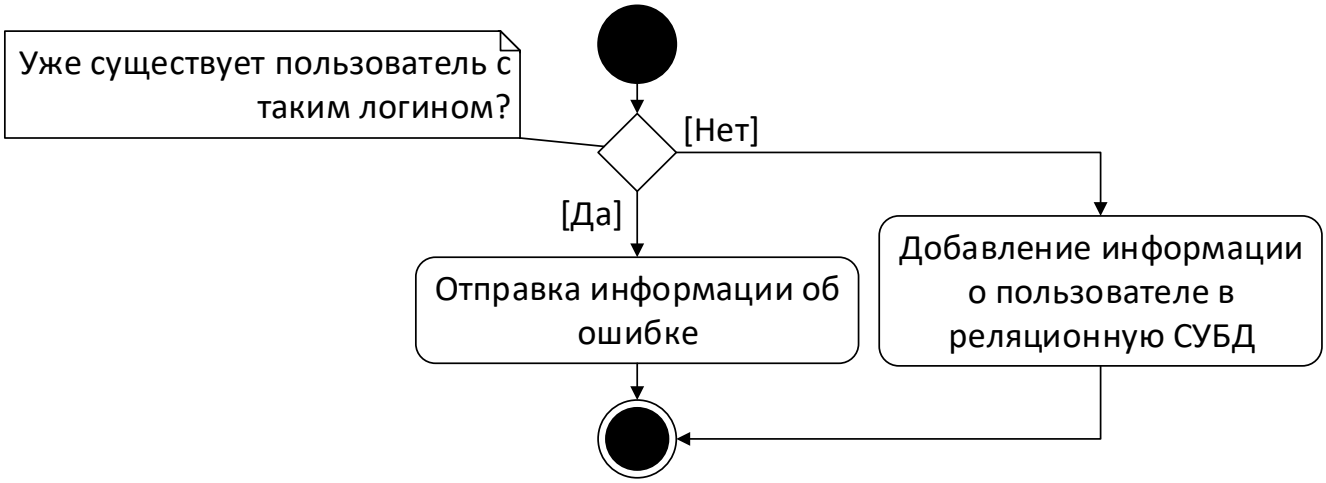


***Рисунок 2.1. Добавление нового пользователя***

Изменение информации о пользователе представлено на рисунке 2.2.

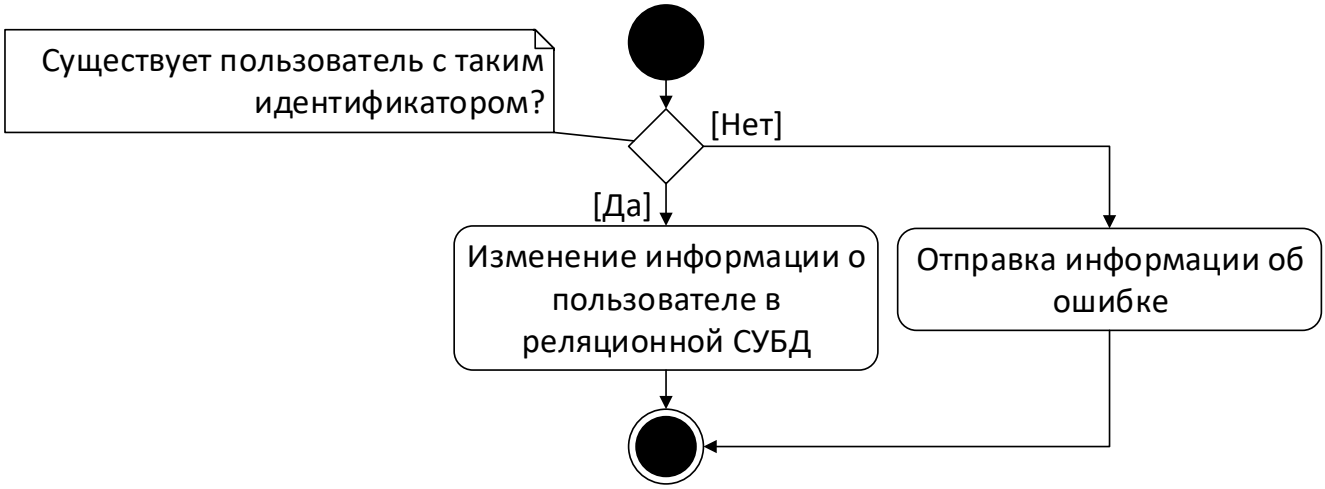


***Рисунок 2.2. Изменение информации о пользователе***

Удаление информации о пользователе представлено на рисунке 2.3.

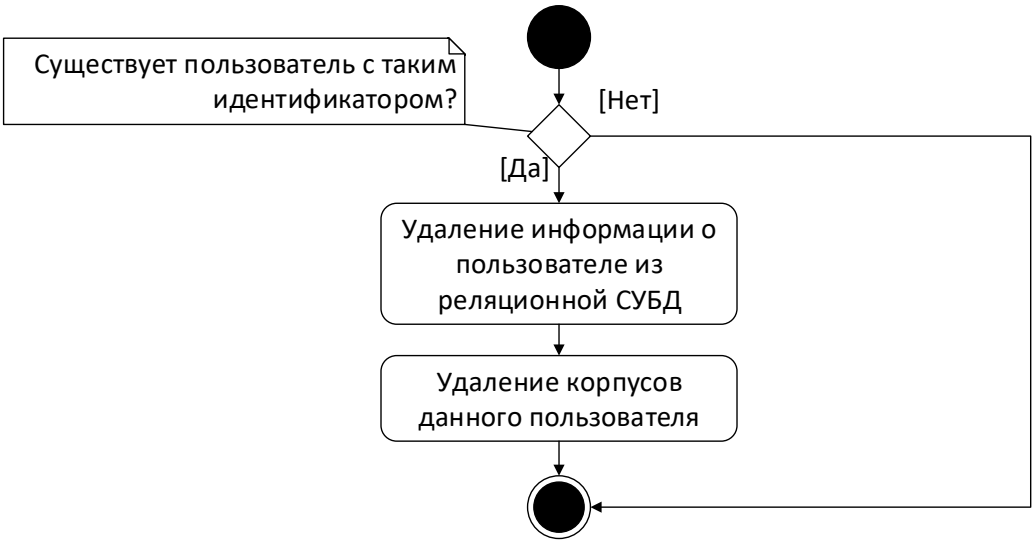


***Рисунок 2.3. Удаление информации о пользователе***

Получение информации о пользователе представлено на рисунке 2.4.

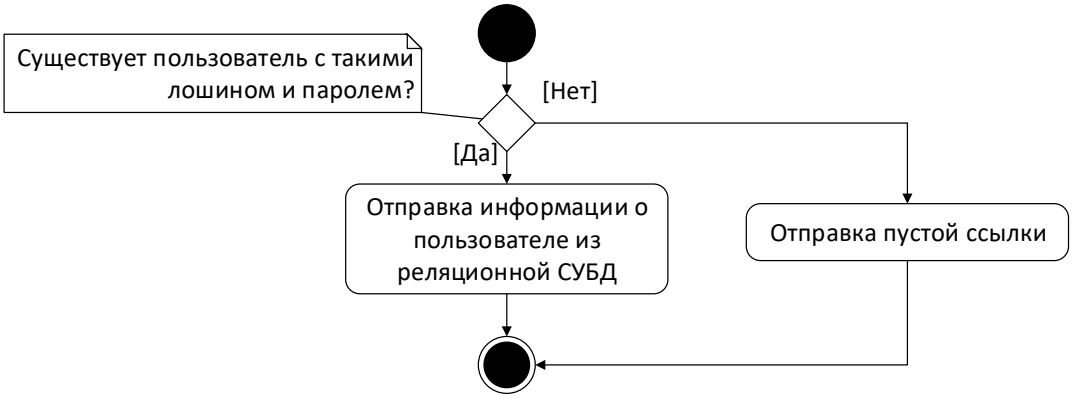


***Рисунок 2.4. Получение информации о пользователе***

Добавление информации о корпусе (создание нового) представлено на рисунке 2.5.

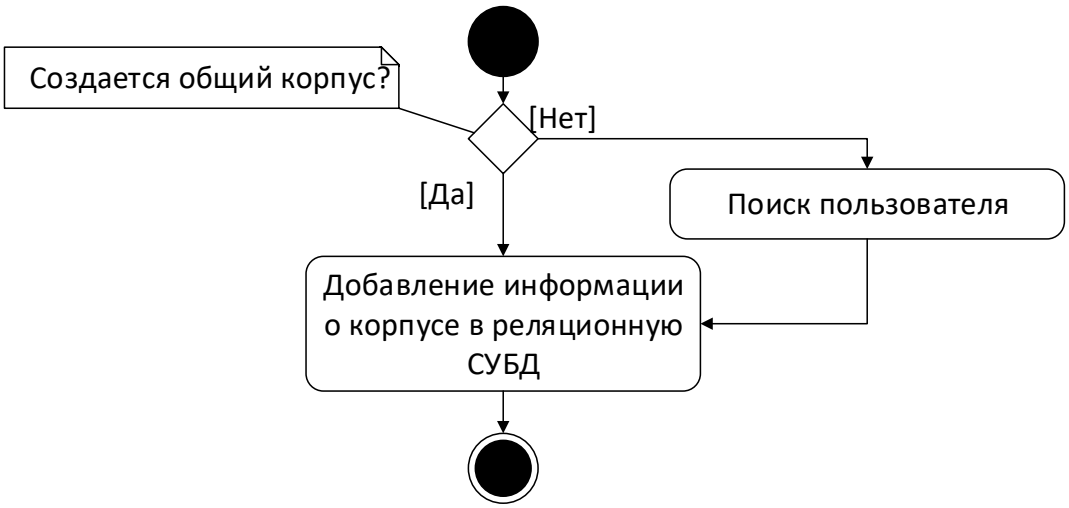


***Рисунок 2.5. Добавление информации о корпусе***

Изменение информации о корпусе представлено на рисунке 2.6.

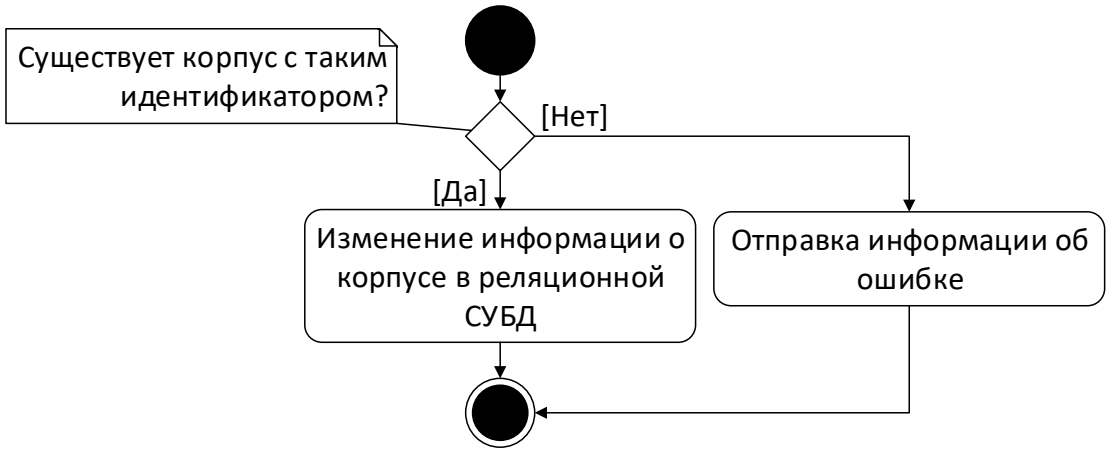


***Рисунок 2.6. Изменение информации о корпусе***

Удаление информации о корпусе представлено на рисунке 2.7.

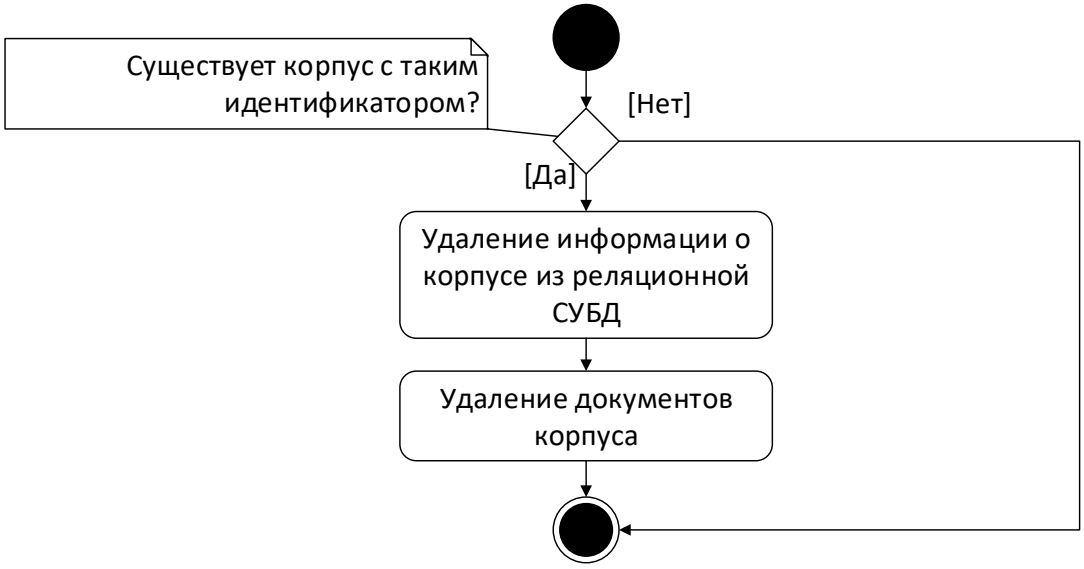


***Рисунок 2.7. Удаление информации о корпусе***

Получение информации о корпусе представлено на рисунке 2.8.

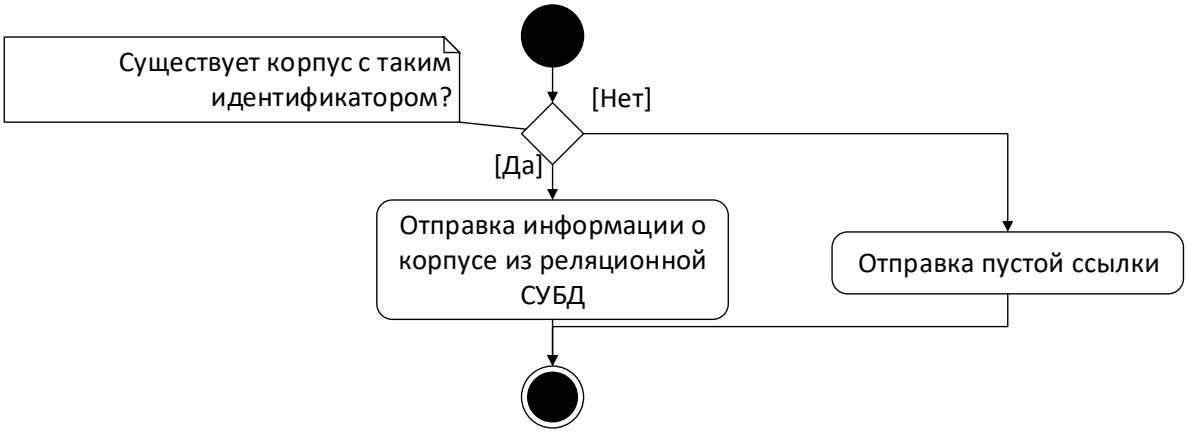


***Рисунок 2.8. Получение информации о корпусе***

Удаление документа представлено на рисунке 2.9.

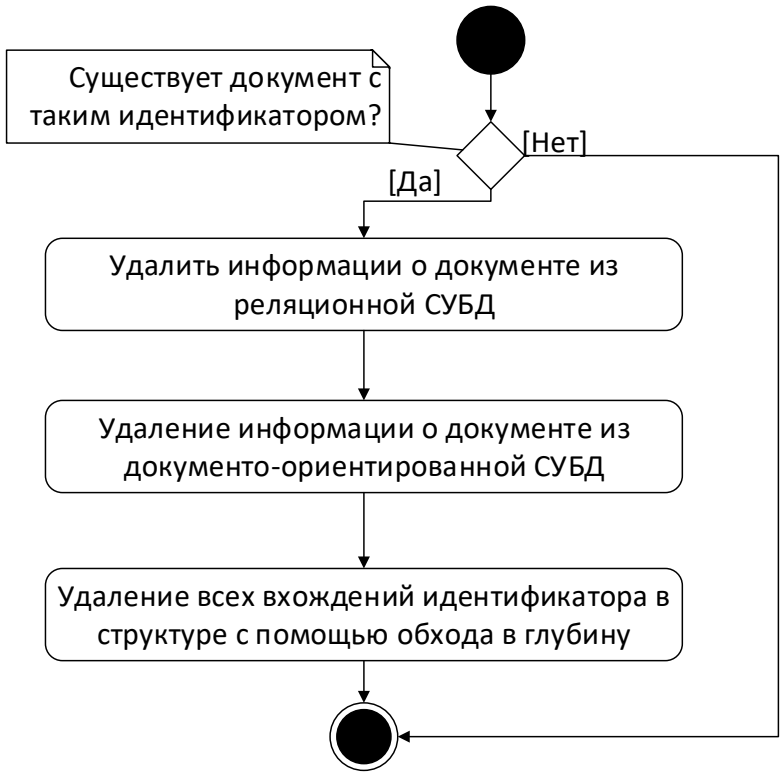


***Рисунок 2.9. Удаление информации о документе***

Изменение документа представлено на рисунке 2.10.

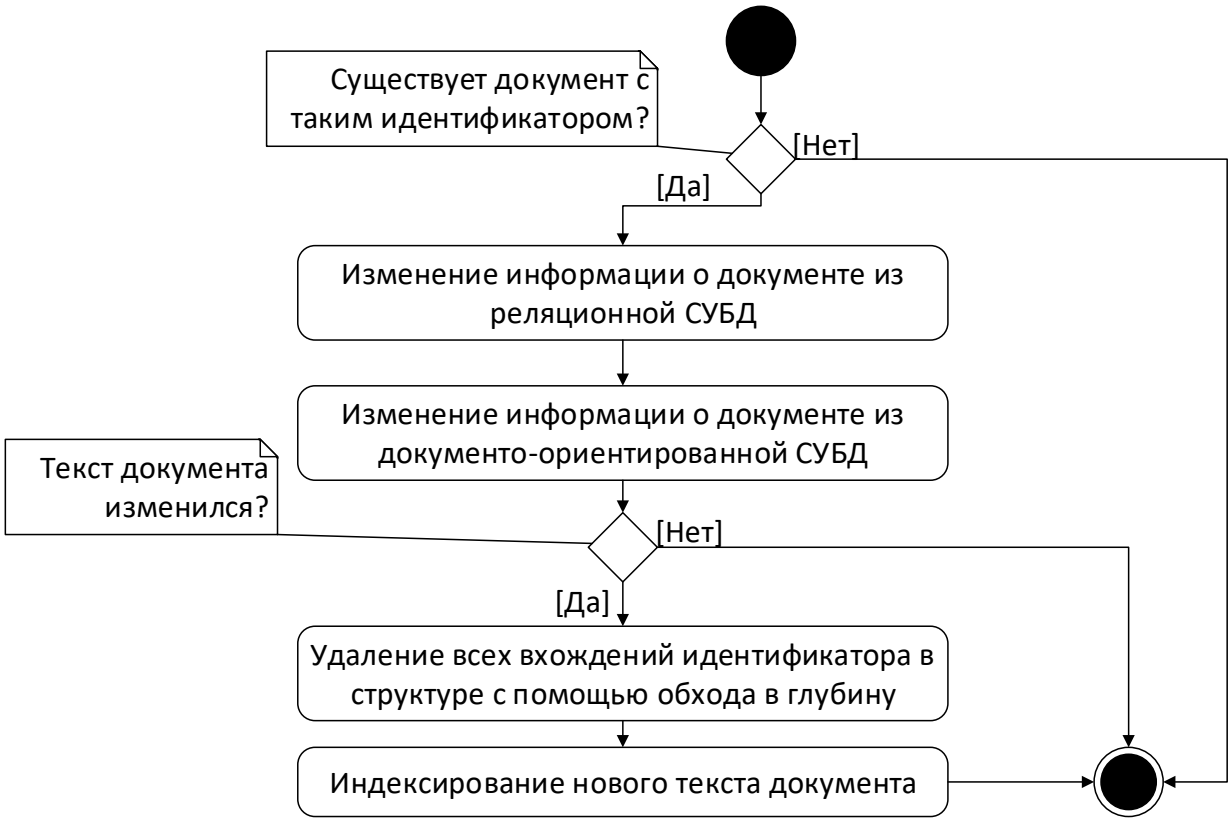


***Рисунок 2.10. Изменение информации о документе***

Получение документа представлено на рисунке 2.11.

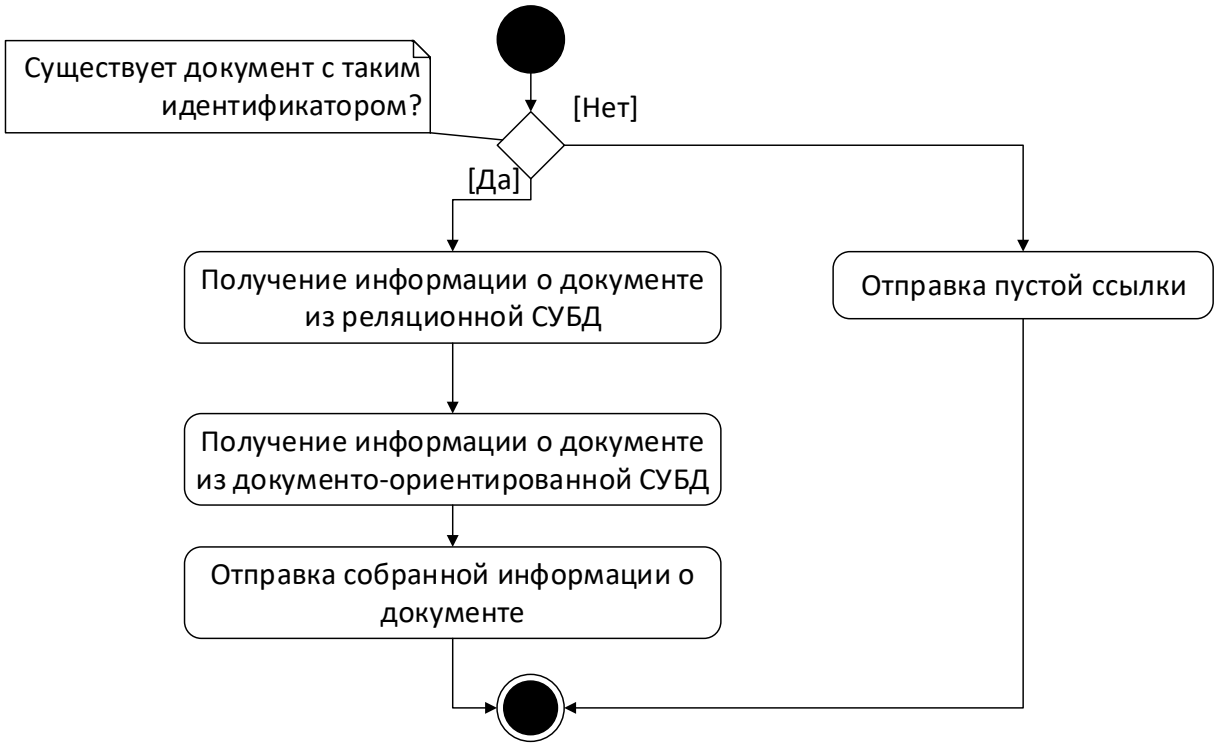


***Рисунок 2.11. Получение информации о документе***

Поиск документа по ключевым словам представлен на рисунке 2.12.

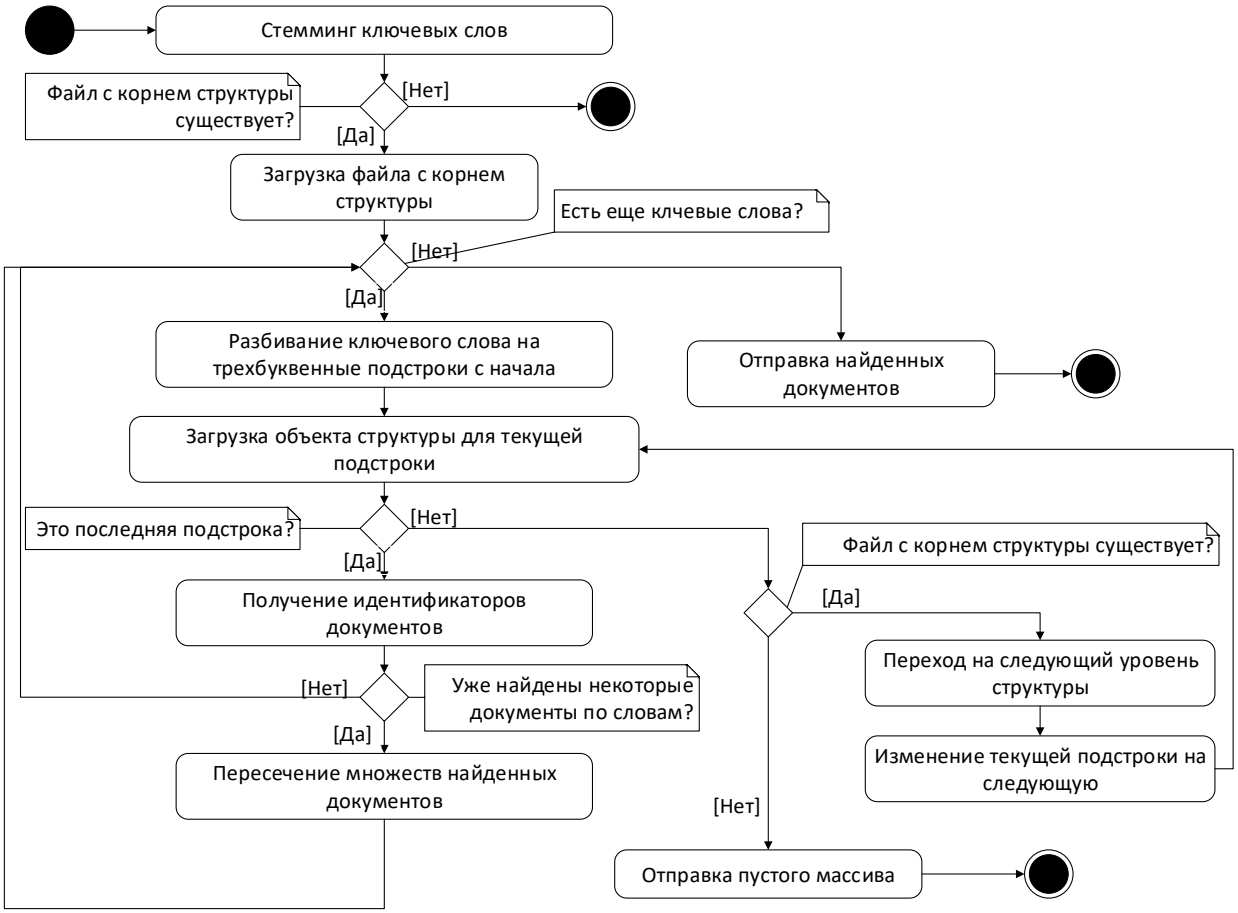


***Рисунок 2.12. Поиск документа по ключевым словам***

Поиск документов по названию представлен на рисунке 2.13.

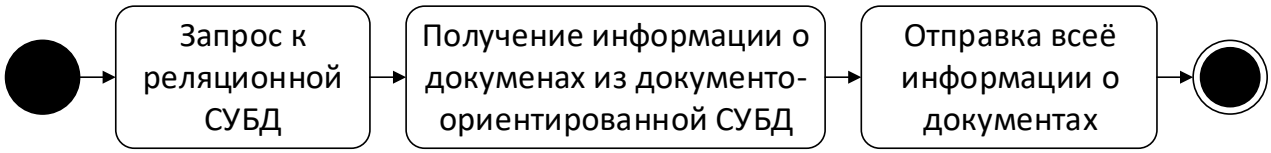


***Рисунок 2.13. Поиск документов по названию***

Для функции добавления документа было выполнено проектирование работы сервиса в виде диаграммы активностей (рис. 2.14).

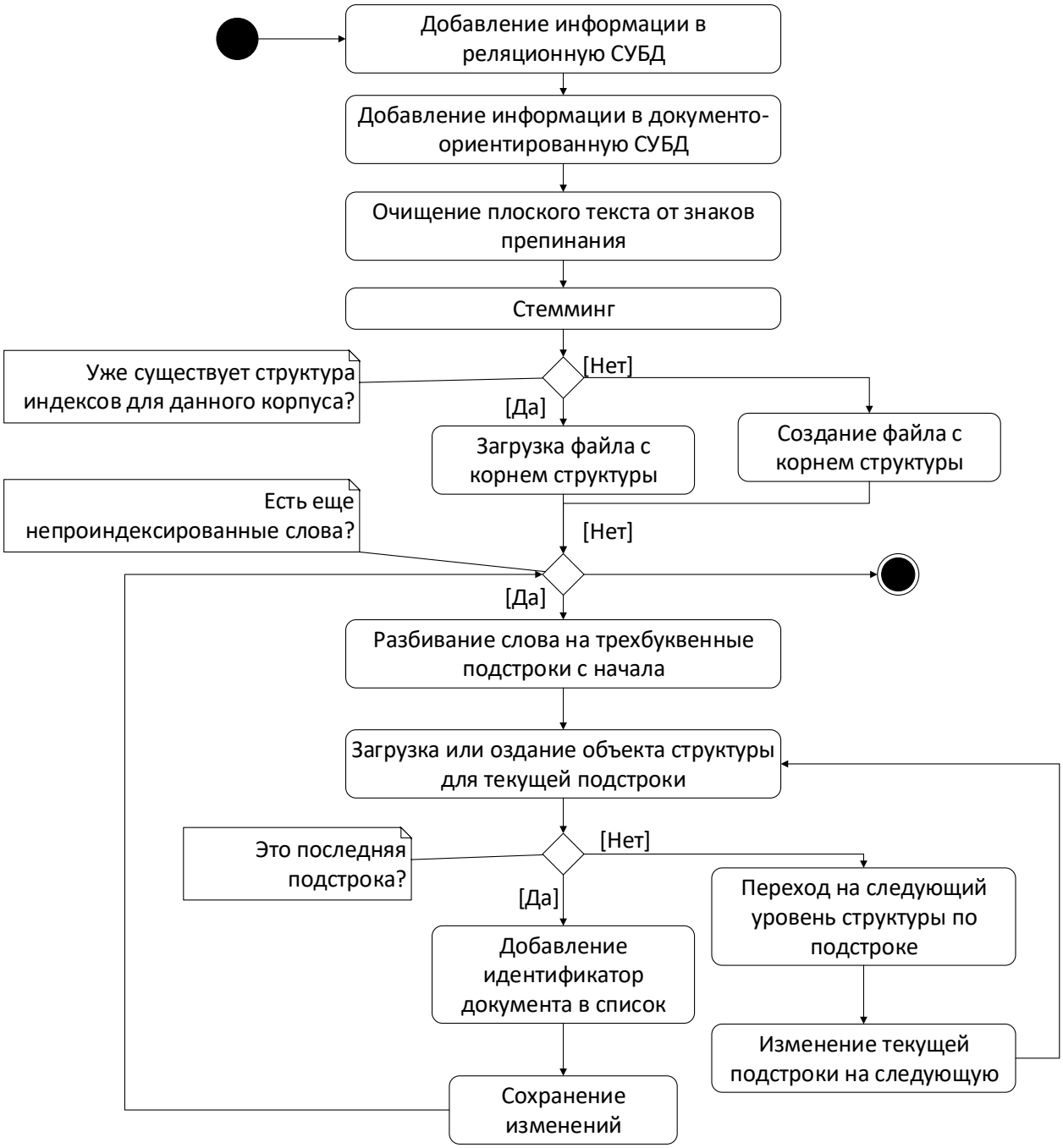


***Рисунок 2.14. Добавление документа***

## 2.2. Проектирование баз данных

На основе поведенного анализа были выделены сущности предметной области. Результаты представлены в виде ERD на рисунке 2.15.

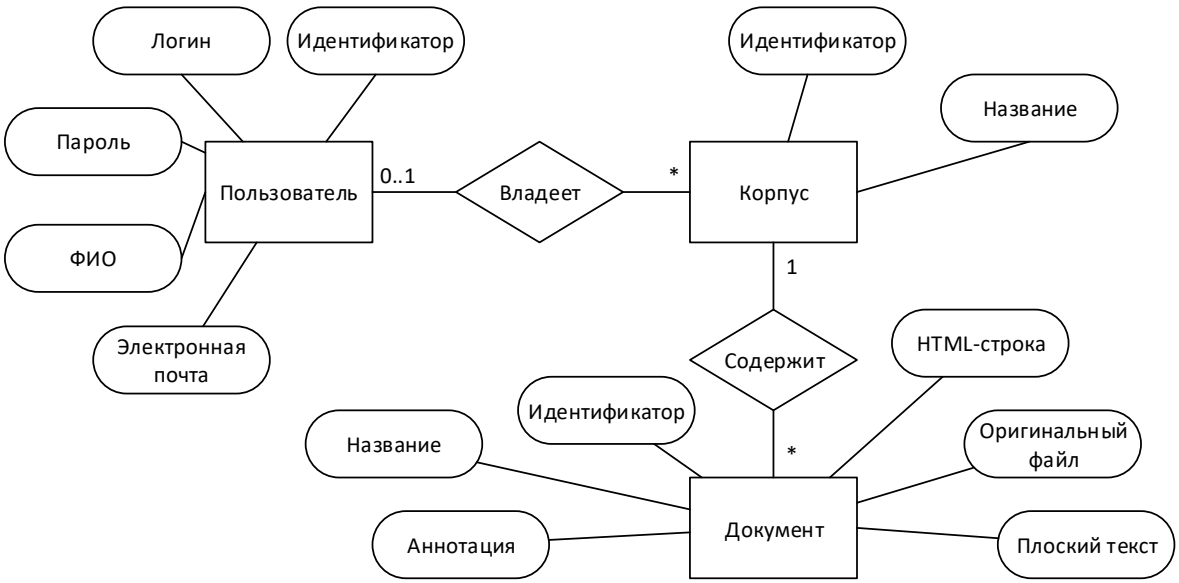


***Рисунок 2.15. Диаграмма сущность-связь***

Были выделены следующие сущности и атрибуты:

1. Пользователь с информацией о ФИО, электронной почтой, логине и пароле пользователя.
2. Корпус с информацией о названии корпуса.
3. Документ, содержащий оригинальный файл, плоский текст, аннотацию, строку HTML-разметки и название.

Информация о пользователях и корпусах содержится в реляционной СУБД (рис. 2.16), а также ключи нереляционной части хранящихся документов в корпусах (DocumentId), названия документов и оригинальный файл.

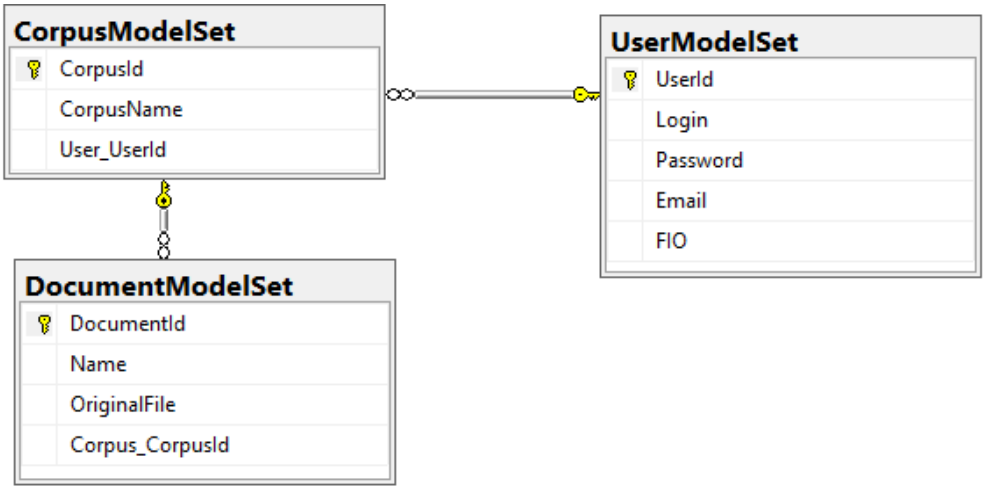


***Рисунок 2.16. Схема реляционной базы данных***

Работа с реляционной базой данной осуществляется с помощью Entity Framework, ниже представлена получившаяся модель (рис. 2.17).

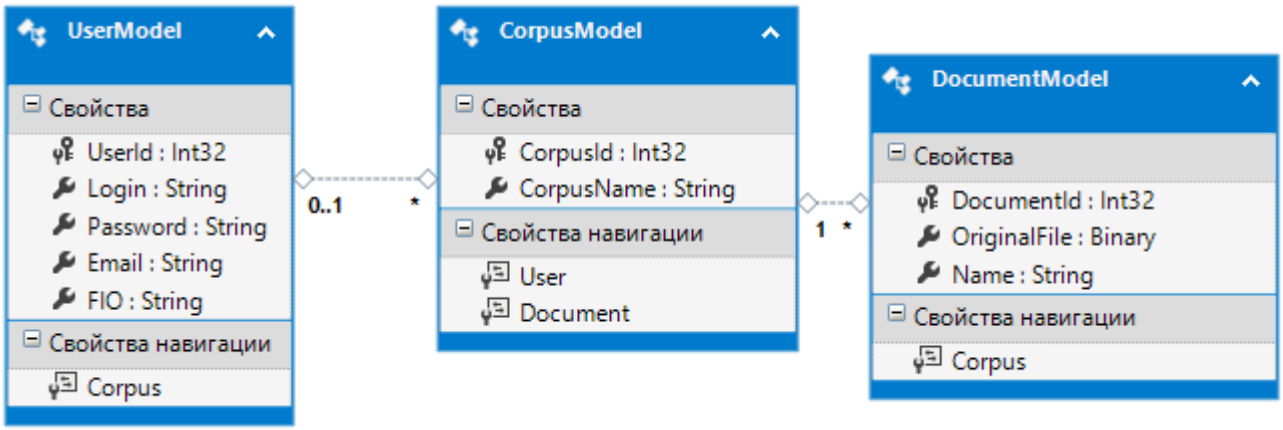


***Рисунок 2.17. Entity Designer Diagram***

В документо-ориентированной базе данных хранится информация о документах. Так как документы в документо-ориентированных СУБД не имеют четкой структуры, была определена примерная схема информации, которая будет храниться в формате JSON (рис. 2.18).

| Документ | |
|---|---|
| DocumentId<br>URI | |
| Annotation | 0..1 |
| FlatText | 0..1 |
| HXML-file | 0..1 |

***Рисунок 2.18. Примерная структура документа в документо-ориентированной базе данных***

Гибкость структуры позволит хранить любую дополнительную информацию о документе помимо определенной выше.

Древовидная структура для хранения индексов на основе естественно-языковой адресации также может храниться в документо-орентированной базе данных в виде файлов с серилизованными объектами.

## 2.3. Диаграмма классов

Следующим этапом проектирования стало создание диаграммы классов (рис. 2.19). Диаграмма классов контракта сервиса включает в себя классы для работы с выделенными сущностями: User, Document, Corpus, а также интерфейс сервиса.

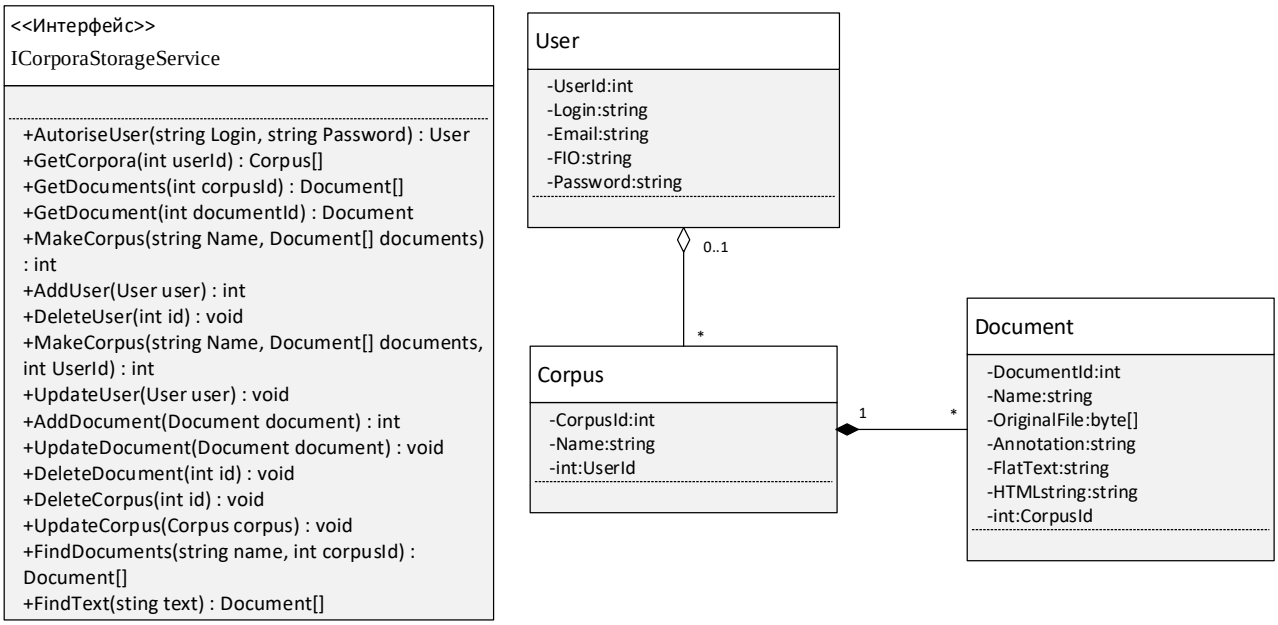


***Рисунок 2.19. Диаграмма классов***

ICorporaStorageServise – интерфейс-контракт сервиса (CorporaStorageServise), описывающий доступные другим сервисам операции:

1. AutorizeUser(string Login, string Password) : User – возвращает информацию о пользователе по логину и паролю.
2. GetCorpora(int UserId) : Corpus[] – возвращает список всех корпусов пользователя.
3. GetDocuments(int CorpusId) : Document[] – возвращает список всех документов в корпусе.
4. GetDocument(int Document) : Document – возвращает информацию о документе.
5. MakeCorpus(string Name, Document[] documents) : int – создает общий корпус с выбранными документами (можно создать и пустой корпус, передав пустой массив).
6. MakeCorpus(string Name, Document[] documents, int UserId) : int – создает частный корпус с выбранными документами (можно создать и пустой корпус, передав пустой массив).
7. AddUser(User user) : int – регистрация нового пользователя.
8. UpdateUser(User user) : void – изменение информации о пользователе. По старому идентификатору обновляются все поля.
9. DeleteUser(int userId) : void – удаление информации о пользователе по идентификатору. При этом происходит удаление всех корпусов пользователя.

10. AddDocument(Document document) : int – добавление нового документа.
11. UpdateDocument(Document document) : void – изменение документа. По старому идентификатору обновляются все поля.
12. DeleteDocument(int documentId) : void – удаление документа по идентификатору.
13. UpdateCorpus(Corpus corpus) : void – изменение корпуса. По старому идентификатору обновляются все поля.
14. DeleteCorpus(int corpusId) : void – удаление корпуса по идентификатору. При этом происходит удаление всех документов корпуса.
15. FindDocuments(string Name, int corpusId) : Document[] – поиск документа по имени.
16. FindText(string Text) : Document[] – поиск фрагмента по документам.

Диаграмма классов (рис. 2.20.) сервиса включает в себя вспомогательные классы для работы с базами данных: UserModel, DocumentModel, CorpusModel – классы для работы с реляционной базой данной с помощью Entity Framework; AzureDocument – класс для работы с документо-ориентированной базой данных.

Класс реализации сервиса помимо реализации контракта имеет различные вспомогательные методы.

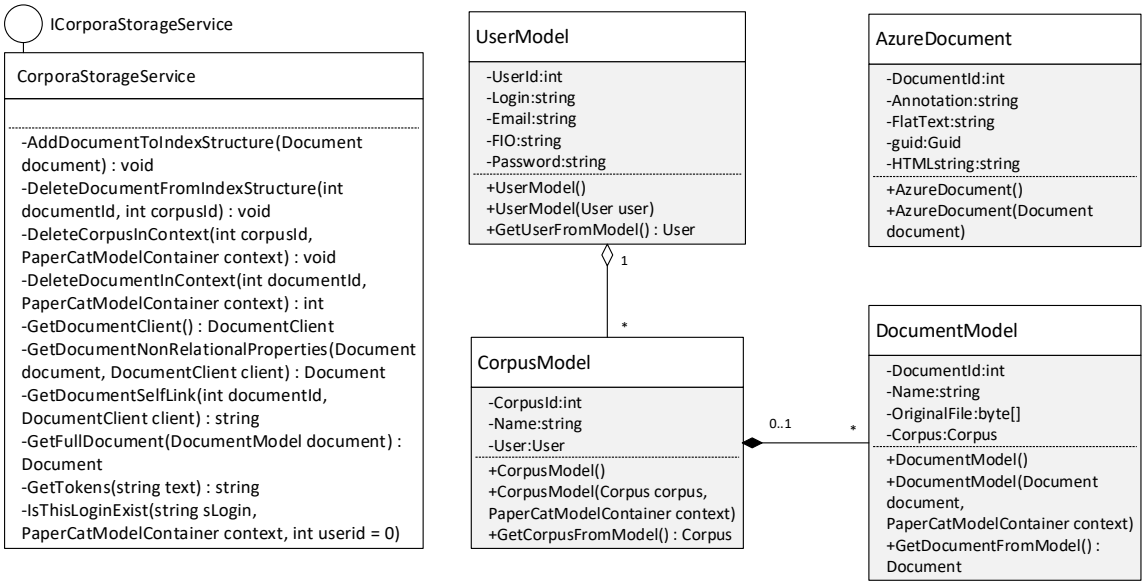


***Рисунок 2.20. Диаграмма классов реализации сервиса***

Классы для работы с индексацией (рис. 2.21) на основе естественно-языковой адресации включают в себя классы структуры индексов: NLAStructure и

NLAHeadDocument, которые включают в себя словарь ссылок на класс IndexStructureObject.

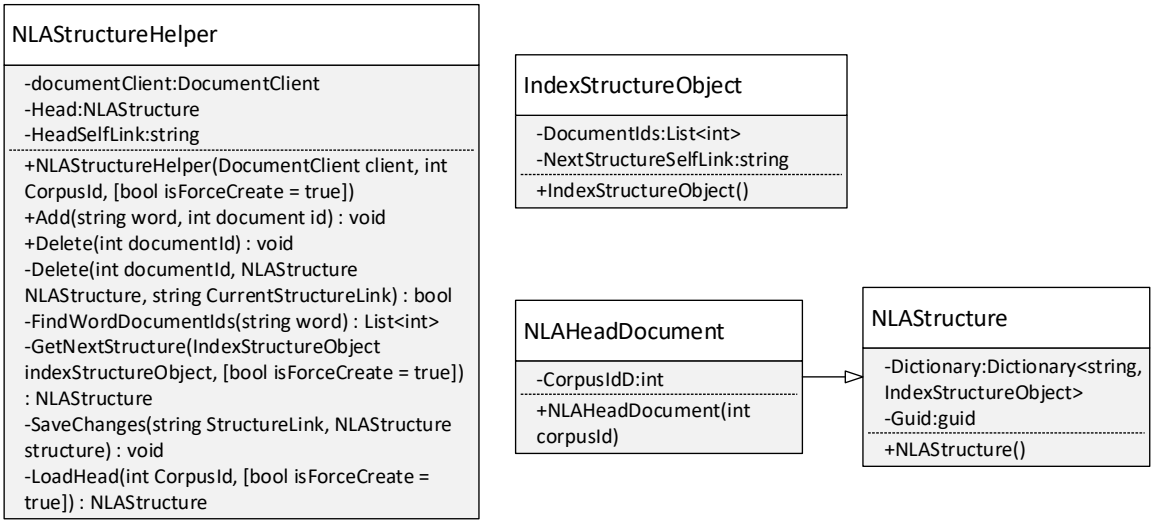


***Рисунок 2.21. Диаграмма классов для работы с индексацией***

Словарь (Dictionary) был выбран, так как его можно сериализовать в формате JSON, а также скорость доступа Dictionary не сильно уступает стандартной реализации хэш-таблицы (Hashtable), а для типов-значений скорость даже превосходит, так как в основе обоих структур лежит хеш-таблица [8, 16].

IndexStructureObject содержит следующий уровень структуры и/или ссылки на идентификаторы документов, в которых содержится соответствующее ключевое слово. Класс NLAStructureHelper содержит методы для добавления идентификатора документа к соответствующему ключевому слову, а также удаления идентификатора документа из всей структуры. Также в классе NLAStructureHelper содержаться вспомогательные методы.

## 2.4. Архитектура системы

При создании системы в отдельные функциональные блоки были выделены:

1. Компонент для стемминга текста.
2. Контракты сервиса.
3. Реляционная база данных
4. Нереляционная база данных.
5. Сам сервис.

Для описания архитектуры системы была создана диаграмма компонентов, представленная на рисунке 2.22.

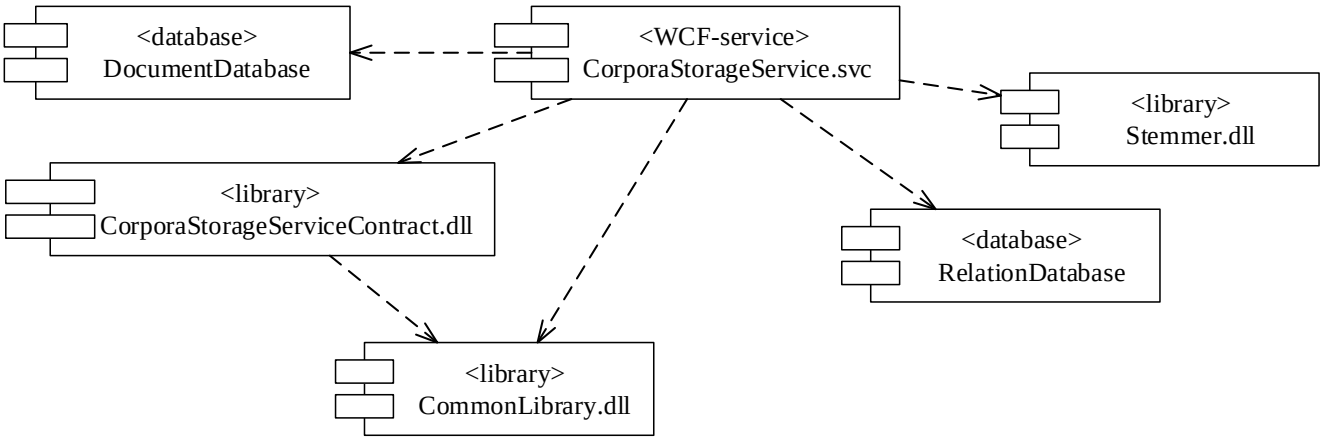


***Рисунок 2.22. Диаграмма компонентов сервиса-хранилища***

CorporaStorageServise.svc – компонент реализации сервиса, который использует библиотеку с контрактом сервиса (CorporaStorageServiseContract.dll), то есть с описанием его интерфейса, а также общую библиотеку (CommonLibrary.dll), в которую войдут все необходимые для реализации стандартов взаимодействия классы.

Для хранения реляционных данных будет использоваться также располагаемая на сервисах Azure база данных SQL Azure (RelationDatabase), предоставляемая как сервис. DocumentDatabase – Cosmos DB, нереляционная база данных как сервис для хранения документов. Также используется сторонняя реализация стеммера по алгоритму стемминга Портера (Stemmer.dll).

## 2.5. Технико-экономическое обоснование

Стоимость разработки программы составит 40 486 рублей, а её расчет и обоснование выбранных инструментальных средств приведены в приложении С. Расчет стоимости проводился по методологии Work Breakdown Structure (WBS) или иерархическая структура работ. Данная методология была выбрана исходя из текущей стадии жизненного цикла программы, а также она не требует накопленной статистики. Данная методология обладает преимуществами для небольших проектов с небольшим количеством участников, а именно:

1. Не требует сильных затрат.
2. Позволяет увеличить производительность и контроль [25].

## 2.6. Итоги проектирования

На этапе проектирования были создана архитектура сервиса для хранения корпусов текстов, которая позволяет удовлетворить выдвинутые ранее требования к

сервису. Во-первых, были выделены классы контракта сервиса, позволяющие взаимодействовать с ним. Во-вторых, были выделены классы для взаимодействия с базами данных, а также для реализации индексации на основе естественно-языковой адресации.

Выделенные методы контракта сервиса покрывают все функциональные требования. Также выделение контрактов в отдельные библиотеки позволяет использовать класс ChannelFactory при работе с сервисом.

# Глава 3. Реализация сервиса для хранения корпусов

Сервис для хранения корпусов был реализован как WCF-сервис на языке C#, при этом в качестве интегрированной среды разработки программы была использована среда Visual Studio 2017.

## 3.1. Реализация CRUD-функций

При реализации CRUD функций были использованы API используемых баз данных (Cosmos DB и SQL Azure), а также технология Entity Framework и LINQ для работы с реляционной базой данных. Для увеличения производительности Entity Framework время жизни контекста ограничивался методом сервиса.

CRUD-функций для пользователя и корпуса, а также реляционная часть документа реализуются с помощью LINQ-запросов. Например, добавление пользователя происходит следующим образом (рис. 3.1):

```
context.UserModelSet.Add(userModel);
context.SaveChanges();
```

***Рисунок 3.1. Добавление пользователя***

При добавлении или изменении пользователя происходит проверка другого наличия пользователя с таким логином (рис. 3.2):

```
context.UserModelSet.Where(u => u.Login == sLogin
   && u.UserId != userid).FirstOrDefault() != null;
```

***Рисунок 3.2. Проверка наличия другого пользователя с таким логином***

При этом, при удалении корпуса происходит удаление связанных с ним документов, а при удалении пользователя удаляются все личные корпусы.

Для нереляционной части документа используется предоставляемая Microsoft библиотека клиента Azure Cosmos DB, которая позволяет добавлять, удалять, изменять документы из коллекции, а также выполнять LINQ-запросы к документам. Например, добавление документа представлено на рисунке 3.3.

```
client.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri
    (Properties.Resources.DataBaseId, Properties.Resources.DocumentsCollectionId),
    new AzureDocument(document)).Wait();
```

***Рисунок 3.3. Добавление документа в документо-ориентированную базу данных***

Также при добавлении, удалении или изменении текста документа происходит соответствующее изменение данных в структуре для индексации на основе естественно-языковой адресации.

## 3.2. Реализация индексации на основе естественно-языковой адресации

Была реализована многомерная древовидная структура в виде многоуровневой хеш-таблицы на основе стандартного класса Dictionary<Tkey, TValue>. Ключом была выбрана подстрока из трех символов. Так как обработка текста и структуры индексов занимает достаточно большое время, данные операции проводятся в отдельных потоках, завершение которых необязательно для завершения метода.

При добавлении нового документа его плоский текст обрабатывается следующим образом для добавления ключевых слов в структуру для индексации на основе естественно-языковой адресации:

1. Очищение текста от знаков препинания и стоп-слов.
2. Стемминг – выделение основ ключевых слов при помощи дополнительной библиотеки.
3. Добавление основы каждого ключевого слова в структуру индексации.

При добавлении в структуру производятся следующие действия:

1. Загружается из документо-ориентированной базы данных файл с соответствующим корнем дерева и десериализуется.
2. Для каждой трехбуквенной (последняя может быть меньше) подстроки с начала слова:
   2.1. Получается объект IndexStructureObject по данной подстроке-ключу в текущей структуре.
   2.2. Если это последняя подстрока, то идентификатор документа добавляется к списку идентификаторов.
   2.3. Иначе загружается или создается структура следующего уровня для данной подстроки, она становится текущей.
   2.4. Изменения сохраняются.

При удалении идентификатора документа из структуры происходит обход дерева в глубину (так как вложенность не слишком высока), и все вхождения данного идентификатора удаляются.

## 3.3. Публикация сервиса

Для работы сервиса на портале Azure были созданы базы данных: реляционная SQL Azure и документо-ориентированная Cosmos DB. Таблицы реляционной СУБД

создавались с помощью Entity Framework. В документо-ориентированной Cosmos DB были созданы две коллекции: для документов и для сериализованной структуры для индексации на основе естественно-языковой адресации.

Далее с помощью предоставляемого Azure профиля публикации веб-приложения и встроенных средств Visual Studio сервис опубликован и доступен по ссылке: https://papercatdatastorage.azurewebsites.net/CorporaStorageService.svc (рис. 3.4).

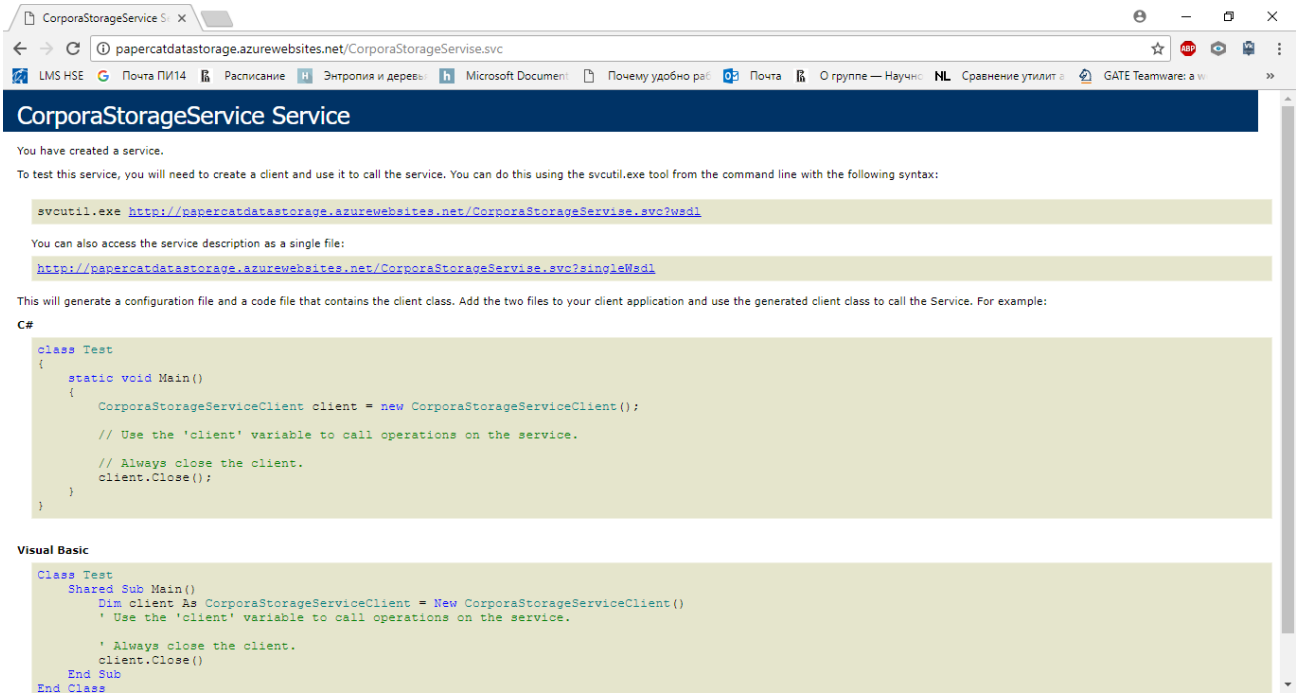


***Рисунок 3.4. Опубликованный сервис***

Таким образом, сервис доступен по сети Интернет и может быть использован разными видами клиентов, все данные при этом хранятся в облачном хранилище портала Azure.

# Глава 4. Тестирование

В ходе работы необходимо протестировать не только реализацию всех функций сервиса для хранения корпусов, но и возможность взаимодействие этого сервиса с другими компонентами системы. Также необходимо исследовать полезность применения индексации на основе естественно-языковой адресации.

## 4.1. Функциональное тестирование

На этапе функционального тестирования проводилось тестирование основных функций сервиса по методу черного ящика:

1. Добавление нового пользователя.

Результаты тестирования добавления информации о пользователе представлены в таблице 4.1.

***Таблица 4.1. Тестирование добавления информации о пользователе***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | null | Ошибка «Параметр не может быть null» | Ошибка «Параметр не может быть null» |
| 2 | User{ FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login”, Password = “Password”} | В базу данных добавлена информация о пользователе с FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login”, Password = “Password”. UserId сгенерирован | В базу данных добавлена информация о пользователе с FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login”, Password = “Password”. UserId сгенерирован (1) |
| 3 | User{UserId = 0, FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login3”, Password = “Password3”} | В базу данных добавлена информация о пользователе с FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login3”, Password = “Password3”. UserId сгенерирован | В базу данных добавлена информация о пользователе с FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login3”, Password = “Password3”. UserId сгенерирован (2) |
| 4 | User{ Email = “email@email.com”, Login = “Login4”, Password = “Password4”} | В базу данных добавлена информация о пользователе с Email = “email@email.com”, Login = “Login”, Password = “Password”. UserId сгенерирован | В базу данных добавлена информация о пользователе с Email = “email@email.com”, Login = “Login”, Password = “Password”. UserId сгенерирован (3) |
| 5 | User{ Email = “email@email.com”, Password = “Password”} | Ошибка валидации Entity Framework | Ошибка валидации Entity Framework |
| 6 | User{UserId = 0, FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login” (Уже существует в БД), Password = “Password”} | Ошибка «Пользователь с таким логином уже существует» | Ошибка «Пользователь с таким логином уже существует» |

2. Удаление информации о пользователе.

Результаты тестирования удаления информации о пользователе представлены в таблице 4.2.

***Таблица 4.2. Тестирование удаления информации о пользователе***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | UserId = 0 (Пользователя с таким id в базе данных нет) | Нет действий | Нет действий |
| 2 | UserId = 1 (Пользователь без корпусов) | Удаление информации о пользователе с UserId = 1 из базы данных | Удаление информации о пользователе UserId = 1 из базы данных |
| 3 | UserId = 2 (Пользователь с одним корпусом) | Удаление информации о пользователе с UserId = 2 из базы данных, удаление связанного корпуса | Удаление информации о пользователе с UserId = 2 из базы данных, удаление связанного корпуса |
| 4 | UserId = 3 (Пользователь с тремя корпусами) | Удаление информации о пользователе с UserId = 3 из базы данных, удаление связанных корпусов | Удаление информации о пользователе с UserId = 3 из базы данных, удаление связанных корпусов |

3. Изменение информации о пользователе.

Результаты тестирования изменения информации о пользователе представлены в таблице 4.3.

***Таблица 4.3. Тестирование изменения информации о пользователе***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | null | Ошибка «Параметр не может быть null» | Ошибка «Параметр не может быть null» |
| 2 | User{ FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login”, Password = “Password”} | Ошибка «Такого пользователя не существует» | Ошибка «Такого пользователя не существует» |
| 3 | User{UserId = 0, FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login3”, Password = “Password3”} | Ошибка «Такого пользователя не существует» | Ошибка «Такого пользователя не существует» |
| 4 | User{ UserId = 3, Email = “email2@email.com”, Login = “Login4”, Password = “Password4”} | В базе данных изменена информация о пользователе с UserId = 3 на Email = “email2@email.com”, Login = “Login”, Password = “Password” | В базе данных изменена информация о пользователе с UserId = 3 на Email = “email2@email.com”, Login = “Login”, Password = “Password” |
| 5 | User{ UserId = 3, Email = “email@email.com”, Password = “Password”} | Ошибка валидации Entity Framework | Ошибка валидации Entity Framework |
| 6 | User{UserId = 3, FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login” (Уже существует в БД), Password = “Password”} | Ошибка «Другой пользователь с таким логином уже существует» | Ошибка «Другой пользователь с таким логином уже существует» |

4. Авторизация пользователя (получение информации о пользователе).

Результаты тестирования авторизации пользователя представлены в таблице 4.4.

***Таблица 4.4. Тестирование авторизации пользователя***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | Login = “Login4”, Password = “Password” | null | null |
| 2 | Login = “Login4”, Password = “Password4” | User { UserId = 4, FIO = null, Email = “email2@email.com”, Login = “Login4”, Password = “Password4”} | User { UserId = 4, FIO = null, Email = “email2@email.com”, Login = “Login4”, Password = “Password4”} |

5. Добавление корпуса.

Результаты тестирования добавления корпуса представлены в таблице 4.5.

***Таблица 4.5. Тестирование добавления корпуса***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | MakeCorpus(Name = “TestName”, Documents = null, UserId = 0) | В базу данных добавлен общий корпус без документов с названием “TestName” | В базу данных добавлен общий корпус без документов с названием “TestName” |
| 2 | MakeCommonCorpus(Name = “Test2Name”, Documents = null) | В базу данных добавлен общий корпус без документов с названием “Test2Name” | В базу данных добавлен общий корпус без документов с названием “Test2Name” |
| 3 | MakeCorpus(Name = “Test3Name”, Documents [ { DocumentId = 0, FlatText = "Another", Name = "Test Document", CorpusId = 0 }], UserId = 2) | В базу данных добавлен частный корпус с документом (DocumentId = 0, FlatText = " Another", Name = "Test Document", CorpusId соответствует) с названием “Test3Name” | В базу данных добавлен частный корпус с документом (DocumentId = 0, FlatText = " Another", Name = "Test Document", CorpusId соответствует) с названием “Test3Name” |
| 4 | MakeCorpus(Name = “TestName”, Documents = [ { DocumentId = 0, FlatText = "Another", Name = "Test Document", CorpusId = 0 }, { DocumentId = 0, FlatText = "Second Another", Name = "Test2 Document", CorpusId = 0 }], UserId = 3) | В базу данных добавлен частный корпус с документами (DocumentId = 0, FlatText = "Another", Name = "Test Document", CorpusId соответствует; DocumentId = 0, FlatText = "Just flat text second", Name = "Test2 Document", CorpusId соответствует) с названием “Test3Name” | В базу данных добавлен частный корпус с документами (DocumentId = 0, FlatText = "Another", Name = "Test Document", CorpusId соответствует; DocumentId = 0, FlatText = "Just flat text second", Name = "Test2 Document", CorpusId соответствует) с названием “Test3Name” |
| 5 | MakeCorpus(Name = “Test5Name”, new Documents[0], UserId = 2) | В базу данных добавлен частный корпус без документов с названием “Test5Name” и UserId=2 | В базу данных добавлен частный корпус без документов с названием “Test5Name” и UserId=2 |

6. Изменение корпуса.

Результаты тестирования изменения корпуса представлены в таблице 4.6.

***Таблица 4.6. Тестирование изменения корпуса***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | null | Ошибка «Параметр не может быть null» | Ошибка «Параметр не может быть null» |
| 2 | Corpus { Name = “”, UserId = 0, CorpusId = 0} | Ошибка «Такого корпуса не существует» | Ошибка «Такого корпуса не существует» |

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 3 | Corpus { Name = “Test”, UserId = 2, CorpusId = 1} | Изменение корпуса в БД с CorpusId = 1 на Name = “Test” | Изменение корпуса в БД с CorpusId = 1 на Name = “Test” |
| 4 | Corpus { Name = “”, CorpusId = 1} | Изменение корпуса в БД с CorpusId = 1 на Name = “ ” | Изменение корпуса в БД с CorpusId = 1 на Name = “ ” |

7. Удаление корпуса.

Результаты тестирования удаления корпуса представлены в таблице 4.7.

***Таблица 4.7. Тестирование удаления корпуса***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | CorpusId = 0 (Корпуса с таким id в базе данных нет) | Нет действий | Нет действий |
| 2 | CorpusId = 1 (Корпус без документов) | Удаление корпуса из базы данных | Удаление корпуса из базы данных |
| 3 | CorpusId = 3 (Корпус с одним документом) | Удаление корпуса из базы данных, удаление связанного документа | Удаление корпуса из базы данных, удаление связанного документа |
| 4 | CorpusId = 4 (Корпус с двумя документами) | Удаление корпуса из базы данных, удаление связанных документов | Удаление корпуса из базы данных, удаление связанных документов |

8. Получение информации о корпусах.

Результаты тестирования получения информации о корпусах пользователя представлены в таблице 4.8.

***Таблица 4.8. Тестирование получения информации о корпусе***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | UserId = 12 (Нет корпусов) | Пустой массив корпусов | Пустой массив корпусов |
| 2 | UserId = 0 (Общественные корпуса) | Массив корпусов из двух пустых общественных | Массив корпусов из двух пустых общественных |
| 3 | UserId = 3 (Пользователь с одним корпусов) | Массив из корпуса с двумя документами | Массив из корпуса с двумя документами |
| 4 | UserId = 2 (Пользователь с двумя корпусами) | Массив из двух корпусов (один пустой, один с одним документом) | Массив из двух корпусов (один пустой, один с одним документом) |

9. Добавление документа.

Результаты тестирования добавления документа представлены в таблице 4.9.

***Таблица 4.9. Тестирование добавления документа***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | null | Ошибка «Параметр не может быть null» | Ошибка «Параметр не может быть null» |
| 2 | Document { FlatText=”Flat text”, Name=”DocumentName”, CorpusId=0 } | Ошибка «Корпуса с таким id не существует» | Ошибка «Корпуса с таким id не существует» |
| 3 | Document {Name=”DocumentName”, CorpusId=1 } | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную |

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 4 | Document { Annotation=”<title>Midnight Rain</title>”, DocumentId=0, FlatText=””, HTMLstring=”<br/>test<br/>”, Name=”Document2Name”, CorpusId=1 } | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную |
| 5 | Document { Annotation=”<title>Midnight Rain</title>”, DocumentId=0, FlatText=”text”, HTMLstring=”<br/>test<br/>”, Name=”Document2Name”, CorpusId=2} | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную. Добавление в документо-ориентированную базу со структурой индексов для слов «flat» и «text» | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную. Добавление в документо-ориентированную базу со структурой индексов для слов «flat» и «text» |
| 6 | Document { Annotation=”<title>Midnight Rain</title>”, DocumentId=3, FlatText=”Flat text”, HTMLstring=”<br/>test<br/>”, Name=”Document2Name”, CorpusId=2} | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную. Добавление в документо-ориентированную базу со структурой индексов для слов «flat» и «text» | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную. Добавление в документо-ориентированную базу со структурой индексов для слов «flat» и «text» |
| 7 | Document { Annotation=”<title>Midnight Rain</title>”, DocumentId=0, FlatText=”Flat? text!!”, HTMLstring=”<br/>test<br/>”, Name=”Document2Name”, CorpusId=2} | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную. Добавление в документо-ориентированную базу со структурой индексов для слов «flat» и «text» | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную. Добавление в документо-ориентированную базу со структурой индексов для слов «flat» и «text» |

10. Изменение документа.

Результаты тестирования изменения документа представлены в таблице 4.10.

***Таблица 4.10. Тестирование изменения документа***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | null | Ошибка «Параметр не может быть null» | Ошибка «Параметр не может быть null» |
| 2 | Document { FlatText=”Flat text”, Name=”DocumentName”, CorpusId=0 }, IsTextChanged = false | Ошибка «Такого документа не существует» | Ошибка «Такого документа не существует» |
| 3 | Document { DocumentId = 1, Name=”DocumentName”, CorpusId=1 }, IsTextChanged = false | Изменение информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных | Изменение информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных |
| 4 | Document { Annotation=”<title>Midnight Rain</title>”, DocumentId=1, FlatText=””, HTMLstring=”<br/>test<br/>”, Name=”Document2Name”, CorpusId=1 }, IsTextChanged = false | Изменение информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных | Изменение информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных |

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 5 | Document { DocumentId=0, Annotation=”<page>Midnight Rain</page >”, DocumentId=2, FlatText=”new text”, HTMLstring=”<br/>test<br/>”, Name=”Document_Name”, CorpusId=4}, IsTextChanged = true | Изменение информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных. Изменение в документо-ориентированной базе индексов: добавление индексов для слова «new» | Изменение информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных. Изменение в документо-ориентированной базе индексов: добавление индексов для слова «new» |
| 6 | Document { Annotation=”<title>Midnight Rain</title>”, DocumentId=2, FlatText=”text”, HTMLstring=”<br/>test2<br/>”, Name=”Document2Name”, CorpusId=2}, IsTextChanged = true | Изменение информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных. Изменение в документо-ориентированной базе индексов: удаление индексов для слова «new» | Изменение информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных. Изменение в документо-ориентированной базе индексов: удаление индексов для слова «new» |
| 7 | Document { Annotation=”<title>Midnight Rain</title>”, DocumentId=3, FlatText=”Flat? Text!!!”, HTMLstring=”<br/>test<br/>”, Name=”Document2Name”, CorpusId=2}, IsTextChanged = false | Изменение информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных. Изменений в документо-ориентированной базе индексов нет | Изменение информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных. Изменений в документо-ориентированной базе индексов нет |

11. Удаление документа.

Результаты тестирования удаления документа представлены в таблице 4.11.

***Таблица 4.11. Тестирование удаления документа***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | DocumentId = 0 (Документа с таким id в базе данных нет) | Нет действий | Нет действий |
| 2 | DocumentId = 1 | Удаление информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных. Удаление вхождений 1 из документо-ориентированной базе индексов | Удаление информации о документе с DocumentId = 1 в реляционной и документо-ориентированной базе данных. Удаление вхождений 1 из документо-ориентированной базе индексов |
| 3 | DocumentId = 3 | Удаление информации о документе с DocumentId = 3 в реляционной и документо-ориентированной базе данных. Удаление вхождений 1 из документо-ориентированной базе индексов | Удаление информации о документе с DocumentId = 3 в реляционной и документо-ориентированной базе данных. Удаление вхождений 1 из документо-ориентированной базе индексов |

12. Получение информации о документе/документах.

Результаты тестирования получения информации о документе/документах представлены в таблице 4.12.

***Таблица 4.12. Тестирование получения информации о документе***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | GetDocument(DocumentId=0) | null | null |

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 2 | GetDocument(DocumentId=2) | Document {Name=”DocumentName”, CorpusId=1 } | Document {Name=”DocumentName”, CorpusId=1 } |
| 3 | GetDocuments(CorpusId=0) | Пустой массив | Пустой массив |
| 4 | GetDocuments(CorpusId=3) | Пустой массив | Пустой массив |
| 5 | GetDocuments(CorpusId =2) | Массив документов | Массив документов |
| 6 | GetDocuments(CorpusId =1) | Массив документов | Массив документов |

13. Поиск документа по названию.

Результаты тестирования поиска документа по названию представлены в таблице 4.13.

***Таблица 4.13. Тестирование поиска документа по названию***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | «Document», CorpusId=1 | Все общедоступные документы с вхождением подстроки «document» в названии | Все общедоступные документы с вхождением подстроки «document» в названии |
| 2 | «DocUMent» , CorpusId=1 | Все общедоступные документы с вхождением подстроки «document» в названии | Все общедоступные документы с вхождением подстроки «document» в названии |
| 3 | «Document» , CorpusId=2 | Все документы корпуса с CorpusId=2 с вхождением подстроки «document» в названии | Все документы корпуса с CorpusId=2 с вхождением подстроки «document» в названии |
| 4 | «Hello» , CorpusId=1 | Пустой массив документов | Пустой массив документов |
| 5 | «Document» , CorpusId=0 | Пустой массив документов | Пустой массив документов |

14. Поиск документа по ключевым словам.

Результаты тестирования поиска документа по ключевым словам представлены в таблице 4.14.

***Таблица 4.14. Тестирование поиска документа по ключевым словам***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | «Text», CorpusId=1 | Все общедоступные документы с вхождением подстроки «text» | Все общедоступные документы с вхождением подстроки «text» |
| 2 | «TeXt» , CorpusId=1 | Все общедоступные документы с вхождением подстроки «text» | Все общедоступные документы с вхождением подстроки «text» |
| 3 | «TeXts» , CorpusId=1 | Все общедоступные документы с вхождением подстроки «text» | Все общедоступные документы с вхождением подстроки «text» |
| 4 | «TeXts» , CorpusId=2 | Все документы корпуса с CorpusId=2 с вхождением подстроки «text» | Все документы корпуса с CorpusId=2 с вхождением подстроки «text» |
| 5 | «Hello» , CorpusId=1 | Пустой массив документов | Пустой массив документов |
| 6 | «TeXts» , CorpusId=0 | Пустой массив документов | Пустой массив документов |

## 4.2. Интеграционное тестирование

Интеграционное тестирование включало в себя тестирование возможности взаимодействия с сервером с помощью тестовых программ, на которых были протестированы основные функции сервиса (табл. 4.15).

***Таблица 4.15. Интеграционное тестирование***

| № | Входные данные | Ожидаемый результат | Реальный результат |
|---|---|---|---|
| 1 | null | Ошибка «Параметр не может быть null» | Ошибка «Параметр не может быть null» |
| 2 | Добавление информации о пользователе: User{UserId = 0, FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login3”, Password = “Password3”} | В базу данных добавлена информация о пользователе с FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login3”, Password = “Password3”. UserId сгенерирован | В базу данных добавлена информация о пользователе с FIO = “Фамилия Имя Отчество”, Email = “email@email.com”, Login = “Login3”, Password = “Password3”. UserId сгенерирован (5) |
| 3 | Добавление корпуса: (Name = “Test3Name”, Documents [ { DocumentId = 0, FlatText = "Another", Name = "Test Document", CorpusId = 0 }], UserId = 5) | В базу данных добавлен частный корпус с документом (DocumentId = 0, FlatText = " Another", Name = "Test Document", CorpusId соответствует) с названием “Test3Name” | В базу данных добавлен частный корпус с документом (DocumentId = 0, FlatText = " Another", Name = "Test Document", CorpusId соответствует) с названием “Test3Name” |
| 4 | Добавление документа: Document { Annotation=”<title>Midnight Rain</title>”, DocumentId=0, FlatText=”text”, HTMLstring=”<br/>test<br/>”, Name=”Document2Name”, CorpusId=5} | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную. Добавление в документо-ориентированную базу со структурой индексов для слов «flat» и «text» | Добавление информации о документе в реляционную базу данных, а также в документо-ориентированную. Добавление в документо-ориентированную базу со структурой индексов для слов «flat» и «text» |
| 5 | Поиск по ключевому слову: «TeXts» , CorpusId=5 | Document { Annotation=”<title>Midnight Rain</title>”, DocumentId=0, FlatText=”text”, HTMLstring=”<br/>test<br/>”, Name=”Document2Name”, CorpusId=5} | Document { Annotation=”<title>Midnight Rain</title>”, DocumentId=0, FlatText=”text”, HTMLstring=”<br/>test<br/>”, Name=”Document2Name”, CorpusId=5} |
| 6 | Удаление пользователя: UserId = 5 | Удаление информации о пользователе из базы данных, удаление связанного корпуса и документов | Удаление информации о пользователе из базы данных, удаление связанного корпуса и документов |

Были созданы и использованы следующие тестовые программы:

1. Консольное приложение, написанное на языке c#.
2. Консольное приложение, написанное на языке Java, которое использует стандартный модуль для создания кода клиента сервиса из документа WSDL.
3. Web-клиент, представляющий собой HTML-страницу, с вызовами функций сервиса с помощью JQuery.

## 4.3. Тестирование индексации на основе естественно-языковой адресации

Тестирование эффективности индексации на основе естественно-языковой адресации включало в себя сравнение скорости поиска документов по ключевому слову

с применением индексации на основе естественно-языковой адресации и при простом последовательном сканировании документов.

Результаты тестирования, приведенные на рисунке 4.1, показали, что поиск с применением индексации имеет преимущество перед последовательным поиском ключевого слова.

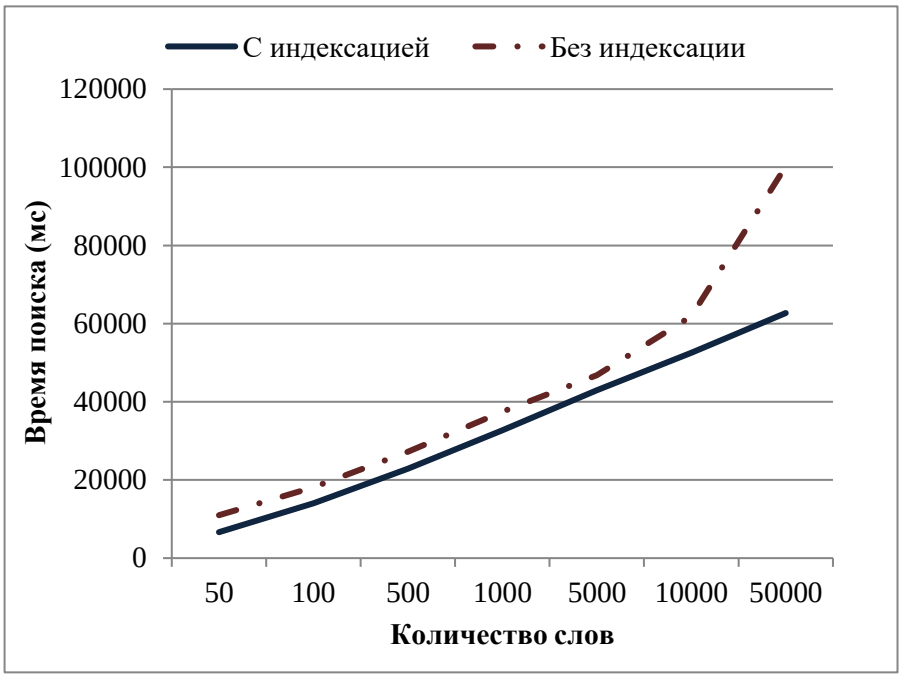


***Рисунок 4.1. Графики зависимости времени поиска от количества слов в корпусе***

При достижении суммарного количества слов в корпусе примерно в 5 тысяч слов разница во времени работы между поиском с индексацией и без индексации всё больше увеличивается.

Таким образом, применение индексации на основе естественно-языковой адресации является целесообразным подходом, даже если суммарное количество слов корпуса не слишком велико.

# Заключение

Итогом данной работы стал WCF-сервис для хранения корпусов текстов с индексацией на основе естественно-языковой адресации, реализованный на языке C#. Данный сервис является частью системы для анализа научных текстов с веб-интерфейсом, которая разрабатывается в рамках работы научно-исследовательской группы НИУ ВШЭ-Пермь.

Была выполнены задачи данной работы: изучены существующие решения, и на основе этого выдвинуты требования к сервису для хранения корпусов текстов, изучены подходы к хранению неструктурированной информации, а именно документо-ориентированные базы данных, среди которых для использования была выбрана Azure Cosmos DB. Выполнение всех задач работы обеспечило достижение поставленной цели.

Индексация на основе естественно-языковой адресации была применена для полнотекстового поиска текстов по ключевым словам в корпусах, что позволила повысить производительность базы данных. Далее был спроектирован контракт сервиса, построены диаграммы прецедентов, классов и компонентов. Разработанный сервис был тщательно протестирован и отлажен.

Созданное приложение позволяет хранить информацию о пользователях, корпусах, а также различную информацию о текстах: название, плоский текст, аннотацию и любые другие благодаря гибкой структуре, документо-ориентированной СУБД. Поиск по ключевым словам текста был реализован с помощью древовидной структуры для индексации на основе естественно-языковой адресации. Данная индексация показала преимущество перед полнотекстовым поиском с полным последовательным чтением текста.

Результаты исследования вместе с результатами работ Каликовой А.Р. и Гуляевым В.Ю. прошли апробацию на всероссийской научно-практической конференции молодых учёных с международным участием «Математика и междисциплинарные исследования – 2018» (ПГНИУ, г. Пермь) в секции «Прикладная лингвистика» и заняли первое место.

Возможной дальнейшей работой является изучение эффективности индексации на основе естественно-языковой адресации по сравнению с другими видами индексации, а также полноценное внедрение сервиса в разрабатываемую систему.

# Библиографический список


1. Бармина Е.И. Система для обработки корпусов текстов / Е.И. Бармина, Р.Н. Бушуев, Н.В. Котельникова, В.В. Ланин, О.А. Плотникова // Математика и междисциплинарные исследования – Пермь: Пермский государственный национальный исследовательский университет, 2016. – С. 245-250.
2. Величко В. К вопросу естественно-языковой адресации / В. Величко, К. Иванова, К. Марков // Problems of Computer in Intellectualization. – София, Болгария: ITHEA 2012. – С. 77-83.
3. Документация по базе данных Azure Cosmos // Техническая документация, материалы по API и примеры кода. URL: https://docs.microsoft.com/ru-ru/azure/cosmos-db/ (дата обращения: 12.11.2017)
4. Индексирование в базах данных // Основы компьютерного моделирования - URL: http://bourabai.ru/dbt/dbms/001.htm (дата обращения: 12.11.2017)
5. Конноли Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Конноли, К. Бегг. – М.: ООО "И.Д. Вильямс". – 2017. – 1440 с.
6. Копотев М.В. Введение в корпусную лингвистику: учеб. пособие для студентов филологических и лингвистических специальностей университетов / М.В. Копотев. – Прага: Animedia Company. – 2014. – 230 с.
7. Сведения о производительности (Entity Framework) // Техническая документация, материалы по API и примеры кода. URL: https://docs.microsoft.com/ru-ru/dotnet/framework/data/adonet/ef/performance-considerations (дата обращения: 04.05.2018)
8. Типы коллекций Hashtable и Dictionary // Техническая документация, материалы по API и примеры кода. URL: https://docs.microsoft.com/ru-ru/dotnet/standard/collections/hashtable-and-dictionary-collection-types (дата обращения: 04.05.2018)
9. Фаулер М. NoSQL: новая методология разработки нереляционных баз данных / М. Фаулер, П. Дж. Садаладж – М.: ООО "И.Д. Вильямс". – 2013. – 192 с.

10. Anthony L. AntConc: design and development of a freeware corpus analysis toolkit for the technical writing classroom / L. Anthony // Anthony Professional Communication Conference, 2005 – IPCC, 2005 – С. 729-737.
11. Bontcheva K. GATE Teamware: a web-based, collaborative text annotation framework / K. Bontcheva , H. Cunningham, Ian Roberts, A. Roberts, V. Tablan, N. Aswani, G. Gorrell // Language Resources and Evaluation. – Sheffield, UK: Springer Netherlands, 2013 –№47 – С. 1007-1029.
12. Boldi P. MG4J at TREC 2005 / P. Boldi, S. Vigna // Proceedings of the Fourteenth Text REtrieval Conference (TREC 2005) – NIST, 2005 – Vol. 500 – С. 266-271.
13. CORSIS // An open-source corpus analysis class library - URL: http://corsis.sourceforge.net/index.php/CORSIS (дата обращения: 27.01.2017)
14. Cunningham H. Gate / H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan // Proceedings of the 40th Annual Meeting on Association for Computational Linguistics – ACL 02, 2002.
15. Cunningham H. Getting More Out of Biomedical Documents with GATEs Full Lifecycle Open Source Text Analytics / H. Cunningham, V. Tablan, A. Roberts, K. Bontcheva // PLoS Computational Biology. – 2013 –№9 – С. 1-16.
16. DictionaryBase.cs // Microsoft Reference Source. URL: https://referencesource.microsoft.com/#mscorlib/system/collections/dictionarybase.cs (дата обращения: 04.05.2018)
17. Entity Framework Windows SQL Azure // Microsoft Developer Network. URL: https://msdn.microsoft.com/en-us/library/jj556244(v=vs.113).aspx (дата обращения: 04.05.2018)
18. GATE. General architecture for text engineering // A full-lifecycle open source solution for text processing. URL: https://gate.ac.uk (дата обращения: 12.11.2017)
19. Hardie A. CQPweb — combining power, flexibility and usability in a corpus analysis tool / A. Hardie // International Journal of Corpus Linguistics. – Amsterdam:John Benjamins Publishing Company, 2012 –№17(3) – С. 380-409.
20. Hirani Z. Entity Framework 6 Recipes / Z. Hirani, L. Tenny, N. Gupta, B. Driscoll, R. Vettor. – Apress. – 2013. – 548 с.

21. Ivanova K. Ontoarm – a system for storing ontologies by natural language addressing / K. Ivanova // International Journal "Information Technologies & Knowledge". – Sofia, Bulgaria: ITHEA 2014. –№8 – С. 303-312.
22. Ivanova K. Practical aspects of natural language addressing / K. Ivanova // Computational models for business and engineering domains. – Sofia, Bulgaria: ITHEA® 2012. – С. 172-186.
23. Manning C. Introduction to Information Retrieval / C. Manning, P. Raghavan, H. Schütze. – Cambridge: Cambridge University Press. – 2008. – 680 с.
24. Markov K. Natural Language Addressing / K. Markov, K. Ivanova, K. Vanhoof, V. Velychko, J. Castellanos. – Sofia, Bulgaria: ITHEA. – 2015. – 315 с.
25. Salvendy G. Handbook of Industrial Engineering, Technology and Operations Management: / G. Salvendy. – Hoboken, NJ, USA: John Wiley & Sons, Inc. – 2001. – 2796 с.
26. Stevens M. Subtleties of Scientific Style / M. Stevens. – Thornleigh, N.S.W. : ScienceScape Editing. – 2007. – 104 с.
27. Windows Communication Foundation // Microsoft Developer Network. URL: https://msdn.microsoft.com/ru-ru/library/dd456779.aspx (дата обращения: 12.11.2017)

# Приложение А.Описание прецедентов

**1. Создание нового корпуса текстов**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для создания корпуса с переданными необходимыми данными в качестве аргументов.

Основной поток:

1. Клиент выполняет вызов функции для создания корпуса (E1).
2. Сервис отправляет запрос в реляционную СУБД для сохранения информации о корпусе (E2).

Альтернативные потоки:

E1: Клиент сначала выполняет поиск пользователя для создания частного корпуса.

E2: При возникновении ошибки система отправляет информацию о ней клиенту.

**2. Изменение корпуса текстов**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для изменения корпуса с переданной новой информацией в качестве аргументов.

Основной поток:

1. Клиент выполняет вызов функции для изменения корпуса.
2. Сервис отправляет запрос в реляционную СУБД для изменения информации о корпусе (E1).

Альтернативные потоки:

E1: При возникновении ошибки система отправляет информацию о ней клиенту.

**3. Удаление корпуса текстов**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для удаления корпуса с переданным идентификатором в качестве аргумента.

Основной поток:

1. Клиент выполняет вызов функции для удаления корпуса (S1).
2. Сервис отправляет запрос в реляционную СУБД для удаления информации о корпусе (E1).

Подпотоки:

S1: Система удаляет все документы из данного корпуса.

Альтернативные потоки:

E1: При возникновении ошибки система отправляет информацию о ней клиенту.

**4. Создание нового документа**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для добавления документа с переданными необходимыми данными в качестве аргументов.

Основной поток:

1. Клиент выполняет вызов функции для добавления документа (E1).
2. Сервис отправляет запрос в реляционную СУБД для сохранения информации о документе (E2).
3. Сервис отправляет запрос в документо-ориентированную СУБД для сохранения информации о документе (S1).

Подпотоки:

S1: Индексация документа.

Альтернативные потоки:

E1: Клиент сначала выполняет поиск корпуса для добавления документа.

E2: При возникновении ошибки система отправляет информацию о ней клиенту.

**5. Изменение документа**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для изменения документа с переданной новой информацией в качестве аргументов.

Основной поток:

1. Клиент выполняет вызов функции для изменения документа.
2. Сервис отправляет запрос в реляционную СУБД для изменения информации о корпусе (E1).
3. Сервис отправляет запрос в документо-ориентированную СУБД для изменения информации о документе (E2).

Альтернативные потоки:

E1: При возникновении ошибки система отправляет информацию о ней клиенту.

E2: При изменении текста документа происходит переиндексация текста (удаления старых индексов, добавление новых).

**6. Удаление документа**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для удаления документа с переданным идентификатором в качестве аргумента.

Основной поток:

1. Клиент выполняет вызов функции для удаления корпуса (S1).
2. Сервис отправляет запрос в реляционную и документо-ориентированную СУБД для удаления информации о корпусе (E1).

Подпотоки:

S1: Система удаляет все индексы для данного документа.

Альтернативные потоки:

E1: При возникновении ошибки система отправляет информацию о ней клиенту.

**7. Регистрация нового пользователя**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для добавления информации о новом пользователе с переданными необходимыми данными в качестве аргументов.

Основной поток:

1. Клиент выполняет вызов функции для добавления информации о пользователе.
2. Сервис отправляет запрос в реляционную СУБД для сохранения информации о пользователе (E1).

Альтернативные потоки:

E1: При возникновении ошибки система отправляет информацию о ней клиенту.

**8. Изменение информации о пользователе**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для изменения информации о пользователе с переданной новой информацией в качестве аргументов.

Основной поток:

1. Клиент выполняет вызов функции для изменения информации о пользователе.

2. Сервис отправляет запрос в реляционную СУБД для изменения информации о пользователе (E1).

Альтернативные потоки:

E1: При возникновении ошибки система отправляет информацию о ней клиенту.

**9. Удаление информации о пользователе**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для удаления информации о пользователе с переданным идентификатором в качестве аргумента.

Основной поток:

1. Клиент выполняет вызов функции для удаления информации о пользователе (S1).
2. Сервис отправляет запрос в реляционную СУБД для удаления информации о пользователе (E1).

Подпотоки:

S1: Система удаляет все корпуса данного пользователя.

Альтернативные потоки:

E1: При возникновении ошибки система отправляет информацию о ней клиенту.

**10. Получение информации о пользователе**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для получения информации о пользователе с переданными логином и паролем в качестве аргументов.

Основной поток:

1. Клиент выполняет вызов функции для получения информации о пользователе.
2. Сервис отправляет запрос в реляционную СУБД для получения информации о пользователе и отправляет её клиенту (E1).

Альтернативные потоки:

E1: Если такой пользователь не найден, возвращается пустая ссылка.

**11. Получение информации о корпусе**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для получения информации о корпусе с переданным идентификатором в качестве аргумента.

Основной поток:

1. Клиент выполняет вызов функции для получения информации о корпусе.
2. Сервис отправляет запрос в реляционную СУБД для получения информации о корпусе и отправляет её клиенту (E1).

Альтернативные потоки:

E1: Если такой корпус не найден, возвращается пустая ссылка.

**12. Получение информации о документе**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для получения информации о документе с переданным идентификатором в качестве аргумента.

Основной поток:

1. Клиент выполняет вызов функции для получения информации о документе.
2. Сервис отправляет запрос в реляционную и документо-ориентированную СУБД для получения информации о документе и отправляет её клиенту (E1).

Альтернативные потоки:

E1: Если такой документ не найден, возвращается пустая ссылка.

**13. Поиск документа**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для поиска документов в корпусе по названию с необходимой информацией в качестве аргументов.

Основной поток:

1. Клиент выполняет вызов функции для поиска документов по названию.
2. Сервис отправляет запрос в реляционную СУБД для поиска документов и отправляет их клиенту (E1).

Альтернативные потоки:

E1: Если такие документы не найдены, возвращается пустой массив.

**14. Поиск документов по ключевым словам**

Акторы: Клиент (Другие приложение).

Краткое описание: Вызов функции для поиска документов в корпусе по ключевым словам с необходимой информацией в качестве аргументов.

Основной поток:

1. Клиент выполняет вызов функции для поиска документов по ключевым словам.
2. Сервис выполняет поиск по специальной структуре индексов для получения идентификаторов подходящих документов, затем выполняется запрос в реляционную и документо-ориентированной СУБД для получения полной информации о документе (E1).

Альтернативные потоки:

E1: Если такие документы не найдены, возвращается пустой массив.

# Приложение В. Техническое задание

**ИНДЕКСАЦИИ ХРАНИЛИЩА ТЕКСТОВ НА ОСНОВЕ ЕСТЕСТВЕННО-ЯЗЫКОВОЙ АДРЕСАЦИИ**

Техническое задание

Листов 8

Руководитель разработки
к.ф.-м.н, доцент кафедры информационных технологий в бизнесе.
_____ Лядова Л.Н.
“_____”_____2018

Ответственный исполнитель
Студентка 4-ого курса группы ПИ-14-1
факультета экономики, менеджмента и бизнес-информатики
_____Симонова Н.А.
“_____”_____2018

2018

# 1. Введение

## 1.1. Наименование системы

Полное наименование: Модуль для хранения, поиска и получения текстов и корпусов с использованием индексации на основе естественно-языковой адресации в виде WCF-сервиса.

Краткое наименование: WCF-сервис «Хранение».

## 1.2. Область применения

Сервис предназначен для использования в качестве хранилища данных (корпусов, информации о пользователях) как часть программной системы для анализа английского научного текста в виде онлайн-портала «PaperCat» (далее программная система).

# 2. Основания для разработки

## 2.1. Основание для проведения разработки

Основанием для разработки является приказ об утверждении тем, руководителей выпускных квалификационных работ студентов образовательной программы «Программная инженерия» факультета экономики, менеджмента и бизнес-информатики, утвержденный 12 декабря 2018 г. № 8.2.6.2-13/3011-03.

## 2.2. Наименование и условное обозначение темы разработки

Название темы разработки – «Разработка WCF-сервиса «Хранение». Условное обозначение темы разработки (шифр темы) – «РСХ-001».

# 3. Назначение разработки

Современные корпусные исследования требуют анализа большого объема текстов, поэтому лингвистам необходимы программы, которые, во-первых, позволяют хранить и обрабатывать большие объемы информации, а во-вторых, предоставляют совместный доступ к корпусам для разделения работы между несколькими лингвистами.

С помощью разрабатываемой системы не только лингвисты смогут продолжать корпусные исследования загруженных корпусов, но и в перспективе студенты могут

перед отправлением статьи на рецензию экспертам получить рекомендации на основе соответствия статистическим данным.

Для обеспечения таких возможностей данным программам необходимо хранить как отдельные тексты, загружаемые студентами, так и целые корпусы для их дальнейшего исследования. Иногда корпусы могут достигать приличного объема, и поэтому необходимы эффективные алгоритмы поиска по документам. Одним из возможных решений является использование индексации на основе естественно-языковой адресации, которая подразумевает использование цифровых кодов букв в качестве индекса.

### 3.1. Функциональное назначение

Функциональным назначением программы является хранение, поиск и предоставление документов через сервис. Эффективность поиска должна обеспечиваться реализацией индексации на основе естественно-языковой адресации.

### 3.2. Эксплуатационное назначение

Сервис должен эксплуатироваться как часть программной системы, которая может быть использована в научных и учебных целях для анализа научного стиля и соответствия текста.

## 4. Требования к программе

### 4.1. Требования к функциональным характеристикам

#### 4.1.1. Требования к составу выполняемых функций

Сервис должен обеспечить возможность выполнения перечисленных ниже функций:

1. Создание/удаление/изменение корпусов текстов.
2. Добавление/удаление/изменение документов в корпусы.
3. Добавление информации о пользователе (регистрация пользователя).
4. Удаление/изменение информации о пользователе.
5. Получение информации о пользователе.
6. Получение информации о документе.
7. Получение информации о корпусе.

8. Поиск документа в базе.
9. Поиск фрагмента документа по тексту документов.

Функциональные требования также представлены на диаграмме прецедентов (рис. А.1).

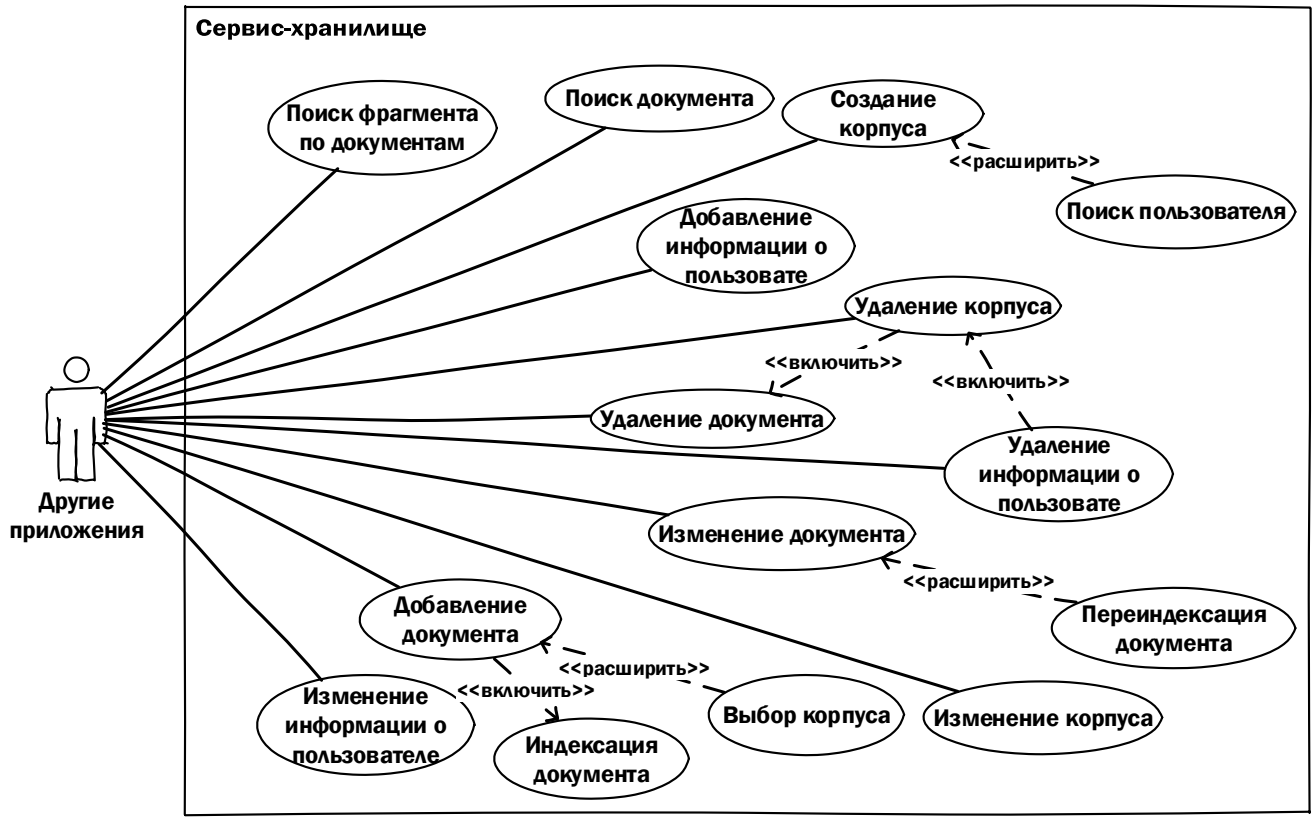


*Рисунок А.1. Диаграмма прецедентов*

### 4.1.2. Требования к организации входных данных

Источниками входных данных являются другие сервисы, реализованные в программной системе. Должны быть разработаны контрактов сервиса в качестве стандартов взаимодействия между сервисами.

### 4.1.3. Требования к организации выходных данных

Выходные данные должны посылаться так же в другие сервисы согласно разработанным контрактам сервиса.

### 4.1.4. Требования к временным характеристикам

В связи с тем, что время обработки не является критичным для разрабатываемой системы (не является системой реального времени), время обработки запросов должно быть ограничено персональными психологическими ощущениями пользователя.

## 4.2. Требования к надежности

Надежность сервиса должна обеспечиваться предварительным тестированием системы, наличием обработчиков исключений. Защита информации от нарушений целостности обеспечивается используемыми СУБД.

### 4.2.1. Требования к обеспечению надежного (устойчивого) функционирования программы

Надежное (устойчивое) функционирование сервиса обеспечивается используемой облачной платформой Azure.

### 4.2.2. Время восстановления после отказа

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать времени, необходимого на перезагрузку операционной системы и запуск программы, при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

### 4.2.3. Отказы из-за некорректных действий оператора

Отказы программы вследствие некорректных действий оператора (пользователя) не возможны, так как пользователями сервиса являются другие сервисы.

### 4.2.4. Последствия отказов системы или отказов наиболее важных функций

Отказ системы или наиболее важных функций не должен влиять на работу ОС и других программ. Отказ сервис-хранилища текстов не должен влиять на работу остальной системы.

### 4.2.5. Допустимый объем данных, утрачиваемых в случае отказа

Допустимый объем данных, утрачиваемых в случае отказа, не должен превышать объема информации об изменениях в записях, изменяемых в момент отказа.

#### 4.2.6. Функции, необходимые для обеспечения устойчивости к ошибкам

Устойчивость к ошибкам должна обеспечиваться функциями проверки входных данных на соответствие стандартам взаимодействия.

### 4.3. Условия эксплуатации

Сервис может использоваться сервисами с любых платформ, поддерживающими стандарты взаимодействия. Сервис должен быть опубликован на облачной платформе Azure по специальной подписке.

#### 4.3.1. Климатические условия эксплуатации

Климатические условия эксплуатации, при которых должны обеспечиваться заданные характеристики, должны удовлетворять требованиям, предъявляемым к техническим средствам в части условий их эксплуатации.

#### 4.3.2. Требования к видам обслуживания

См. Требования к обеспечению надежного (устойчивого) функционирования программы.

#### 4.3.3. Требования к численности и квалификации персонала

Требования к составу и квалификации персонала не предъявляются.

### 4.4. Требования к составу и параметрам технических средств

Требования к составу и параметрам технических средств не предъявляются, так как сервис расположен на серверах Azure.

### 4.5. Требования к информационной и программной совместимости

Данная программа должна быть совместима с другими сервисами программной системы, посредством поддержки стандартов взаимодействия (контрактов сервиса).

#### 4.5.1. Требования к информационным структурам и методам решения

Информационная структура входных и выходных данных должна соответствовать контрактам данных в программной системе.

Для индексации хранимых документов должны использоваться индексация на основе естественно-языковой адресации.

#### 4.5.2. Требования к исходным кодам и языкам программирования

Исходные коды программы должны быть реализованы на языке C#. В качестве интегрированной среды разработки программы должна быть использована среда Visual Studio 2017.

#### 4.5.3. Требования к программным средствам, используемым программой

Сервис должен быть расположен на серверах Azure. В качестве нереляционной СУБД должна быть использована документо-ориентированная Azure Cosmos DB, в качества реляционной – SQL Azure с помощью технологии Entity Framework.

#### 4.5.4. Требования к защите информации и программ

Целостность данных обеспечивается используемыми СУБД. Требования к защите информации не предъявляются.

### 4.6. Требования к транспортированию и хранению

Сервис предоставляется посредством сети Интернет через URL. Сервис должен быть опубликован на облачной платформе Azure.

### 4.7. Специальные требования

Программа должна обеспечивать взаимодействие с другими сервисами посредством сети Интернет с помощью разработанных контрактов сервиса.

## 5. Требования к программной документации

### 5.1. Предварительный состав программной документации

В состав программной документации должны входить:

1) руководство программиста;

2) отчет о проделанной работе.

Руководство пользователя не предполагается, так как программа является сервисом и не имеет пользовательского интерфейса. Руководство администратора так

же не предполагается, так как администрирование будет проводиться только со стороны облачной платформы Azure.

## 6. Технико-экономические показатели

Стоимость разработки программы составила 40 486 рублей, а её расчет приведен в Технико-экономическом обосновании. Владение и поддержка программы будут бесплатными. Предполагаемое число использования программы в год – 250 сеансов работы для лингвистов, 10 для студентов.

## 7. Стадии и этапы разработки

### 7.1. Стадии разработки

Разработка должна быть проведена в три стадии:

1. Анализ задачи и разработка технического задания.
2. Проектирование приложения.
3. Реализация и тестирование сервиса.

Исполнителем на всех стадиях и этапах разработки является студентка НИУ ВШЭ-Пермь Симонова Наталья Андреевна.

### 7.2. Этапы разработки

На стадии «Анализ задачи и разработка технического задания» должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания, а также анализ предметной области и автоматизируемых бизнес-процессов.

Срок окончания: «22» января 2018 года.

На стадии «Проектирование приложения» должны быть выполнены перечисленные ниже этапы работ:

1. Моделирования вариантов использования.
2. Создание модели данных.
3. Проектирование архитектуры приложения.

Срок окончания: «19» марта 2018 года.

На стадии «Реализация и тестирование сервиса» должны быть выполнены перечисленные ниже этапы работ:

1. Разработка сервиса.
2. Разработка программной документации.

3. Испытания сервиса.

Срок окончания: «28» апреля 2016 года.

### 7.3. Содержание работ по этапам

Содержание работ по этапам представлено в таблице А.1.

***Таблица А.1. Содержание работ по этапам***

| Стадия | Этап | Проводимые работы |
|---|---|---|
| Анализ задачи и разработка технического задания | Анализа и разработки технического задания | 1. Постановка задачи.<br>2. Определение и уточнение требований к техническим средствам.<br>3. Определение требований к программе.<br>4. Определение стадий, этапов и сроков разработки программы и документации на нее.<br>5. Выбор языков программирования.<br>6. Согласование и утверждение технического задания.<br>7. Описание методологий (WCF-сервис, естественно-языковая адресация). |
| Проектирование приложения | Моделирования вариантов использования | Создание диаграммы вариантов использования UML |
| | Создания модели данных | Создание концептуальной модели предметной области в нотации ERD. |
| | Проектирования архитектуры сервиса | Описание архитектуры системы в одной из стандартных нотаций, а также производится построение архитектуры использования баз данных (реляционных и нереляционных). |
| Реализация и тестирование сервиса | Разработки сервиса | Выполнение работ по программированию (кодированию) и отладке программы. Результаты тестирования и отладки документируются |
| | Разработки программной документации | Разработка программных документов в соответствии с требованиями ГОСТ 19.101-77 |
| | Испытаний | Публикация сервиса на портале Azure, и проведение испытания совместимости с другими сервисами. Результаты тестирования и отладки также документируются. |

## 8. Порядок контроля и приемки

Контроль и приемка разработки выполняются в тестирования сервиса, которое включает в себя функциональное и интеграционное тестирование, а также демонстрации сервиса руководителю.

# Приложение С.Технико-экономическое обоснование

**ИНДЕКСАЦИИ ХРАНИЛИЩА ТЕКСТОВ НА ОСНОВЕ ЕСТЕСТВЕННО-ЯЗЫКОВОЙ АДРЕСАЦИИ**

Технико-экономическое обоснование

Листов 4

Руководитель разработки
к.ф.-м.н, доцент кафедры информационных технологий в бизнесе.
_____ Лядова Л.Н.
“_____”_____2018

Ответственный исполнитель
Студентка 4-ого курса группы ПИ-14-1
факультета экономики, менеджмента и бизнес-информатики
_____Симонова Н.А.
“_____”_____2018

2018

# 1. Расчет стоимости разработки и поддержки сервиса

Итоговая стоимость реализации сервиса, в которую включена стоимость анализа требований, проектирования, разработки, тестирования и публикации сервиса, представлена в таблице В.1 и составляет 40 486 руб. В качестве стоимости часа взята средняя стоимость для рынка города Перми в 2017 году.

Расчет стоимости проводился по методологии Work Breakdown Structure (WBS) или иерархическая структура работ, которая подразумевает последовательную декомпозицию работ, пока необходимое количество часов на разработку не будет очевидным.

***Таблица В.1. Расчет стоимости разработки***

| Работа | Количество часов | Профессиональный стандарт | Стоимость часа | Итоговая стоимость (руб.) |
|---|---|---|---|---|
| 1. Проанализировать: | 18 | | | 4407 |
| 1.1. Существующие технологии хранения больших массивов текстовой информации. | 5 | Системный аналитик | 250 | 1250 |
| 1.2. Протоколы передачи. | 2 | Системный аналитик | 250 | 500 |
| 1.3. Подходы к поиску и индексации, в том числе изучить принцип индексации на основе естественно-языковой адресации. | 8 | Системный аналитик | 250 | 2000 |
| 1.4. Разработать техническое задание. | 3 | Технический писатель | 219 | 657 |
| 2. Произвести моделирование web-сервиса для хранения корпусов: | 14 | | | 5625 |
| 2.1. Произвести моделирование вариантов использования с помощью диаграммы прецедентов. | 3 | Специалист по информационным системам | 375 | 1125 |
| 2.2. Произвести моделирование предметной области с помощью диаграммы классов. | 8 | Специалист по информационным системам | 375 | 3000 |
| 2.3. Произвести моделирование архитектуры системы с помощью диаграммы компонентов. | 3 | Архитектор программного обеспечения | 500 | 1500 |
| 3. Реализовать web-сервис: | 95 | | | 30454 |
| 3.1. Реализовать CRUD-функции для корпусов. | 39 | | | 14625 |
| 3.1.1. Реализовать добавление корпусов. | 3 | Программист | 375 | 1125 |
| 3.1.2. Реализовать удаление корпусов. | 3 | Программист | 375 | 1125 |
| 3.1.3. Реализовать изменение корпусов. | 3 | Программист | 375 | 1125 |
| 3.1.4. Реализовать добавление документов. | 7 | Программист | 375 | 2625 |
| 3.1.5. Реализовать удаление документов. | 7 | Программист | 375 | 2625 |
| 3.1.6. Реализовать изменение документов. | 7 | Программист | 375 | 2625 |
| 3.1.10. Реализовать добавление пользователей. | 3 | Программист | 375 | 1125 |

| Работа | Количество часов | Профессиональный стандарт | Стоимость часа | Итоговая стоимость (руб.) |
|---|---|---|---|---|
| 3.1.11. Реализовать удаление пользователей. | 3 | Программист | 375 | 1125 |
| 3.1.12. Реализовать изменение пользователей. | 3 | Программист | 375 | 1125 |
| 3.2. Реализовать индексацию на основе естественно-языковой адресации. | 20 | | | 8500 |
| 3.2.1. Разработать структуру для хранения индексов. | 8 | Архитектор программного обеспечения | 500 | 4000 |
| 3.2.2. Реализовать функции для работы со структурой. | 8 | Программист | 375 | 3000 |
| 3.2.3. Реализовать изменение структуры. | 4 | Программист | 375 | 1500 |
| 3.3. Опубликовать созданный сервис. | 3 | Программист | 375 | 1125 |
| 3.4. Произвести функциональное и интеграционное тестирование. | 33 | | | 6204 |
| 3.4.1. Протестировать добавление корпусов. | 2 | Специалист по тестированию в области информационных технологий | 188 | 376 |
| 3.4.2. Протестировать удаление корпусов. | 2 | Специалист по тестированию в области информационных технологий | 188 | 376 |
| 3.4.3. Протестировать изменение корпусов. | 2 | Специалист по тестированию в области информационных технологий | 188 | 376 |
| 3.4.4. Протестировать добавление документов. | 4 | Специалист по тестированию в области информационных технологий | 188 | 752 |
| 3.4.5. Протестировать удаление документов. | 4 | Специалист по тестированию в области информационных технологий | 188 | 752 |
| 3.4.6. Протестировать изменение документов. | 4 | Специалист по тестированию в области информационных технологий | 188 | 752 |
| 3.4.7. Протестировать добавление индексов. | 3 | Специалист по тестированию в области информационных технологий | 188 | 564 |

| Работа | Количество часов | Профессиональный стандарт | Стоимость часа | Итоговая стоимость (руб.) |
|---|---|---|---|---|
| 3.4.8. Протестировать удаление индексов. | 3 | Специалист по тестированию в области информационных технологий | 188 | 564 |
| 3.4.9. Протестировать изменение индексов. | 3 | Специалист по тестированию в области информационных технологий | 188 | 564 |
| 3.4.10. Протестировать добавление пользователей. | 2 | Специалист по тестированию в области информационных технологий | 188 | 376 |
| 3.4.11. Протестировать удаление пользователей. | 2 | Специалист по тестированию в области информационных технологий | 188 | 376 |
| 3.4.12. Протестировать изменение пользователей. | 2 | Специалист по тестированию в области информационных технологий | 188 | 376 |
| **ИТОГО:** | **127** | | | **40 486** |

Стоимость поддержки программы включает в себя только стоимость хостинга для сервиса. На данный момент сервис может быть бесплатно опубликован на портале Azure по подписке для студентов и некоторых некоммерческих проектов. Данное хранилище было выбрано, так как является бесплатным, имеет удобный и понятный интерфейс, а также предоставляет возможность использования NoSQL базы данных как сервис.

Проект является некоммерческим, поэтому расчет срока окупаемости не производится.

## 2. Обоснование выбранных технологий

В качестве инструментов разработки и подходов были выбраны:

1. C# в качестве языка программирования, так как он поддерживает объектно-ориентированные технологии и является высокоуровневым, постоянно развивающимся языком.
2. Visual Studio 2015, так как данная IDE поддерживает C#, является удобным инструментом разработки кода (имеет встроенные возможности

рефакторинга, навигации по коду, исправлений и отладки), а также имеет встроенную поддержку интеграции с сервисами Azure и является бесплатной.

3. Windows Communication Foundation, так как в отличие от обычного сервиса, WCF позволяет иметь несколько точек доступа, поддерживает сессии на уровне сервиса, а не метода, позволяет использовать различные протоколы и предоставляет некоторые другие расширенные возможности.
4. Документо-ориентированный подход, так как он обеспечивает более простую работу с неструктурированными текстовыми данными, а также многие NoSQL базы специально разработаны для хранения крупных объемов данных с более простой поддержкой масштабирования. Преимуществами данного подхода также являются гибкость схемы, легкая масштабируемость и эффективность разработки.
5. Технология Entity Framework для работы с реляционной базой данных, так как она обеспечивает независимость базы данных и более быструю реализацию, а также поддерживает LINQ, транзакции и параллелизм.
6. Cosmos DB в качестве СУБД для хранения документов, так как данная СУБД уже является сервисом, не требует сложных настроек, поддерживает различные языки, хоть и предоставляется только по подписке.
7. Индексация на основе естественно языковой адресации, так как данный подход позволяет уменьшить необходимый объем памяти (нет необходимости отдельно хранить индексы), а значит позволит, при необходимости, уменьшить затраты на память (например, в облачном хранилище).

# Приложение D.Руководство программиста

**ИНДЕКСАЦИИ ХРАНИЛИЩА ТЕКСТОВ НА ОСНОВЕ ЕСТЕСТВЕННО-ЯЗЫКОВОЙ АДРЕСАЦИИ**

Руководство программиста

Листов 6

Руководитель разработки
к.ф.-м.н, доцент кафедры информационных технологий в бизнесе.
_____ Лядова Л.Н.
“_____”_____2018

Ответственный исполнитель
Студентка 4-ого курса группы ПИ-14-1
факультета экономики, менеджмента и бизнес-информатики
_____Симонова Н.А.
“_____”_____2018

2018

# 1. Назначение и условия применения программ

## 1.1. Назначение системы

Назначением программы является хранение, поиск и предоставление документов через сервис. Эффективность поиска обеспечивается реализацией индексации на основе естественно-языковой адресации.

## 1.2. Функции, выполняемые системой

Сервис обеспечивает возможность выполнения перечисленных ниже функций:

1. Создание/удаление/изменение корпусов текстов.
2. Добавление/удаление/изменение документов в корпусы.
3. Добавление/удаление/изменение аннотаций к текстам.
4. Добавление/удаление/изменение других представлений текстов.
5. Добавление информации о пользователе (регистрация пользователя).
6. Удаление/изменение информации о пользователе.
7. Поиск документа в базе.
8. Поиск фрагмента документа по тексту документов.

## 1.3. Требования к составу и параметрам технических средств

Требования к составу и параметрам технических средств не предъявляются, так как сервис расположен на серверах Azure.

## 1.4. Требования к программному обеспечению

Для работы над программой необходима среда разработки Visual Studio 2013 или выше. Для работы программы с облачными базами данных Azure необходима подписка Azure.

# 2. Характеристика программы

## 2.1. Характеристика используемых средств

Исходные коды программы реализованы на языке C#. В качестве интегрированной среды разработки программы использована среда Visual Studio 2017.

Программа использует SQL Azure для хранения реляционных данных. Обращение к БД происходит с помощью Entity Framework и LINQ.

Программа использует Azure Cosmos DB для хранения нереляционных данных. Обращение к БД происходит с помощью API и LINQ.

## 2.2. Структура программы

Архитектуры системы создана представленная на диаграмме компонентов (рис. С.1).

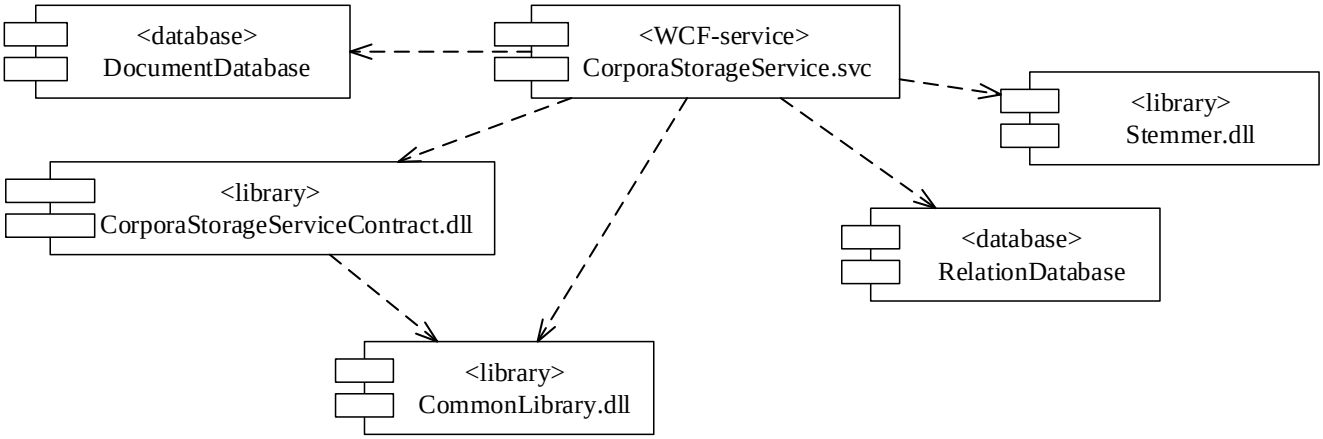


***Рисунок С.1. Диаграмма компонентов сервиса-хранилища***

CorporaStorageServise.svc – компонент реализации сервиса, который использует библиотеку с контрактом сервиса (CorporaStorageServiseContract.dll), то есть с описанием его интерфейса, а также общую библиотеку (CommonLibrary.dll), в которую войдут все необходимые для реализации стандартов взаимодействия классы.

Для хранения реляционных данных будет использоваться также располагаемая на сервисах Azure база данных SQL Azure (RelationDatabase.svc), предоставляемая как сервис. DocumentDatabase.svc – Cosmos DB, нереляционная база данных как сервис для хранения документов. Также используется сторонняя реализация стеммера по алгоритму стемминга Портера (Stemmer.dll).

## 2.3. Структура баз данных

Информация о пользователях и корпусах содержится в реляционной СУБД (рис. С.2), а также ключи нереляционной части хранящихся документов в корпусах (DocumentId), названия документов и оригинальный файл.

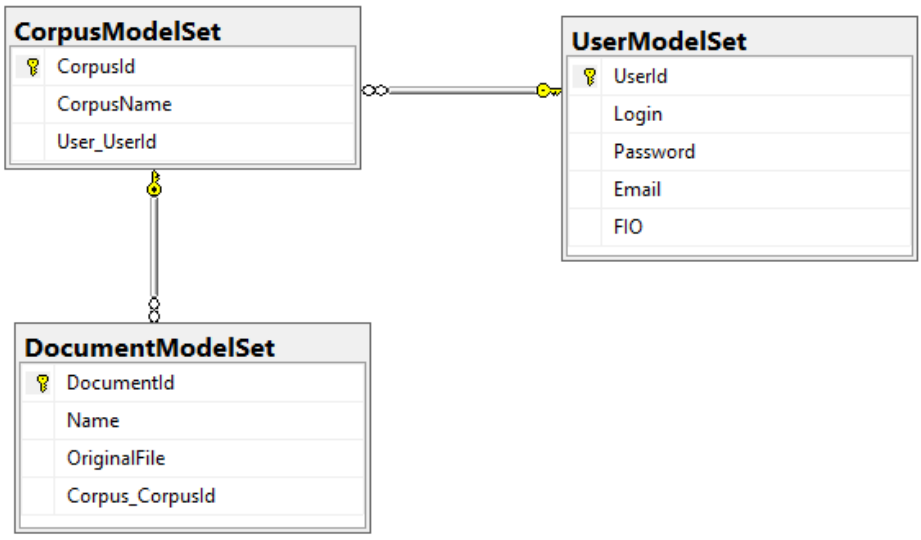


***Рисунок С.2. Схема реляционной базы данных***

Работа с реляционной базой данной осуществляется с помощью Entity Framework, ниже представлена получившаяся модель (рис. С.3).

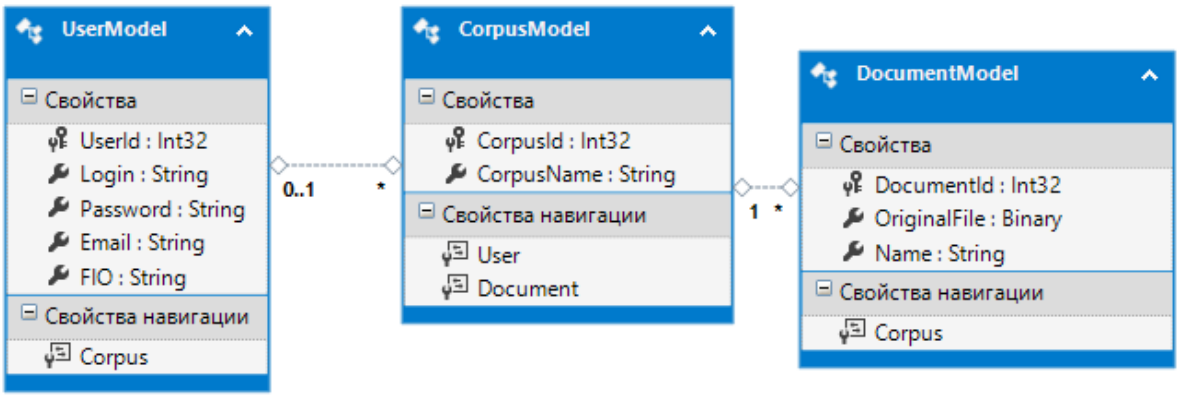


***Рисунок С.3. Entity Designer Diagram***

В документо-ориентированной базе данных хранится информация о документах. Так как документы в документо-ориентированных СУБД не имеют четкой структуры, была определена примерная схема информации, которая будет храниться в формате JSON (рис. С.4).

| Документ | |
|---|---|
| DocumentId<br>URI | |
| Annotation | 0..1 |
| FlatText | 0..1 |
| HXML-file | 0..1 |

***Рисунок С.4. Примерная структура документа в документо-ориентированной базе данных***

Гибкость структуры позволит хранить любую дополнительную информацию о документе помимо определенной выше.

Древовидная структура для хранения индексов на основе естественно-языковой адресации также храниться в документо-орентированной базе данных в виде файлов с серилизованными объектами.

Классы для работы с индексацией (рис. С.5) на основе естественно-языковой адресации включают в себя классы структуры индексов: NLAStructure и NLAHeadDocument, которые включают в себя словарь ссылок на класс IndexStructureObject.

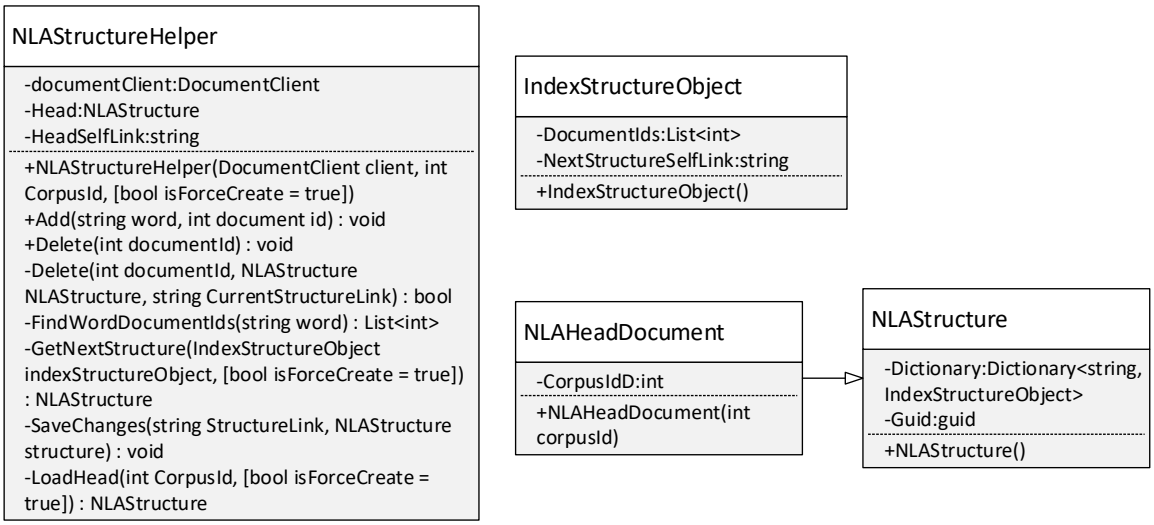


***Рисунок С.5. Диаграмма классов для работы с индексацией***

IndexStructureObject содержит следующий уровень структуры и/или ссылки на идентификаторы документов, в которых содержится соответствующее ключевое слово. Класс NLAStructureHelper содержит методы для добавления идентификатора документа к соответствующему ключевому слову, а также удаления идентификатора документа из всей структуры. Также в классе NLAStructureHelper содержаться вспомогательные методы.

## 2.4. Временные характеристики

Время обработки запроса зависит от качества Интернет-соединения, а также объема обрабатываемых данных, однако время обработки запросов ограничено персональными психологическими ощущениями пользователя.

# 3. Обращение к программе

Обращение к программе происходит путем вызова методов WCF-сервиса, определенных в контракте, одним из следующих способов:

1. Добавление ссылки на службу в проект Visual Studio.
2. Использование класса ChannelFactory и предоставляемой библиотеки с контрактом класса.

Более подробную информацию можно посмотреть по ссылкам:

1. https://docs.microsoft.com/ru-ru/dotnet/framework/wcf/how-to-create-a-wcf-client (Практическое руководство. Создание клиента Windows Communication Foundation)
2. https://msdn.microsoft.com/en-us/library/system.servicemodel.channelfactory.aspx?cs-save-lang=1&cs-lang=csharp#code-snippet-2

(ChannelFactory Class, Microsoft Developer Network)

В контракте сервиса определены следующие методы:

1. AutorizeUser(string Login, string Password) : User – возвращает информацию о пользователе по логину и паролю.
2. GetCorpora(int UserId) : Corpus[] – возвращает список всех корпусов пользователя.
3. GetDocuments(int CorpusId) : Document[] – возвращает список всех документов в корпусе.
4. GetDocument(int Document) : Document – возвращает информацию о документе.
5. MakeCorpus(string Name, Document[] documents) : int – создает общий корпус с выбранными документами (можно создать и пустой корпус, передав пустой массив).
6. MakeCorpus(string Name, Document[] documents, int UserId) : int – создает частный корпус с выбранными документами (можно создать и пустой корпус, передав пустой массив).
7. AddUser(User user) : int – регистрация нового пользователя.
8. UpdateUser(User user) : void – изменение информации о пользователе. По старому идентификатору обновляются все поля.
9. DeleteUser(int userId) : void – удаление информации о пользователе по идентификатору. При этом происходит удаление всех корпусов пользователя.
10. AddDocument(Document document) : int – добавление нового документа.

11. UpdateDocument(Document document) : void – изменение документа. По старому идентификатору обновляются все поля.
12. DeleteDocument(int documentId) : void – удаление документа по идентификатору.
13. UpdateCorpus(Corpus corpus) : void – изменение корпуса. По старому идентификатору обновляются все поля.
14. DeleteCorpus(int corpusId) : void – удаление корпуса по идентификатору. При этом происходит удаление всех документов корпуса.
15. FindDocuments(string Name, int corpusId) : Document[] – поиск документа по имени.
16. FindText(string Text) : Document[] – поиск фрагмента по документам.

## 4. Входные и выходные данные

Источниками входных данных являются другие сервисы, реализованные в программной системе. Выходные данные должны посылаться так же в другие сервисы согласно стандартам взаимодействия, реализованные в программной системе.

Стандартами взаимодействия являются вызов методов сервиса, а также использование классов, определенных в контракте сервиса:

Были выделены следующие сущности и атрибуты:

1. User – пользователь с информацией о ФИО (nullable), электронной почтой (not nullable), логине (not nullable) и пароле (not nullable) пользователя.
2. Corpus – корпус с информацией о названии корпуса (not nullable).
3. Document – документ, содержащий оригинальный файл (nullable), плоский текст (not nullable), аннотацию (nullable), строку HTML-разметки (nullable) и название (not nullable).

## 5. Сообщения

При работе с сервисом в случае успешной работы метод возвращает данные в соответствии с сигнатурой метода, например:

1. При вызове метода FindDocuments вернется массив документов.
2. При вызове метода DeleteCorpus нет возвращаемого значения.

Возможные виды специфичных ошибок при работе с сервисом представлены в таблице С.1. Остальные возможные виды выдаваемых ошибок связаны с WCF-

сервисами и используемыми базами данных, сведенья о них можно найти в соответствующей документации.

***Таблица С.1. Сообщения специфичных ошибок***

| Класс ошибки | Сообщение ошибки | Описание ошибки | Требуемые действия при возникновении ошибки. |
|---|---|---|---|
| Document | Такого документа не существует | Метод UpdateDocument. Документа с данным идентификатором нет в базе данных. | Изменить идентификатор документа или добавить данный документ в базу. |
| Corpus | Такого корпуса не существует | Метод UpdateCorpus. Корпуса с данным идентификатором нет в базе данных. | Изменить идентификатор корпуса или добавить данный корпус в базу. |
| User | Другой пользователь с таким логином уже существует | Метод UpdateUser. В базе данных существует другой пользователь с таким логином | Изменить новый логин изменяемого пользователя. |
| User | Такого пользователя не существует | Метод UpdateUser. Пользователя с данным идентификатором нет в базе данных. | Изменить идентификатор пользователя или добавить данного пользователя в базу. |
| User | Пользователь с таким логином уже существует | Метод AddUser. В базе данных существует пользователь с таким логином | Изменить логин добавляемого пользователя. |

```
namespace CorporaStorageServiseContract
{
   [ServiceContract]
   public interface ICorporaStorageService
   {
      [OperationContract]
      [WebGet(ResponseFormat = WebMessageFormat.Json, UriTemplate = "AutorizeUser?Login={Login}&Password={Password}")]
      User AutorizeUser(string Login, string Password);

      [OperationContract]
      [WebGet(ResponseFormat = WebMessageFormat.Json, UriTemplate = "GetCorpora?UserId={UserId}")]
      Corpus[] GetCorpora(int UserId);

      [OperationContract]
      [WebGet(ResponseFormat = WebMessageFormat.Json, UriTemplate = "GetDocuments?CorpusId={CorpusId}")]
      Document[] GetDocuments(int CorpusId);

      [OperationContract]
      [WebGet(ResponseFormat = WebMessageFormat.Json, UriTemplate = "GetDocument?DocumentId={DocumentId}")]
      Document GetDocument(int DocumentId);

      [OperationContract]
      [WebInvoke(Method = "POST",
      BodyStyle = WebMessageBodyStyle.Wrapped,
      ResponseFormat = WebMessageFormat.Json,
         UriTemplate = "MakeCommonCorpus")]
      int MakeCommonCorpus(string Name, Document[] documents);

      [OperationContract]
      [WebInvoke(Method = "POST",
      BodyStyle = WebMessageBodyStyle.Wrapped,
      ResponseFormat = WebMessageFormat.Json,
         UriTemplate = "MakeCorpus")]
      int MakeCorpus(string Name, Document[] documents, int UserId);

      [OperationContract]
      [WebInvoke(Method = "POST",
      BodyStyle = WebMessageBodyStyle.Wrapped,
      ResponseFormat = WebMessageFormat.Json,
         UriTemplate = "AddUser")]
      int AddUser(User user);

      [OperationContract]
      [WebInvoke(Method = "POST",
      BodyStyle = WebMessageBodyStyle.Wrapped,
      ResponseFormat = WebMessageFormat.Json,
         UriTemplate = "UpdateUser")]
      void UpdateUser(User user);

      [OperationContract]
      [WebGet(ResponseFormat = WebMessageFormat.Json, UriTemplate = "DeleteUser?userId={userId}")]
      void DeleteUser(int userId);

      [OperationContract]
      [WebInvoke(Method = "POST",
      BodyStyle = WebMessageBodyStyle.Wrapped,
```

```
            ResponseFormat = WebMessageFormat.Json,
              UriTemplate = "AddDocument")]
            int AddDocument(Document document);

            [OperationContract]
            [WebInvoke(Method = "POST",
            BodyStyle = WebMessageBodyStyle.Wrapped,
            ResponseFormat = WebMessageFormat.Json,
              UriTemplate = "UpdateDocument")]
            void UpdateDocument(Document document, bool IsTextChanged);

            [OperationContract]
            [WebGet(ResponseFormat = WebMessageFormat.Json, UriTemplate =
"DeleteDocument?id={documentId}")]
            void DeleteDocument(int documentId);

            [OperationContract]
            [WebInvoke(Method = "POST",
            BodyStyle = WebMessageBodyStyle.Wrapped,
            ResponseFormat = WebMessageFormat.Json,
              UriTemplate = "UpdateCorpus")]
            void UpdateCorpus(Corpus corpus);

            [OperationContract]
            [WebGet(ResponseFormat = WebMessageFormat.Json, UriTemplate = "DeleteCorpus?id={corpusId}")]
            void DeleteCorpus(int corpusId);

            [OperationContract]
            [WebInvoke(Method = "POST",
            BodyStyle = WebMessageBodyStyle.Wrapped,
            ResponseFormat = WebMessageFormat.Json,
              UriTemplate = "FindDocuments")]
            Document[] FindDocuments(string Name, int corpusId);

            [OperationContract]
            [WebInvoke(Method = "POST",
            BodyStyle = WebMessageBodyStyle.Wrapped,
            ResponseFormat = WebMessageFormat.Json,
              UriTemplate = "FindText")]
            int[] FindText(string Text, int corpusId);
        }
    }

    namespace PaperCatPortal
    {
        public class CorporaStorageService : ICorporaStorageService
        {
            static HashSet<string> stopwords = new HashSet<string>() { "a", "about", "above", "after", "again",
"against", "all", "am", "an", "and", "any", "are", "aren't", "as", "at", "be", "because", "been", "before", "being", "below",
"between", "both", "but", "by", "can't", "cannot", "could", "couldn't", "did", "didn't", "do", "does", "doesn't", "doing",
"don't", "down", "during", "each", "few", "for", "from", "further", "had", "hadn't", "has", "hasn't", "have", "haven't",
"having", "he", "he'd", "he'll", "he's", "her", "here", "here's", "hers", "herself", "him", "himself", "his", "how", "how's", "i",
"i'd", "i'll", "i'm", "i've", "if", "in", "into", "is", "isn't", "it", "it's", "its", "itself", "let's", "me", "more", "most", "mustn't",
"my", "myself", "no", "nor", "not", "of", "off", "on", "once", "only", "or", "other", "ought", "our", "ours", "ourselves",
"out", "over", "own", "same", "shan't", "she", "she'd", "she'll", "she's", "should", "shouldn't", "so", "some", "such", "than",
"that", "that's", "the", "their", "theirs", "them", "themselves", "then", "there", "there's", "these", "they", "they'd", "they'll",
"they're", "they've", "this", "those", "through", "to", "too", "under", "until", "up", "very", "was", "wasn't", "we", "we'd",
"we'll", "we're", "we've", "were", "weren't", "what", "what's", "when", "when's", "where", "where's", "which", "while",
"who", "who's", "whom", "why", "why's", "with", "won't", "would", "wouldn't", "you", "you'd", "you'll", "you're",
"you've", "your", "yours", "yourself", "yourselves" };

            public int AddDocument(Document document)
```

```
        {
            if (document == null)
                throw new Exception("Параметр не может быть null");
            using (var context = new PaperCatModelContainer())
            {
                DocumentModel documentModel = new DocumentModel(document, context);
                if (documentModel.Corpus == null)
                    throw new Exception("Корпуса с таким id не существует");
                context.DocumentModelSet.Add(documentModel);
                context.SaveChanges();
                document.DocumentId = documentModel.DocumentId;
            }
            DocumentClient client = GetDocumentClient();
            client.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri(Properties.Resources.DataBaseId,
Properties.Resources.DocumentsCollectionId), new AzureDocument(document)).Wait();
            Task.Run(() => AddDocumentToIndexStructure(document));
            return document.DocumentId;
        }

        private void AddDocumentToIndexStructure(Document document)
        {
            string text = document.FlatText;
            text = GetTokens(text);
            NLAStructureHelper nlaHelper = new NLAStructureHelper(GetDocumentClient(), document.CorpusId);

            foreach (string item in text.Split(' ').Distinct())
            {
                nlaHelper.Add(item, document.DocumentId);
            }
        }

        /// <summary>
        /// Очищает от знаков и стоп-слов, стеммит
        /// </summary>
        /// <param name="text"></param>
        /// <returns></returns>
        private string GetTokens(string text)
        {
            string newText = "";
            Regex rgx = new Regex("[^a-zA-Za-яA-Я ]");
            text = rgx.Replace(text, "");
            string[] words = text.Split(' ');
            Stemmer stemmer = new Stemmer();
            for (int i = 0; i < words.Length; i++)
            {
                //убираем стоп-слова
                if (words[i].Length != 1 && !stopwords.Contains(words[i]))
                {
                    words[i] = stemmer.stem(words[i]);
                    newText += " " + words[i];
                }
            }
            return newText.Trim(' ');
        }

        public int AddUser(User user)
        {
            if (user == null)
                throw new Exception("Параметр не может быть null");
            using (var context = new PaperCatModelContainer())
            {
                UserModel userModel = new UserModel(user);
```

```
            if (IsThisLoginExist(user.Login, context))
            {
                throw new Exception("Пользователь с таким логином уже существует");
            }
            context.UserModelSet.Add(userModel);
            context.SaveChanges();
            return userModel.UserId;
        }

    }

    private static bool IsThisLoginExist(string sLogin, PaperCatModelContainer context, int userid = 0)
    {
        var us = context.UserModelSet.Where(u => u.Login == sLogin && u.UserId != userid).FirstOrDefault();
        return us != null;
    }

    public User AutorizeUser(string Login, string Password)
    {
        using (var context = new PaperCatModelContainer())
        {
            var userModel = context.UserModelSet.Where(u => u.Login == Login && u.Password ==
Password).FirstOrDefault();
            if (userModel == null) return null; //такого пользователя нет
            return userModel.GetUserFromModel();
        }

    }

    public void DeleteCorpus(int corpusId)
    {
        using (var context = new PaperCatModelContainer())
        {
            DeleteCorpusInContext(corpusId, context);
            context.SaveChanges();
        }
    }
    private void DeleteCorpusInContext(int corpusId, PaperCatModelContainer context)
    {
        var DocumentList = context.DocumentModelSet.Where(d => d.Corpus.CorpusId == corpusId).ToList();
        DocumentClient client = GetDocumentClient();
        foreach (var item in DocumentList)
        {
            DeleteDocumentInContext(item.DocumentId, context);
            Task.Run(() => DeleteDocumentFromIndexStructure(item.DocumentId, corpusId));
            string SelfLink = GetDocumentSelfLink(item.DocumentId, client);
            if (SelfLink != null)
                client.DeleteDocumentAsync(SelfLink).Wait();
        }
        CorpusModel corpusModel = context.CorpusModelSet.Find(corpusId);
        if (corpusModel != null)
        {
            context.CorpusModelSet.Remove(corpusModel);
        }
    }
    /// <summary>
    //// Возвращает id корпуса, из которого был удален документ
    /// </summary>
    /// <param name="documentId"></param>
    /// <param name="context"></param>
    /// <returns></returns>
    private int DeleteDocumentInContext(int documentId, PaperCatModelContainer context)
```

```
        {
            int corpusId = -1;
            DocumentModel documentModel = context.DocumentModelSet.Find(documentId);
            if (documentModel != null)
            {
                corpusId = documentModel.Corpus.CorpusId;
                context.DocumentModelSet.Remove(documentModel);
            }
            return corpusId;
        }
        public void DeleteDocument(int documentId)
        {
            int corpusid;
            using (var context = new PaperCatModelContainer())
            {
                corpusid = DeleteDocumentInContext(documentId, context);
                context.SaveChanges();
            }
            Task.Run(() => DeleteDocumentFromIndexStructure(documentId, corpusid));
            DocumentClient client = GetDocumentClient();
            string SelfLink = GetDocumentSelfLink(documentId, client);
            if (SelfLink != null)
                client.DeleteDocumentAsync(SelfLink).Wait();
        }

        private static string GetDocumentSelfLink(int documentId, DocumentClient client)
        {
            return
client.CreateDocumentQuery(UriFactory.CreateDocumentCollectionUri(Properties.Resources.DataBaseId,
Properties.Resources.DocumentsCollectionId)).AsEnumerable().Where(d => d.GetPropertyValue<int>("DocumentId") ==
documentId).FirstOrDefault()?.SelfLink;
        }

        private static DocumentClient GetDocumentClient()
        {
            return new DocumentClient(new Uri(Properties.Resources.EndpointUrl),
Properties.Resources.PrimaryKey);
        }
        private void DeleteDocumentFromIndexStructure(int documentId, int corpusId)
        {
            NLAStructureHelper nlaHelper = new NLAStructureHelper(GetDocumentClient(), corpusId, false);
            nlaHelper.Delete(documentId);
        }
        public void DeleteUser(int userId)
        {
            using (var context = new PaperCatModelContainer())
            {
                var corpusList = context.CorpusModelSet.Where(c => c.User.UserId == userId).ToList();
                foreach (var item in corpusList)
                {
                    DeleteCorpusInContext(item.CorpusId, context);
                }
                UserModel UserModel = context.UserModelSet.Find(userId);
                if (UserModel == null)
                    return;
                context.UserModelSet.Remove(UserModel);
                context.SaveChanges();
            }

        }
        public Document[] FindDocuments(string Name, int corpusId)
        {
```

```
            List<DocumentModel> DocumentModelList;
            using (var context = new PaperCatModelContainer())
            {
                DocumentModelList = context.DocumentModelSet.Where(d => d.Corpus.CorpusId == corpusId &&
d.Name.ToLower().Contains(Name.ToLower())).ToList();

                Document[] documents = new Document[DocumentModelList.Count()];
                for (int i = 0; i < DocumentModelList.Count(); i++)
                {
                    documents[i] = GetFullDocument(DocumentModelList[i]);
                }
                return documents;
            }
        }

        private static Document GetDocumentNonRelationalProperties(Document document, DocumentClient client)
        {
            Microsoft.Azure.Documents.Document AzureDocument =
client.CreateDocumentQuery(UriFactory.CreateDocumentCollectionUri(Properties.Resources.DataBaseId,
Properties.Resources.DocumentsCollectionId)).AsEnumerable().Where(d => d.GetPropertyValue<int>("DocumentId") ==
document.DocumentId).FirstOrDefault();
            document.Annotation = AzureDocument.GetPropertyValue<string>("Annotation");
            document.FlatText = AzureDocument.GetPropertyValue<string>("FlatText");
            document.HTMLstring = AzureDocument.GetPropertyValue<string>("HTMLstring");
            return document;
        }

        public int[] FindTextSimple(string Text, int corpusId)
        {
            int[] document = null;
            var documents = GetDocuments(corpusId);
            Text = GetTokens(Text);
            foreach (var item in Text.Split(' '))
            {
                List<int> d = new List<int>();
                foreach (var ditem in documents)
                {
                    string str = GetTokens(ditem.FlatText);
                    if (str.Contains(" " + item + " "))
                        d.Add(ditem.DocumentId);
                }
                if (document == null)
                {
                    document = d.ToArray();
                }
                else
                    document.Intersect(d.ToArray());
            }
            return document;
        }
        public int[] FindText(string Text, int corpusId)
        {
            //Последовательность вхождения не смотриться, просто наличие ключевых слов в тексте
            Text = GetTokens(Text);
            int[] documentIds = null;
            NLAStructureHelper nlaHelper = new NLAStructureHelper(GetDocumentClient(), corpusId, false);
            if (nlaHelper == null)
                return null;
            foreach (var item in Text.Split(' '))
            {
                if (documentIds == null)
                    documentIds = nlaHelper.FindWordDocumentIds(item).ToArray();
```

```
                else
                    documentIds.Intersect(nlaHelper.FindWordDocumentIds(item));
            }

            return documentIds;
            //Document[] documents = new Document[documentIds.Count()];
            //for (int i = 0; i < documentIds.Count(); i++)
            //{
            //    documents[i] = GetDocument(documentIds[i]);
            //}
            //return documents;
        }

        public Corpus[] GetCorpora(int UserId)
        {
            using (var context = new PaperCatModelContainer())
            {
                List<CorpusModel> corpusModels;
                if (UserId != 0)
                    corpusModels = context.CorpusModelSet.Where(c => c.User.UserId == UserId).ToList();
                else
                    corpusModels = context.CorpusModelSet.Where(c => c.User == null).ToList();
                Corpus[] corpuses = new Corpus[corpusModels.Count()];
                for (int i = 0; i < corpusModels.Count(); i++)
                {
                    corpuses[i] = corpusModels[i].GetCorpusFromModel();
                }
                return corpuses;
            }
        }

        public Document GetDocument(int DocumentId)
        {
            DocumentModel doc;
            using (var context = new PaperCatModelContainer())
            {
                doc = context.DocumentModelSet.Find(DocumentId);
                if (doc == null)
                    return null;
                return GetFullDocument(doc);
            }
        }

        private static Document GetFullDocument(DocumentModel document)
        {
            return                          GetDocumentNonRelationalProperties(document.GetDocumentFromModel(),
GetDocumentClient());
        }

        public Document[] GetDocuments(int CorpusId)
        {
            using (var context = new PaperCatModelContainer())
            {
                var documentList = context.DocumentModelSet.Where(d => d.Corpus.CorpusId == CorpusId).ToList();
                Document[] documents = new Document[documentList.Count];
                for (int i = 0; i < documentList.Count; i++)
                {
                    documents[i] = GetFullDocument(documentList[i]);
                }
                return documents;
            }
        }
```

```
public int MakeCommonCorpus(string Name, Document[] documents)
{
   return MakeCorpus(Name, documents, 0);
}
public int MakeCorpus(string Name, Document[] documents, int UserId)
{
   using (var context = new PaperCatModelContainer())
   {
      CorpusModel corpusModel = new CorpusModel()
      {
         CorpusName = Name,
         User = context.UserModelSet.FirstOrDefault(u => u.UserId == UserId)
      };
      context.CorpusModelSet.Add(corpusModel);
      context.SaveChanges();
      if (documents != null)
         foreach (var item in documents)
         {
            item.CorpusId = corpusModel.CorpusId;
            AddDocument(item);
         }
      return corpusModel.CorpusId;
   }
}

public void UpdateCorpus(Corpus corpus)
{
   if (corpus == null)
      throw new Exception("Параметр не может быть null");
   using (var context = new PaperCatModelContainer())
   {
      var corpusModel = context.CorpusModelSet.Find(corpus.CorpusId);
      if (corpusModel == null)
      {
         throw new Exception("Такого корпуса не существует");
      }
      corpusModel.CorpusName = corpus.CorpusName;
      context.SaveChanges();
   }
}

public void UpdateDocument(Document document, bool IsTextChanged)
{
   if (document == null)
      throw new Exception("Параметр не может быть null");
   using (var context = new PaperCatModelContainer())
   {
      var documentModel = context.DocumentModelSet.Find(document.DocumentId);
      if (documentModel == null)
      {
         throw new Exception("Такого документа не существует");
      }
      documentModel.Name = document.Name;
      documentModel.OriginalFile = document.OriginalFile;
      document.CorpusId = documentModel.Corpus.CorpusId; //документ остается в том же корпусе
      context.SaveChanges();
   }
   DocumentClient client = GetDocumentClient();
   string selfLink = GetDocumentSelfLink(document.DocumentId, client);

   if (IsTextChanged)
   {
```

```
                Task.Run(() => { DeleteDocumentFromIndexStructure(document.DocumentId, document.CorpusId);
AddDocumentToIndexStructure(document); } );

            }
            if (selfLink != null)
                client.ReplaceDocumentAsync(selfLink, new AzureDocument(document)).Wait();
        }

        public void UpdateUser(User user)
        {
            if (user == null)
                throw new Exception("Параметр не может быть null");
            using (var context = new PaperCatModelContainer())
            {
                var userModel = context.UserModelSet.Find(user.UserId);
                if (userModel == null)
                {
                    throw new Exception("Такого пользователя не существует");
                }
                userModel.FIO = user.FIO;
                if (IsThisLoginExist(user.Login, context, user.UserId))
                {
                    throw new Exception("Другой пользователь с таким логином уже существует");
                }
                userModel.Login = user.Login;
                userModel.Password = user.Password;
                userModel.Email = user.Email;
                context.SaveChanges();
            }
        }
    }
}
```

```
{
    public class NLAStructureHelper
    {
        private const int nSymbols = 3;
        string HeadSelfLink = "";
        NLAStructure Head;
        DocumentClient documentClient;

        /// <summary>
        /// Поиск осуществляется отдельно для каждого корпуса
        /// </summary>
        /// <param name="client"></param>
        /// <param name="Corpusid"></param>
        public NLAStructureHelper(DocumentClient client, int Corpusid, bool isForceCreate =
true)
        {
            documentClient = client;
            Head = LoadHead(Corpusid, isForceCreate);
        }
        /// <summary>
        /// Добавить id документа к данному слову
        /// </summary>
        /// <param name="word"></param>
        /// <param name="documentId"></param>
        public void Add(string word, int documentId)
        {
            NLAStructure structure = Head;
            string CurrentStructureLink = HeadSelfLink;
            for (int i = 0; i < word.Length; i += nSymbols)
            {
```

```
                string key = word.Length < i + nSymbols ? word.Substring(i) :
word.Substring(i, nSymbols);
                IndexStructureObject obj;
                if (structure.Dictionary.ContainsKey(key))
                    obj = structure.Dictionary[key];
                else
                {
                    obj = new IndexStructureObject();
                    structure.Dictionary.Add(key, obj);
                }

                if (i + nSymbols >= word.Length) //дошли до нужного листа
                {
                    obj.DocumentIds.Add(documentId);
                    SaveChanges(CurrentStructureLink, structure);
                }
                else
                {
                    NLAStructure oldNLAStructure = structure;
                    structure = GetNextStructure(obj);
                    SaveChanges(CurrentStructureLink, oldNLAStructure);
                    CurrentStructureLink = obj.NextStructureSelfLink;
                }
            }
        }
        /// <summary>
        //// Удалить все вхождения id документа из структуры
        /// </summary>
        /// <param name="documentId"></param>
        public void Delete(int documentId)
        {
            Delete(documentId, Head, HeadSelfLink);
        }

        private bool Delete(int documentId, NLAStructure NLAStructure, string
CurrentStructureLink)
        {
            if (NLAStructure != null)
            {
                var TemtDictionary = new Dictionary<string,
IndexStructureObject>(NLAStructure.Dictionary);
                foreach (var item in TemtDictionary)
                {
                    IndexStructureObject iso = item.Value;
                    NLAStructure nextNLAStructure = GetNextStructure(iso, false);
                    if (nextNLAStructure != null)
                        if (Delete(documentId, nextNLAStructure, iso.NextStructureSelfLink))
                            iso.NextStructureSelfLink = "";
                    iso.DocumentIds.Remove(documentId);
                    if (iso.NextStructureSelfLink == "" && iso.DocumentIds.Count == 0)
                    {
                        NLAStructure.Dictionary.Remove(item.Key);
                    }
                }
            }
            if (NLAStructure.Dictionary.Count == 0)
            {
                documentClient.DeleteDocumentAsync(CurrentStructureLink);
                return true;
            }
            SaveChanges(CurrentStructureLink, NLAStructure);
            return false;
        }
```

```
        private NLAStructure GetNextStructure(IndexStructureObject indexStructureObject, bool
isForcedCreate = true)
        {
            if (indexStructureObject.NextStructureSelfLink != "")
            {
                Document doc =
documentClient.ReadDocumentAsync(indexStructureObject.NextStructureSelfLink).Result;
                NLAStructure Structure = (dynamic)doc;
                return Structure;
            }
            else if (isForcedCreate)
            {
                NLAStructure newStructure = new NLAStructure();
                Document doc =
documentClient.CreateDocumentAsync(UriFactory.CreateDocumentCollectionUri(Properties.Resources
.DataBaseId, Properties.Resources.IndicesCollectionId), newStructure).Result;
                indexStructureObject.NextStructureSelfLink = doc.SelfLink;
                return newStructure;
            }
            else return null;
        }
        private void SaveChanges(string StructureLink, NLAStructure structure)
        {
            documentClient.ReplaceDocumentAsync(StructureLink, structure).Wait();
        }
        private NLAStructure LoadHead(int CorpusId, bool isForceCreate = true)
        {
            Uri documentCollectionUri =
UriFactory.CreateDocumentCollectionUri(Properties.Resources.DataBaseId,
Properties.Resources.IndicesCollectionId);
            Microsoft.Azure.Documents.Document doc = CorpusId != 0 ?

documentClient.CreateDocumentQuery(documentCollectionUri).AsEnumerable().Where(d =>
d.GetPropertyValue<int>("CorpusId") == CorpusId).FirstOrDefault()
                : null ;
            NLAHeadDocument Structure = null;

            if (doc != null)
            {
                Structure = (dynamic)doc;
                HeadSelfLink = doc.SelfLink;
            }
            else if (isForceCreate)
            {
                Structure = new NLAHeadDocument(CorpusId);
                doc = documentClient.CreateDocumentAsync(documentCollectionUri,
Structure).Result;
                HeadSelfLink = doc.SelfLink;
            }
            return Structure;
        }

        internal List<int> FindWordDocumentIds(string word)
        {
            NLAStructure structure = Head;
            IndexStructureObject obj = null;
            for (int i = 0; i < word.Length; i += nSymbols)
            {
                string key = word.Length < i + nSymbols ? word.Substring(i) :
word.Substring(i, nSymbols);
                if (structure != null && structure.Dictionary.ContainsKey(key))
                {
                    obj = structure.Dictionary[key];
                    structure = GetNextStructure(obj, false);
```

```
            }
            else return new List<int>();
        }
        if (obj != null)
            return obj.DocumentIds;
        return new List<int>();
    }
}
   }
```