ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ИМ. Н. П. ОГАРЁВА»**

Институт электроники и светотехники

Кафедра автоматизированных систем обработки информации и управления

УТВЕРЖДАЮ

Зав. кафедрой

к. т. н., проф.

\_\_\_\_\_\_\_\_\_С. А. Федосин

(подпись)

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2020 г.

**МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ**

**РАЗРАБОТКА ТЕМАТИЧЕСКОГО АГРЕГАТОРА ДАННЫХ С ПРИМЕНЕНИЕМ АЛГОРИТМОВ АНАЛИЗА ЕСТЕСТВЕННОГО ЯЗЫКА**

Автор магистерской диссертации (подпись) (дата) А. А. Тимофеев

Обозначение магистерской диссертации МД–02069964–09.04.01–08–20

Направление 09.04.01 Информатика и вычислительная техника

Руководитель работы

канд. техн. наук, доцент (подпись) (дата) С. А. Ямашкин

Нормоконтролер

канд. техн. наук, доцент (подпись) (дата) А. А. Аббакумов

Рецензент

канд. техн. наук, доцент (подпись) (дата) С. Д. Шибайкин

Саранск

2020

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**МОРДОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ИМ. Н. П. ОГАРЁВА»**

Институт электроники и светотехники

Кафедра автоматизированных систем обработки информации и управления

УТВЕРЖДАЮ

Зав. кафедрой

к. т. н., проф.

\_\_\_\_\_\_\_\_\_С. А. Федосин

(подпись)

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2020 г.

**ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ**

(в форме магистерской диссертации)

Студент Тимофеев Александр Александрович

1 Тема «Разработка тематического агрегатора данных с использованием алгоритмов анализа естественного языка»

Утверждена по МордГу № 41815 от 07.11.2018

2 Срок представления работы к защите 26.06.2020

3 Исходные данные для научного исследования (проектирования)

Библиография, отражающая современное состояние исследований

4 Содержание выпускной квалификационной работы

4.1 Анализ предметной области

4.2 Контекстно-свободные грамматики

4.3 Проектирование приложения

4.4 Разработка тематического агрегатора данных

4.5 Апробация разработанного тематического агрегатора данных

5 Приложения программный код, графические материалы

|  |  |  |
| --- | --- | --- |
| Руководитель работы | \_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | С. А. Ямашкин |
| Задание принял к исполнению | \_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | А. А. Тимофеев |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *1* |  |  |  |  |  |
| *2* |  |  | *Документация текстовая* |  |  |
| *3* |  |  |  |  |  |
| *4* | *А4* | *МД-02069964-090401-08-20* | *Пояснительная записка* | *83* |  |
| *5* |  |  |  |  |  |
| *6* |  |  | *Документация графическая* |  |  |
| *7* |  |  |  |  |  |
| *8* | *А1* | *МД-02069964-090401-08-20* | *Диаграмма вариантов* | *1* |  |
| *9* |  |  | *использования* |  |  |
| *10* | *А1* | *МД-02069964-090401-08-20* | *Диаграмма последовательности* | *1* |  |
| *11* | *А1* | *МД-02069964-090401-08-20* | *Диаграмма классов* | *1* |  |
| *12* | *А1* | *МД-02069964-090401-08-20* | *ER-диаграмма* | *1* |  |
| *13* |  |  |  |  |  |
| *14* |  |  | *Документация прочая* |  |  |
| *15* |  |  |  |  |  |
| *16* | *А4* | *МД-02069964-090401-01-20* | *Частичный программный код* | *44* | *Прил. А* |
| *17* |  |  | *разработанных модулей* |  |  |
| *18* | *А4* | *МД-02069964-090401-01-20* | *Графический материал* | *4* | *Прил. Б* |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *№ строки* | *Формат* | *Обозначение* | *Наименование* | *Кол.*  *листов* | *Примеча­ние* |

**РЕФЕРАТ**

Пояснительная записка содержит 83 листа, 19 рисунков, 2 таблицы, 28 использованных источников, 2 приложения.

АГРЕГАЦИЯ ДАННЫХ, DATA MINING, ЕСТЕСТВЕННЫЕ ЯЗЫКИ, ОБРАБОТКА ТЕКСТОВ, NLP, ИМЕНОВАННЫЕ СУЩНОСТИ, NER, ПАРСИНГ, ТОМИТА-ПАРСЕР.

Объектом разработки является программный комплекс – тематический агрегатор данных.

Цель работы – разработка программного комплекса, позволяющего автоматизировать сбор и структурирование информации на естественном языке с тематических интернет-ресурсов – тематического агрегатора данных.

В результате проделанной работы была анализирована предметная область и современное состояние исследований в области агрегации данных и извлечение именованных сущностей, разработан программный комплекс – тематический агрегатор данных, библиотека контекстно-свободных грамматик, спроектирован пользовательский интерфейс, а также проведена апробация системы.

Эффективность разработанной системы заключается в автоматизации сбора структурированной информации с тематических интернет-ресурсов, , обеспечения оптимальной организации процесса сбора данных, повышения удобства работы с информацией для пользователя.

Разработанный программный комплекс может быть применен для решения задачи автоматического извлечения и структурирования (агрегации) данных из документов на естественном языке.

[ВВЕДЕНИЕ 5](#_Toc41477892)

[1 Анализ предметной области 8](#_Toc41477893)

[1.1 Анализ опыта агрегации данных с тематических ресурсов](#_Toc41477894) 8

[1.2 Применение анализа естественных языков для агрегации данных 10](#_Toc41477895)

[1.3 Извлечение именованных сущностей 11](#_Toc41477896)

[1.4 Подходы к извлечению именованных сущностей 12](#_Toc41477896)

[1.4 Анализ существующих программных средств для извлечения именованных сущностей из текстов на естественном языке 14](#_Toc41477896)

[2 Контекстно-свободные грамматики 20](#_Toc41477897)

[2.1 Алгоритм GLR 20](#_Toc41477898)  
[2.2 Контекстно-свободные грамматики в инструменте «Томита-парсер» 22](#_Toc41477898)

[2.3 Синтаксис контекстно-свободных грамматик 24](#_Toc41477899)

[2.4 Интерпретация фактов с помощью контекстно-свободных грамматик 27](#_Toc41477900)

[3 Проектирование приложения 30](#_Toc41477901)

[3.1 Функциональные требования к системе 30](#_Toc41477902)

[3.2 Модель данных 32](#_Toc41477903)

[3.3 Инструменты реализации 40](#_Toc41477904)

[3.4 Поведение системы 43](#_Toc41477905)

[3.5 Разработка прототипа 49](#_Toc41477906)

[4 Разработка тематического агрегатора данных 57](#_Toc41477907)

[4.1 Модуль работы с источниками 57](#_Toc41477908)

[4.2 Модуль работы с базой данных 57](#_Toc41477909)

[4.3 Модуль извлечения именованных сущностей 59](#_Toc41477910)

[4.4 Модуль работы с КС-грамматиками 60](#_Toc41477911)

[4.5 Модуль обработки действий пользователя. 64](#_Toc41477912)

[5 Апробация разработанного тематического агрегатора данных 66](#_Toc41477915)

[5.1 Постановка задачи и подготовка к апробации агрегатора данных 66](#_Toc41477916)

[5.2 Тестирование тематического агрегатора данных 68](#_Toc41477917)

[5.3 Результаты тестирования тематического агрегатора данных 72](#_Toc41477918)

[ЗАКЛЮЧЕНИЕ 74](#_Toc41477919)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 76](#_Toc41477920)

[ПРИЛОЖЕНИЕ А (Обязательное) 80](#_Toc41477921)

[ПРИЛОЖЕНИЕ Б (Обязательное) 126](#_Toc41477921)

# **ВВЕДЕНИЕ**

***Актуальность.*** В настоящее время объемы информации, доступные человеку, многократно выросли благодаря развитию сети Интернет.

Крупные объёмы информации и быстрые темпы их прироста создают необходимость автоматического извлечения и структурирования (то есть агрегации) данных из документов на естественном языке, так как поиск нужной информации среди большого объема данных на множестве интернет- ресурсов становится все более сложной и затратной по времени задачей для человека. Информацию необходимо собирать, извлекать ее из множества неструктурированных источников близкой тематики, обрабатывать, структурировать, стремиться к повышению удобства и эффективности ее восприятия.

Решение этих задач возможно с применением обработки естественных языков, реализацией ее алгоритмов. Одна из наиболее важных задач обработки естественных языков, связанная с агрегацией данных и необходимая для ее реализации – это извлечение именованных сущностей. Извлечение именованных сущностей – один из важнейших аспектов извлечения структурированных данных из неструктурированных документов.

***Цель магистерской диссертации*** – разработка программного комплекса, позволяющего автоматизировать сбор и структурирование информации на естественном языке с тематических интернет-ресурсов – тематического агрегатора данных.

Для достижения поставленной цели необходимо решить ***следующие задачи***:

1. Проанализировать литературу и современное состояние исследований в области агрегации данных и анализа естественных языков.
2. Провести анализ существующих научных и практических решений в выбранной области, изучить методы, принципы и технологии извлечения именованных сущностей, возможности их применения для агрегации данных.
3. Спроектировать компонентную реализацию программной системы для агрегации тематических данных.
4. Разработать программный комплекс – тематический агрегатор данных.
5. Апробировать разработанный программный комплекс на практике, сделать вывод об эффективности созданной системы агрегации данных.

Результатом научно-конструкторских работ стали контекстно-свободные грамматики, реализующие задачи извлечения именованных сущностей (фактов и связанных с ними атрибутов) из текстов на естественном языке для различных тематик..

***Решение поставленных задач определило структуру выпускной квалификационной работы.*** В первой части проведено исследование ключевых аспектов применения автоматизированного анализа текстов; приводится обзор методов и алгоритмов извлечения информации из текстов на естественном языке; во второй – описан процесс разработки эффективных средств выделения фактов из текстов; в третьей главе представлена архитектура системы агрегации данных из текстов на естественном языке с применением контекстно-свободных грамматик.

***Объект исследования*** – агрегация тематических данных

***Предмет исследования*** – методы и алгоритмы обработки естественных языков и извлечения именованных сущностей, применимые для агрегации данных.

***Методология и методы исследования.*** При выполнении магистерской диссертационной работы использованы методы КС-грамматики, объектно-ориентированное программирование, UI/UX-проектирование, системный анализ.

***Научная новизна и теоретическая значимость работы.*** Новизна данной магистерской диссертационной работы заключается в том, что в ней:

а) Спроектирована новая компонентная реализация программной системы для агрегации тематических данных, *отличающаяся* наличием взаимодействующих модулей парсинга текста на естественном языке и работы с базой данных.

б) Разработан программный комплекс, реализующий агрегацию данных из текстов на естественном языке на основе извлечения именованных сущностей с помощью контекстно-свободных грамматик.

в) Проведено тестирование программного комплекса «Тематический агрегатор данных» на практических примерах.

***Практическая значимость.*** В ходе выполнения работы был создан программный комплекс – тематический агрегатор данных с применением алгоритмов анализа естественного языка, применимый для извлечения структурированной информации из текстов, была создана библиотека контекстно-свободных грамматик для извлечения разных типов фактов и работы с текстами разной тематики.

# **1 Анализ предметной области**

**1.1 Анализ опыта в области агрегации данных с тематических ресурсов**

Агрегация данных (извлечение информации) – это процесс получения структурированных данных из неструктурированных или слабоструктурированных источников.

Извлечение данных (data extraction) предваряет собой извлечение знаний (data mining) и является необходимым этапом предварительной обработки информации, крайне важным для итогового качества извлеченных данных. Даже такая простая задача, как извлечение текста из содержимого веб-страницы, сопряжена с конкретными сложностями – требуется, к примеру, отсечь «шапку» сайта, верхнее или вертикальное меню, навигационные и рекламные блоки, нижнюю часть сайта с указанием правообладателя, студии-разработчика, контактов, нижнее меню и прочие элементы дизайна, которые не содержат полезной информации.

В контексте агрегации данных с тематических интернет-ресурсов под неструктурированными источниками будем понимать веб-страницы, не имеющие значащей семантической машиночитаемой разметки, а под слабоструктурированными – логически взаимосвязанные фрагменты данных из неструктурированного источника (например, из таблиц), размещённые в пределах одной или нескольких веб-страниц.

Основные проблемы, возникающие при разработке тематических агрегаторов данных, описаны ниже:

– Разные принципы работы интернет-ресурсов с сервисами: некоторые из них предоставляют API (программные интерфейсы приложения, то есть наборы готовых [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [процедур](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [функций](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [структур](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и [констант](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D0%B0%D0%BD%D1%82%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) для работы со сторонними приложениями), а с другими необходимо работать посредством бота – имитации действий человека.

– Неоднородность ответов от сервисов.

– Вытекает из пункта 2 – неоднородность структур данных. [1]

Основные проблемы существующих агрегаторов данных прямо следуют из способов поиска, котоыре в них применяются. Разработка продвинутых поисковых машин, использующих, к примеру, семантический поиск, требует крупных затрат времени и труда, однако единственной альтернативой остается использование синтаксического анализа HTML-кода страниц, также называемого парсингом. Обычно парсер (модуль приложения, осуществляющий синтаксический анализ) использует очередь классифицированных лексем (каковыми являются, к примеру, регулярные выражения) для выделения требуемого контента на веб-странице по HTML-кодам, соответствующим нужным блокам на странице.[2,3]

Проблема извлечения текста и слабоструктурированных данных из веб-страниц усугубляется тем, что нарушение рекоментаций по верстке веб-страниц и смешение стандартов стали почти повсеместным явлением. Ситуация ещё более усугубляется совершенством веб-браузеров, корректно отображающих данные страницы, однако специализированные парсеры обычно не способны справиться с данной проблемой.

Эти проблемы не позволяют производить извлечение данных с помощью парсинга HTML-кода автоматически, применяя один алгоритм ко всем ресурсам. Для каждого нового сайта требуется определённая настройка, например, предварительн разметка оператором элементов веб-страницы в графическом пользовательском интерфейсе, или более сложные действия – составление регулярных выражений или запросов. В таком случае агрегация данных будет полуавтоматической. Для того, чтобы улучшить эффективность агрегации данных с тематических интернет-ресурсов, необходимо использовать автоматические методы извлечения данных. Наиболее перспективным вариантом видится использование методов анализа естественных языков. Это позволит частично обойти сложности, связанные с парсингом кода страниц на языке HTML, а также расширить возможности агрегации данных, получить возможность извлекать информацию не только со страниц интернет-ресурсов, но и в принципе из любых связных текстов необходимой тематики. [4]

Агрегация данных изучались многими отечественными специалистами. Так, вопросы агрегации данных рассмотрены в работах: Т.В. Петрусенко, А. Баргесяна, В. Аверченкова, Д. Евграфова, Е. Чиркина, М. Беззуба, Т. Родченко и др.

Различным аспектам обработки текстов на естественном языке посвящены исследования: А. В. Анисимова, А.В. Уланова, Е. И. Большаковой, В. В. Диковицкого, З.В. Дударя, М. А. Артемова, О. В. Золотарева, И. А. Тихомирова, П.А. Степанова, К. А. Найденова и др.

Основные проблемы извлечения именованных сущностей рассмотрены в работах: И. А. Виолентова, А. В. Рубайло, В. Ф. Хорошевского, С.Н. Судоплатовой, А.А. Петрова, Н. Э. Евремовой, Е.Б. Тутова, Р. В. Огородника и др.

**1.2 Применение анализа естественных языков для агрегации данных**

Естественный язык – одна из самых малоиспользуемых форм данных, доступных в наше время. Его анализ позволяет увеличить полезность приложений данных и сделать их неотъемлемой частью жизни. Современная жизнь основана на источниках данных из естественного языка, и приложения, основанные на его анализе, делают эти данные более доступными. [4]

Естественный язык представляет собой неструктурированные данные, которые используются людьми для общения между собой. Структурированные или полуструктурированные данные, в свою очередь, включают поля или разметку, позволяющие компьютеру анализировать их. Главной сложностью работы с естественным языком является то, что они определяются не правилами, а контекстом использования, который требуется реконструировать для компьютерной обработки. В отличие от формальных языков, которые всегда являются предметными, естественные языки намного более универсальны. Одни и те же слова могут использоваться для обозначения большого числа объектов, и из-за поддержания множества смыслов естественные языки избыточны. Избыточность естественных языков представляет серьезную проблему при машинной обработке, так как невозможно указать буквальный смысл для каждой ассоциации, а значит, каждый символ по умолчанию является неоднозначным. Для четкого определения смысла слов требуется больше вычислений, чем простой поиск в словаре. Избыточность, неоднозначность и ассоциации делают естественные языки динамичными, быстро развивающимися, способными отражать текущий опыт. Развитие естественных языков подразумевает включение новых определений, контекстов и приемов использования. Эти качества, делающие естественные языки удобным инструментом человеческого общения, затрудняют его анализ с применением детерминированных правил. Однако, несмотря на отсутствие машиночитаемой структуры, неструктурированные данные не являются случайными. Напротив, они подчиняются лингвистическим правилам, которые делают эти данные понятными для людей. [5,6]

Анализ естественных языков основывается на современной инфраструктуре анализа текстовой информации, коллекции приемов и методов, объединяющей инструменты для работы со строками, лексических ресурсах, копмьютерной лингвистике, алгоритмах машинного обучения и др. В настоящее время исследования в области анализа естественных языков ведутся в нескольких основных направлениях. Главным образом это:

– Компьютерная лингвистика и машинные переводы. Компьютерная лингвистика ставит перед собой задачи моделирования структуры естественных языков, формализации грамматики и синтаксиса для использования в области автоматизации переводов текстов на различных естественных языках, а также классификации и кластеризации текстовых данных.

– Извлечение информации из текстов на естественном языке. Исследования в этой области направлены на разработку методов выделения полезных данных из неструктурированных или слабоструктурированных текстов на естественном языке, а также автоматизации поиска элементов данных, удовлетворяющих определенным признакам (информационный поиск). Для этого используется семантический анализ, а также алгоритмы для поиска и выделения из текстов на естественном языке именованных сущностей (Named Entity) и связей между ними.

Извлечение информации и, в особенности, извлечение именованных сущностей относятся к той области анализа естественных языков, которая наилучшим образом подходит для решения задачи агрегации тематических данных. Теперь требуется рассмотреть существующие методы, подходы и существующие программные средства, предназначенные для извлечения именованных сущностей из текстов на естественном языке, и оценить их применимость для решения поставленных задач.

## **1.3 Извлечение именованных сущностей**

В современности большая часть доступной информации все еще представлена в виде текстов на естественном языке, так называемого plain text. Массивы информации, которые представляют собой тексты, могут быть использованы для того, чтобы получать семантическе структуры разного типа, такие, как субъект-объектные отношения, действия, признаки и т.п. Решение этих задач вручную требует большого количества временных и трудовых ресурсов, что является причиной возникновения самой задачи автоматизации извлечения данных из текстов. Одной из наиболее актуальных подзадач в данной области является извлечение именованных сущностей.

Задача извлечения именованных сущностей (Named Entity Recognition, NER) состоит в распознавании в тексте объектов, которые можно отнести к именованным сущностям (как правило, это слова и словосочетания) и их классификации по заранее определенным категориям. Данная задача является подзадачей задачи извлечения информации (Information extraction, IE), которая состоит в автоматическом извлечении структурированных данных из источников неструктурированной или слабоструктурированной информации, в частности, Интернета, и которая, в свою очередь, связана с информационным поиском и обработкой информации на естественном языке.

Как именованную сущность можно классифицировать слово или словосочетание, которое применяется для обозначения конкретного, вполне определенного объекта (предмета или явления), выделяющее этот объект или явление из ряда однотипных ему. Например, к именованным сущностям относятся имена собственные, названия организаций и т.п. Извлечение именованных сущностей может быть применено во многих областях, связанных с обработкой текстов на естественных языках и извлечением информации, таких, как, например, поиск информации, системы «вопрос-ответ», автоматизация сбора новостей и др. Это позволяет говорить о том, что алгоритмы извлечения именованных сущностей также могут быть применены и для решения задачи агрегации тематических данных.

## **1.4 Подходы к извлечению именованных сущностей**

Задача обработки текстов на естественном языке делится на три основных подзадачи, связанных с извлечением именованнх сущностей:

* обнаружение именованных сущностей;
* разрешение кореференции;
* снятие омонимии.

Кроме, собственно, задачи автоматизированного извлечения знаний из текста, разрешение проблемы неоднозначности применяется и в информационном поиске. Решение подзадачи снятия омонимии даст возможность классифицировать документы с большей точностью. Большая часть именованных сущностей будет встречаться крайне редко даже в достаточно объемном аннотированном корпусе текстов.

Для извлечения именованных сущностей применяются несколько типов признаков:

– признаки уровня слов (N-граммы, суффиксы, префиксы, части речи и т.д.);

– признаки уровня документа (наличие в корпусе текстов акронимов, положение термина в предложении, наличие термина в заголовке или тексте и т.д.);

– дополнительная информация (газеттиры, слова-указатели, списки стоп-слов, слов с капитализацией, не являющихся именованными сущностями, и т.д.).

В одном и том же тексте может встречаться несколько вхождений одного и того же термина, который может относиться к одной и той же сущности либо к различным объектам. В наиболее простом случае принято исходить из предположения, что в одном документе один и тот же термин относится к одной и той же сущности. Многие методы извлечения именованных сущностей исходят из предположения о том, что именованные сущности независимы от соседних слов, когда на самом же деле такая зависимость имеет место. Существуют и техники распознавания именованных сущностей, базирующиеся на предположении о том, что одинаковые термины будут иметь одинаковые метки во всем корпусе текстов, а не только в пределах одного текста. В случае работы с неразмеченными данными проводится кластеризация, а номер кластера используется как классифицирующий признак.[9-10]

Извлекать именованные сущности из текста можно следующими способами:

– используя онтологии (структуры, в которых описываются понятия и объекты, отношения между ними, их характеристики);

– используя машинное обучение;

– используя правила и шаблоны (контексты).

Первый способ позволяет обеспечить извлечение сущностей с высокой точностью и снятие омонимии. В число основных проблем этого метода входят сложность обновления и необходимость полноты онтологии, пропорциональной размеру тезауруса (поскольку извлекаются только те сущности, которые имеются в онтологии). [9,10]

Подход к извлечению именованных сущностей с помощью машинного обучения базируется на том, что в обработке естественного языка нужны столь же нечеткие и гибкие, как и сами естественные языки, вычислительные методы. В настоящее время для анализа текстов применяются статистические методы машинного обучения. В ходе практической реализации для работы с текстами на естественном языке выбирается семейство моделей, определюящих связи между целевыми и входными данными, задается форма, включающая параметры и особенности, а затем с помощью процедур оптимизации минимизируется ошибка модели на обучающих данных. Затем обученной модели можно передавать новые данные, на основе которых она будет строить прогнозы и возвращать метки, вероятности, признаки принадлежности и значения. Достоинством машинного обучения является то, что для использования нет необходимости составлять большое количество правил – в данном случае они будут иметь более общий вид. Машинное обучение также дает возможность извлекать правила автоматически. Недостатком является то, что для реализации метода машинного обучения необходим большой корпус текстов (набор данных, dataset), полностью и корректно размеченный. Без обучения на достаточно обширной выборке текстов метод машинного обучения не даст удовлетворительных результатов. [11,12,13]

Вышеперечисленные методы извлечения именованных сущностей используют метод анализа текста как так называемого «мешка слов». Такой подход возволяет исследовать отношения между текстами, содержащими одинаковые наборы из отдельных слов. Однако, он не учитывает контекст появления слов, который играет важную роль в передаче смысла. Это не значит, что такие методы полностью неэффективны, однако эффективность анализа текста может быть значительно улучшена, если использовать извлечение контекстных признаков.

В методе, основанном на контекстах, используются синтаксический и морфологический анализ. Цель синтаксического анализа – найти значимые связи между словами, обычно путем деления предложения на части, или связи между лексемами в древовидной структуре. Синтаксис является обязательной основой для работы с семантикой, так как это критически важный инструмент для понимания того, как слова влияют друг на друга при формировании фраз. Морфология же в анализе текста определяет форму отдельных слов или лексем. Структура слов может помощь выявить множественное число, род, время, спряжение и т.д. Анализ морфологии – сложная задача, так как в большинстве естественных языков имеется большое количество исключений из правил и специальных случаев. Главная задача морфологического анализа – выявить части слов, по которым эти слова можно отнести к тем или иным классам. Эти части слов можно условно назвать «тэгами частей речи». Затем полученные части речи используются для создания более крупных структур, таких, как фрагменты или фразы, или древа слов, которые затем можно использовать для построения структур данных семантического анализа.

Одним из наиболее эффективных способов извлечения именованных сущностей из текстов на естественном языке является внедрение грамматик и парсеров для создания облегченных синтаксических структур, непосредственно затрагивающих динамические коллекции текста, которые могут иметь большое значение. Грамматика представляет собой набор правил, описывающих, как синтаксические единицы естественного языка должны делиться на составляющие их элементы. В свою очередь, контекстно-свободная грамматика – это набор правил объединения синтаксических компонентов в осмысленные строки. С помощью грамматик можно определить разнообразные правила для сборки фраз или фрагментов из частей речи. Еще одним преимуществом является то, что грамматики могут содержать слова, обозначающие связь между объектами. Это облегчает решение описанной выше проблемы определения связи именованной сущности с соседними словами. Минусом метода является то, что КС-грамматики должны составляться вручную. Это требует определенного объема ручного труда.  [7]

Для работы с грамматиками нужен механизм, выполняющий систематический поиск осмысленных синтаксических структур в корпусе текста. Механизм, играющий эту роль, называется парсером. Грамматики определяют критерий поиска «осмысленности» информации в контексте нашего языка, парсер же непосредственно выполняет поиск. Синтаксический парсер – это программный компонент, который преобразует предложения в дерево синтаксического анализа, состоящее из иерархических элементов, также называемых синтаксическими категориями.

Когда парсер встречает предложение, он проверяет соответствеи его структуры известной грамматике, и, если такое есть, выполняет парсинг предложения согласно правилам этой грамматики, производя дерево синтаксического анализа. Парсеры часто используются для выявления важных структур, таких, как субъект и объект действия в предложении или последовательности слов, которые должны быть сгруппированы в каждой синтаксической ситуации.

## **1.4 Анализ существующих программных средств для извлечения именованных сущностей из текстов на естественном языке**

В ходе изучения состояния исследований в области извлечения именованных сущностей мной были рассмотрены следующие программные средства:

* General Architecture for Text Engineering (GATE) – система, разработанная в Университете Шеффилда и предназначенная для обработки естественного языка. Система применима для решения задач выявления смыслового содержания текстов на естественном языке и его упорядочивания в структурированной форме с помощью аннотирования отдельных сегментом текста. GATE может использоваться для анализа кореферентности документов, семантической аннотации, работы с онтологиями, машинного обучения. Инструмент GATE, как и программные модули для него, написан на языке Java. Есть возможность обработки различных форматов текстовых документов: простой текстовый, HTML, DOC и др.. Для описания правил в системе GATE применяется язык Jape. [9]
* К минусам системы можно отнести архитектуру обработки естественных языков. На практике архитектура системы не подходит для обработки гибких языков наподобие русского, немецкого или латинского, так как GATE не имеет возможности обрабатывать морфологию.. К плюсам системы можно отнести свободу, которую предлагает язык правил Jape, работу с огромным количеством входных форматов, даже таких обычно «проблемных» для парсинга, как pdf и docx, кроссплатформенность за счет платформы Java, на которой система была разработана. Также важным преимуществом является открытость кода системы GATE.
* Томита-парсер – программный инструмент для извлечения именованных сущностей из текстов на естественном языке. Томита-парсер позволяет создавать и быстро прототипировать системы извлечения фактов с помощью контекстно-свободных грамматик. Инструмент хорошо подходит для для обработки неструктурированных или слабоструктурированных текстов на естественном языке с целью извлечения из них структурированных данных. В инструменте для разбора входных данных используется алгоритм GLR (Generalized Left-to-right). Для распознавания и извлечения необходимых конструкций из текстов используются правила, которые составляются на языке расширенных контекстно-свободных грамматик. [8]
* Pullenti SDK. Pullenti SDK – комплекc средств разработки от компании «ООО Семантик», предназначенный для работы с неструктурированными данными. Pullenti SDK применим для решения задач анализа текста и извлечения именованных сущностей из неструктурированных русскоязычных текстов. Выделение именованных сущностей основано на правилах. Однако для некоторых типов сущностей можно подгружать внешние словари (онтологии), содержащие описания существующих сущностей, тогда система при выделении именованных сущностей будет пытаться привязываться к внешним сущностям, описанным в онтологиях. Плюсом системы является то, что она бесплатна для некоммерческого использования. [10]
* FreeLing – мультиязычная библиотека для обработки текстов. В инструменте содержатся два модуля для извлечения именованных сущностей – Basic-модуль и Bio-модуль. Первый, Bio-модуль, основан на машинном обучении. Он обеспечивает довольно высокую точность извлечения (свыше 90%), однако работает медленнее, чем Basic-модуль. Инструмент используется как библиотека, написанная на C/C++. В функционал, предоставляемый инструментом, входит разметка текста (токенизация), выделение предложений, орфологический анализ и т.д. Для работы FreeLing требуется наличие нескольких установленных внешних библиотек. FreeLing поддерживает русский и английский языки. Поддерживаемые платформы: GNU/Linux. [11]
* Stanford NER. Инструмент для обработки естественных языков, основанный на машинном обучении. Для извлечения именованных сущностей в инструменте есть три встроенных класса, основанных на регулярных выражениях, и один класс, предназначенный для разметки токенов через модель. Инструмент разработан на языке Java. Существуют также API для работы с инструментом, написанные и на других языках программирования:  Python, Ruby, Perl, F#/ C#/ .NET. Поддерживает только английский язык. [12]
* ReVerb. Инструмент извлечения именованных сущностей, основанный на машинном обучении. На вход используемая в ReVerb нейросеть принимает простой текст, а результатом работы являются тройки вида: аргумент 1, связывающая фраза, аргумент 2. Инструмент разработан на языке Java. Приложение использует библиотеку анализа естественных языков OpenNLP для разметки текста по частям речи и выделения в нем именных групп. Поддерживает только английский язык. [13]

Таким образом, сравнительный анализ программных средств показывает, что наиболее подходящим средством для решения задачи агрегации тематических данных в рамках данной магистерской диссертации является  инструмент «Томита-парсер».  «Томита-парсер» использует алгоритм GLR-анализатора, что делает его эффективным инструментом распознавания и извлечения именованных сущностей из текстов и документов на естественном языке при помощи контекстно-свободных грамматик и газеттиров – словарей ключевых слов. На принятие решения большое влияние оказал тот факт, что Томита-парсер является проектом с открытым исходным кодом. Это дает широкие возможности для применения инструмента в дальнейших исследованиях и позволяет разрабатывать программные решения в области агрегации данных с использованием обработки текста на естественном языке, основанных на извлечении именуемых сущностей с применением алгоритма Томита-парсера.

# 2 **Контекстно-свободные грамматики**

В предыдущей главе были сделаны выводы о том, что распознавание именованных сущностей является областью анализа естественных языков, наилучшим образом подходящей для решения задач агрегации тематических данных. В свою очередь, для извлечения из текстов структурированных данных наилучшим образом подходит использование контекстно-свободных грамматик, то есть правил, описывающих синтаксическую структуру выделяемых из текста цепочек. Также сравнительный анализ программных средств показал, что наиболее подходящим средством для реализации агрегатора данных на основе контекстно-свободных грамматик является инструмент «Томита-парсер». Таким образом, структура следующей главы определяется необходимостью изучить алгоритм, лежащий в основе работы инструмента «Томита-парсер», принципы работы контекстно-свободных грамматик и их реализации в инструменте «Томита-парсер», а также предоставляемые ими возможности, применимые при создании тематического агрегатора данных.

## **2.1 Алгоритм GLR**

Инструмент «Томита-парсер» работает на основе алгоритма GLR. Алгоритм парсинга GLR (GLR расшифровывается как «Generalized LR», то есть «Обобщенный LR», где L означает «left-to-right», то есть «слева направо», а R означает "rightmost", то есть «наиболее правую» продукцию) является расширенным вариантом алгоритма LR-парсера, предназначенного для разбора текста по недетермирированным или неоднозначным грамматикам. Первая реализация алгоритма GLR была описана Масару Томитой в статье 1984 года и называлась также «параллельным парсером». Целью создания алгоритма было точное и эффективное распознавание текстов на естественном языке. Обычный алгоритм LR-парсинга не может разрешать неоднозначность и недетерминированность естественных языков, а алгоритм GLR может справиться с этой задачей.

Принцип работы алгоритма Generalized LR схож с принципом работы обычного алгоритма LR-парсинга за исключением того, что после получения конкретной грамматики в качестве входных данных, алгоритм GLR использует поиск в ширину для обработки всех возможных интерпретаций входной последовательности. Генератор GLR-парсера, как и генератор LR-парсера, конвертирует входную грамматику в таблицы парсера, однако таблицы LR-парсера допускают только одно изменение состояния (определенное исходным состоянием парсера и входным терминальным символом), таблицы GLR-парсера позволяют производить несколько результирующих состояний. Результатом такого подхода является допустимость конфликтов типов сдвиг/свертка и свертка/свертка .

Когда парсер встречается с конфликтом, стек парсера (магазинная память) разветвляется на два или больше параллельных стека, в которых состояние, соответствующее каждому возможному переходу, являтся верхним. Затем следующий входной символ читается и используется, чтобы определить следующие переходы на верхних состояниях каждой ветви стека. При этом опять может возникнуть необходимость разветвления стека. Если же для какого-либо верхнего состояния и входного символа в таблице парсера не существует ни одного перехода, то эта ветвь стека считается ошибочной и может быть отброшена.

Основой оптимизации алгоритма является возможность использования общих префиксов и суффиксов стека несколькими его ветвями. Это сокращает общий объём памяти, необходимый GLR-парсеру для разбора входной последовательности. В результате такой оптимизации возникают сложные структуры, из-за которых граф поиска становится больше похожим на направленный ациклический граф, чем на поисковое дерево.

Алгоритм GLR имеет такую же сложность в худшем случае, как алгоритм Кока-Янгера-Касами и алгоритм Эрли, но по сравнению с ними у алгоритма GLR-парсинга имеется два преимущества:

1. Время, необходимое для выполнения алгоритма, пропорционально степени неопределенности в исходной грамматике. При полностью детерминированной грамматике GLR-алгоритм работает за время *O*(*n*). (Для алгоритма Эрли и алгоритма Кока-Янгера-Касами это не так, хотя оригинальный алгоритм Эрли может быть модифицирован для получения такого же преимущества).
2. Алгоритм GLR можно назвать «оперативным» или работающим в реальном времени  — то есть, он считывает символы из входного буфера в определенном порядке и производит как можно больше работы по анализу, доступной по прочтению данной входной последовательности. [17]

**2.2 Контекстно-свободные грамматики в инструменте «Томита-парсер»**

Томита-парсер позволяет извлекать структурированные данные из неструктурированных или слабоструктурированных текстов на естественном языке. Томита-парсер может работать и применяя только словари ключевых слов, но наибольшей эффективности в выделении фактов из текста достигает при использовании написанныъ пользователем шаблонов – контекстно-свободных грамматик. При их применении Томита-парсер выделяет из текста факты или цепочки слов, разбитые на поля. [12,13]

# Грамматика - это множество правил, описывающих, как синтаксические единицы естественного языка должны делиться на составляющие их элементы. В свою очередь, контекстно-свободная грамматика – это набор правил объединения синтаксических компонентов в осмысленные строки. С помощью грамматик можно определить разнообразные правила для сборки фраз или фрагментов из частей речи. Контекстно-свободными грамматиками описывается синтаксическая структура выделяемых цепочек. Одна цепочка соответствует одному предложению в тексте. Терминалы грамматики перед запуском отображаются на слова (или словосочетания, о чем будет сказано ниже) предложения. Одиночному слову может соответствовать много терминальных символов. Таким образом, парсер получает последовательность множеств терминальных символов в качестве входных данных. Из цепочки парсер выделяет подцепочки, которые, в свою очередь, интерпретируются им в разбитые по полям факты. На выходе получаются цепочки слов, распознанные этой грамматикой. При выделении цепочек по умолчанию происходит их нормализация, т.е. они приводятся к начальной форме. [14,15,16]

## **2.3 Синтаксис контекстно-свободных грамматик**

Правила грамматики, которые выделяют подцепочки, выглядят так:

S -> Noun;

Это правило выделяет из цепочки все существительные. В данном правиле Noun является терминалом. Терминал может стоять только в правой части правила. В Томите к терминалам относятся названия частей речи (Noun, Verb, Adj), некоторые символы (Comma, Punct, Ampersand, PlusSign), а также морфологические леммы, выделяемые одинарными кавычками.

S – это нетерминал, который строится из терминалов и должен хотя бы один раз встретиться в левой части правила. Если нетерминал встречается только в левой части и никогда – в правой, значит, это вершина грамматики. Ее также можно указать явно при помощи директивы #GRAMMAR\_ROOT.

Чтобы извлекать из текста не только отдельные существительные, но и, например, именные группы, применяется операция конкатенации. В качестве оператора конкатенации выступает обычный пробел:

S -> Adj Noun;

Файл грамматики состоит из двух частей: блока с директивами, влияющими на работу грамматики в целом, и множества правил.

Директивы начинаются со знака # и заканчиваются переносом строки. Некоторые директивы совпадают по смыслу с аналогичными директивами препроцессора языка C++.

* #encoding – Указывает кодировку данного файла с грамматикой. Кодировка по умолчанию: utf-8. Кодировка указывается в кавычках.
* #include – Включает в данную грамматику текст другой грамматики. При этом фильтры грамматик суммируются, а корень включаемой грамматики (см. #grammar\_root) игнорируется. Называние включаемой грамматики указывается в кавычках.
* #GRAMMAR\_ROOT указывает нетерминал, который является корнем данной грамматики. Корень грамматики можно не задавать явно, если в грамматике есть только один нетерминал, который ни разу не встречается в правой части правил грамматики.
* #GRAMMAR\_KWSET Директива #GRAMMAR\_KWSET позволяет явно указать имена или типы статей из газеттира, чьи найденные в предложении ключи должны передаваться парсеру в качестве терминалов. C точки зрения текущей грамматики, эти цепочки становятся multiword-ами. В частности, директива #GRAMMAR\_KWSET может использоваться для уточнения значения пометы kwtype=none. Синтаксис описания статей аналогичен синтаксису пометы kwset: перечисления в квадратных скобках [ ] через запятую.
* #filter. Фильтры позволяют ускорить работу грамматик. Если предложение на входе не соответствует ни одному из объявленных фильтров, то грамматика на них не запускается. Фильтр записывается как последовательность терминалов. Предложение проходит фильтр если в нем в указанном порядке встретились слова описываемые этими терминалами. Расстояние между словами может быть любым. Перед названиями терминалов в фильтрах ставится амперсанд &. У терминалов могут быть их обычные пометы: kwtype, wff и другие. Операторы (+, \* и другие) в фильтрах использовать нельзя. Между терминалами может быть указано максимально допустимое расстояние в квадратных скобках. Если в грамматику включается другая грамматика директивой #include, то фильтры включаемой грамматики тоже учитываются.
* Объявление подстановки: #define, #undef. Директива #define также используется для реализации подстановок. В этом случае у директивы два аргумента: имя подстановки и ее значение. Именем подстановки может быть любая последовательность из букв, цифр и символа подчеркивания, первый символ имени подстановки не может быть цифрой. Значением подстановки может быть любое завершенное выражение, в синтаксисе, разрешенном в правой части правил грамматики: список граммем, цепочка нетерминалов с ограничениями и интерпретациями и т.п. Знак переноса строки означает конец значения переменной. После объявления имя подстановки можно использовать с помощью синтаксиса в фигурных скобках со знаком доллара перед ним: ${ … }. Между знаком $, фигурными скобками и именем подстановки пробелы не допускаются.

Следует понимать, что подстановка «подставляется» не буквально (как, например, в языке C++), а сама по себе является сложным токеном в тексте грамматики. Поэтому подстановки нельзя использовать внутри строкового литерала (там она просто не будет распознана), а также внутри идентификаторов и других длинных токенов. Все определенные с помощью #define имена действуют дальше по тексту грамматики, в том числе во всех включенных позже файлах. Подстановку можно отменить, указав ее имя после директивы #undef. Рекомендуется всегда отменять подстановки явно, чтобы избежать неожиданных эффектов от включения одной грамматики в другую.

### #NO\_INTERPRETATION. Вводит запрет на интерпретацию в рамках текущей грамматики. Все операторы interp перестают срабатывать.[17-23]

В контекстно-свободных грамматиках Томита-парсера также используются синтаксические пометы. Терминал или нетерминал с пометой rt (root, корень) становится синтаксической вершиной поддерева, которое соответствует данному правилу. Это означает, что нетерминал в левой части правила получает грамматические и другие характеристики (не)терминала с пометой rt. В правой части правила не может находиться более одного (не)терминала с пометой rt. По умолчанию помета rt приписывается самому левому (не)терминалу в правой части правила. Сочетание пометы ${ } с именем подстановки фигурных скобках сообщает препроцессору, что требуется заменить имя подстановки на его значение.

На множество цепочек, которое описывает терминал или нетерминал можно наложить ряд ограничений. Для этого после (не)терминала в угловых скобках < > через запятую записываются пометы-ограничения, которые уточняют свойства (не)терминала. Список таких помет-ограничений приведен в таблице в конце этой главы.

Ограничения можно применять только к (не)терминалам стоящим в правой части правила и к (не)терминалам в фильтрах. Ограничение при нетерминале применяется к синтаксически главному слову группы, которую описывает данный нетерминал. К большей части помет можно применить оператор отрицания ~, который означает «допустимы любые значения ограничения, кроме перечисленных». Оператор отрицания ставится слева от пометы.

Ограничения могут быть разнообразными по своей структуре. Некоторые представляют собой унарный оператор, некоторые имеют поле, которое может быть заполнено разными значениями. Ограничение «согласование» определяется на паре (не)терминалов. У некоторых ограничений существуют свои особенности: некоторые из них нельзя применять к нетерминалам, некоторые не употребляются с отрицанием. Все эти свойства также описаны в сводной таблице. Ниже перечислены пометы, требующие подробного описания.

Помета kwtype (тип ключевого слова, keyword type) указывает, что слово (или главное слово многословной сущности), которому соответствует данных (не)терминал, должно быть объектом заданного типа. Значение поля kwtype заполняется названием статьи словаря или ее названием типа статей словаря. Информация о статьях и их типах находится в одном из gzt-словарей.

Каждая такая статья словаря является инструкцией для построения объекта. В свою очередь, объект может быть как ключевым словом из определенного словаря, так и синтаксической группой, выделенной другой грамматикой. Поле kwtype играет ключевую роль в построении правил грамматик, поскольку позволяет правилам использовать результаты работы других грамматик в качестве атомарных объектов.

У пометы kwtype также существует специальное значение none. Оно означает, что данный символ не может быть объектом соответствующим какой-либо статье. При проверке омонимов слова на соответствие помете kwtype=none, рассматриваются только те объекты, которые определены в правилах данной грамматики. Например, если некоторое слово — это название компании, но описывающая его статья «имя\_компании» не упоминается в правилах данной грамматики, то это слово будет считаться простой единицей и удовлетворит ограничению kwtype=none.

После срабатывания kwtype у символа остается только один из омонимов, для которого есть эта помета. Следовательно, выше в синтаксическом дереве применение к тому же символу еще одного ограничения kwtype с другим значением бессмысленно: ведь к данному символу приписан всего один омоним и правило, которое требует наличия другого омонима, просто не сработает. Этим помета kwtype отличается от пометы kwset.

Ограничение kwset (множество типов ключевых слов, keyword set) выполняет ту же функцию, что и kwtype. Разница между этими ограничениями состоит в том, что kwset применяется ко всем омонимам слова. Во-первых, это означает, что полученный после применения kwset символ может содержать другие омонимы, если есть другие статьи словаря которые также описывают этот символ. Во-вторых в помете kwset можно перечислить сразу несколько названий статей или их типов.

Запись kwset=["статья\_1","статья\_2",…,"статья\_n"] означает, что у слова должен существовать хотя бы один омоним, который описывается одной из перечисленных статей или типов. Запись с отрицанием kwset=~["статья\_1","статья\_2",…,"статья\_n"] означает, что если хотя бы один омоним слова описывается одной из перечисленных статей или типов, то правило не сработает.

Использование символов косая черта / звездочка \* в поле пометы kwset аналогично их использованию в поле пометы kwtype.

Помета gram ограничивает символ допустимыми значениями грамматических характеристик. Например, запись gram="nom,pl" означает, что у слова или у главного слова многословной сущности должны быть граммемы nom (nominative, именительный падеж) и pl (plural, множественное число). В списке граммем можно указывать любые граммемы, в том числе и часть речи, т.е. терминал NOUN можно записать как Word<gram="S">.

Если у слова есть два омонима, то помета gram последовательно проверяет значения грамматических характеристик у каждого из них. Например, у слова «леса» есть омоним с граммемами nom и pl (именительный падеж, множественное число) и омоним с граммемами gen и sg (родительный падеж, единственное число). Это слово будет соответствовать правилу с пометой gram="gen,sg" или с пометой gram="pl", но не будет соответствовать правилу с пометой gram="gen,pl".

Отрицание граммемы в поле пометы gram применяется сразу ко всем омонимам. В одной помете gram можно указывать одновременно граммемы с отрицанием и без.

Помета GU (grammar union) предоставляет более широкие возможности использования граммем в грамматиках. В своей самой простой форме GU=["nom,pl"] эта помета аналогична помете gram: она проверяет грамматические характеристики (в примере выше — «именительный падеж множественного числа») у каждого омонима и если находится омоним, удовлетворяющий этому условию, то правило срабатывает.

Граммемы записываются через запятую в квадратных скобках [ ]. Отрицание отдельных граммем в этой записи запрещено. Отрицание ~ перед квадратными скобками означает, что пересечение перечисленных граммем с граммемами каждого омонима пусто, т.е. если интерпретация хотя бы одного омонима подойдет по множество граммем в квадратных скобках, то правило с таким ограничением не сработает.

Для проверки совпадения значений одного или нескольких признаков одновременно у двух символов в Томите реализован механизм согласования. Согласование — это бинарное отношение, которое записывается при каждом из двух символов в правой части правила. Чтобы различить согласования одного типа внутри одного правила, паре символов приписывается идентификатор (целое число). Если проверка согласования прошла успешно и если один из участников согласования — вершина синтаксической группы, то ему приписывается вектор граммем, полученный пересечением граммем участников согласования.

Отрицание согласования означает, что у двух символов не может быть тождества значений заданных согласованием признаков. При этом часть значений признаков у этих символов может и совпадать.

Один символ может одновременно участвовать в нескольких отношениях согласования с разными участниками. Например, существительное может одновременно быть согласовано с определением и со сказуемым или с двумя прилагательными.

## **2.4 Интерпретация фактов с помощью контекстно-свободных грамматик**

Интерпретация – это способ сохранить результат анализа текста в табличной форме, удобной для последующей обработки, например, для сохранения в базу данных. Интерпретация является последним этапом обработки текста в Томита-парсере. Процесс интерпретации состоит в распределении подцепочек из выделенной грамматикой цепочки по полям факта. Факт – это множество значений полей-атрибутов, связанных смысловыми отношениями.

Интерпретация – это процедура, которая позволяет отобразить древесную синтаксическую структуру во множество линейно организованных фактов. Каждый факт можно представить как таблицу из одной строки и одной или нескольких колонок: в каждой ячейке этой таблицы содержится объект. Цель интерпретации состоит в сопоставлении множества строящихся нетерминалов грамматики множеству выделяемых для определенного типа факта объектов, т.е. в распределении подцепочек слов по заранее заданным полям в таблице.

КС-грамматики Томита-парсера могут интерпретировать факты. Для записи цепочки в поле факта интерпретации в Томите используется оператор interp. Цепочка, распознанная символом за которым следует interp, записывается в поле факта, которое обозначается в скобках после interp.

Любой факт, порождаемый грамматикой, должен быть описан. Факты описываются в словаре типа gzt.

Описание статьи словаря состоит из типа статьи, названия статьи и содержания статьи. Содержание статьи ограничивается фигурными скобками { }. Содержание статьи – это названия полей и их значения после знака равенства =. Каждое следующее поле отделяется переносом строки или фигурными скобками. Ключ — это основное поле статьи. В ключе указывается, как именно ищется цепочка. В одной статье может быть несколько ключей. Текст поля key записывается в кавычках. Слова разделяются пробелами. Регистр значения не имеет. Название статьи записывается в кавычках. Оно состоит из букв любого алфавита с учетом регистра, цифр и символов подчеркивание (\_) и косая черта (/). Цифры в начале названия статьи запрещены. Названия статей должны быть уникальными. Перед названием статьи обязательно указываться ее тип.

Название факта задается служебным словом message, после имени факта записывается его тип. Базовый тип всех фактов NFactType.TFact. Поля фактов перечисляются в фигурных скобках { }.

Типы фактов могут наследоваться. При этом все поля, входящие в базовый тип факта, войдут и в производный тип. Для того, чтобы от базового типа факта можно было наследовать производные, нужно указывать диапазон номеров полей, зарезервированных для использования в производных типах.

Все поля фактов имеют следующие характеристики: имя поля, тип поля, обязательность заполнения поля. Явное указание всех трех характеристик обязательно при описании типа факта.

Поля фактов относятся к одному из четырех типов:

– String – cтрока. Это универсальное поле, может содержать любые строковые значения.

– Bool – булевский тип данных, допускает одно из двух значений: true и false.

– NFactType.TFio – тип для хранения ФИО: полных фамилии, имени и отчества или инициалов.

– NFactType.TDate – тип для хранения даты и времени или временного интервала, выделенных статьей одного из двух специальных типов: date и date\_chain.

Обязательность заполнения поля факта задается служебным словом required или optional, затем следует название типа поля (см. таблицу выше), а затем его название и после знака равенства = номер поля факта и точка с запятой. Слова required или optional означают, что поле факта соответственно обязательное или факультативное. Если в процессе построения дерева у факта остается незаполненным хотя бы одно обязательное поле, то факт не извлекается.

Номера поля должны быть уникальными для каждого факта. Поэтому важно иметь в виду, что если факт унаследовал свой тип от другого факта, то он также унаследовал и его поля с их номерами.

При описании поля факта ему можно приписать значение по умолчанию. В этом случае если будет заполнено хотя бы одно другое поле этого факта, то в этом поле уже будет заданное по умолчанию значение. Значение по умолчанию изменится, если при интерпретации это поле будет заполнено другой цепочкой.

Значение по умолчанию задается в квадратных скобках [ ] после объявления поля, в параметре default.

В тех случаях, когда интерпретируемый символ представляет собой отдельный объект, построенный другой грамматикой, в которой были извлечены некоторые факты, эти факты можно получить в текущей грамматике. Оператор from позволяет явно указать при интерпретации из какого поля другого факта нужно извлечь данные для текущей интерпретации. Поля обоих фактов должны быть одинакового типа.

Для записи цепочки в поле факта интерпретации в Томите используется оператор interp. Цепочка, распознанная символом за которым следует interp, записывается в поле факта, которое обозначается в скобках после interp. Записываемое в поле факта значение должно соответствовать типу данных этого поля. Если в конце построения дерева у факта остались незаполненные поля типа required, то факт не будет сохранен в выходной файл.

В тех случаях, когда интерпретируемый символ представляет собой отдельный объект, построенный другой грамматикой, в которой были извлечены некоторые факты, эти факты можно получить в текущей грамматике. Оператор from позволяет явно указать при интерпретации из какого поля другого факта нужно извлечь данные для текущей интерпретации. Поля обоих фактов должны быть одинакового типа.

Название факта после оператора from можно опустить. В этом случае интерпретация будет искать любой факт с заданным полем.

Если имена полей обоих фактов совпадают, то оператор from можно не использовать. Интерпретация автоматически извлечет значение поля с таким же именем.

В случае, если КС-грамматика построит дерево, в котором поле одного факта будет интерпретировано несколько раз, интерпретация осуществляется по символам правой части правила пошагово: слева направо.

В случае, если цепочки не пересекаются и у этого же факта есть другое поле, которое интерпретируется в дереве один раз, то это второе поле дублируется, и на выходе порождаются два факта с разными цепочками в первом поле и одинаковыми во втором. Если же цепочки не пересекаются и у этого же факта нет поля, которое интерпретируется в дереве один раз, то факты интерпретируются попарно в порядке парсинга: слева направо.

При интерпретации осуществляется нормализация значений полей факта. Нормализация выделенных объектов реализует два принципа:

* Морфологическая нормализация. Выбирается синтаксическая вершина цепочки и ставится в словарную форму. Если в цепочке есть слова связанные с вершиной согласованием, то соответствующие граммемы также принимают новые значения, так чтобы согласование было соблюдено.
* Нормализация через gzt-словарь. Если при создании цепочки, которая попала в поле факта, участвовала статья словаря с заполненным полем lemma, то при нормализации часть цепочки описанной этой статьей заменяется значением поля lemma этой статьи. При этом к новой цепочке, взятой из поля lemma, применяются граммемы, которые применялись к исходной цепочке в процессе построения дерева.[24]

# **3 Проектирование веб-приложения**

## **3.1 Функциональные требования к системе**

Программный комплекс планируется использовать для агрегации (автоматизированного сбора и упорядочивания) данных на заданную тему с использованием алгоритмов анализа естественного языка.

Данное приложение использует один актор – пользователь системы. Его возможности представлены на Use-case диаграмме приложения (рисунок 3.1)

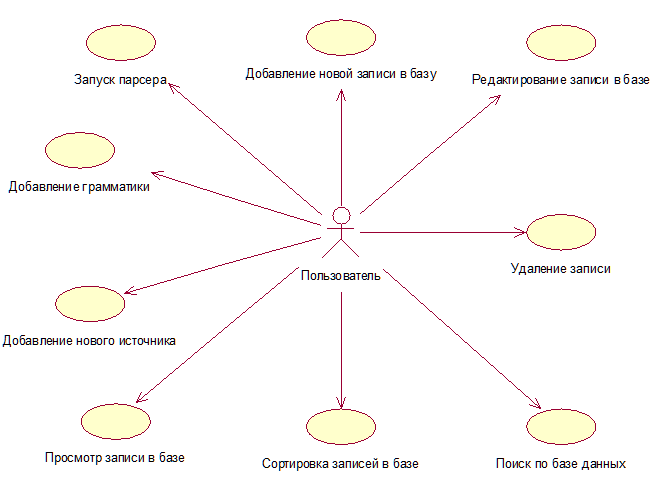


Рисунок 3.1 – Use-case диаграмма приложения

Подробное описание возможностей пользователя системы представлено в таблице реестра вариантов использования (таблица 1).

Таблица 1 – Реестр вариантов использования

|  |  |  |  |
| --- | --- | --- | --- |
| Код | Основной актор | Наименование | Формулировка |
| P1 | Пользователь | Запуск парсера | Функция, позволяющая запустить агрегатор данных для обновления базы |
| P2 | Пользователь | Добавление новой записи в базу | Функция, позволяющая добавить запись в базу данных в ручном режиме. |
| P3 | Пользователь | Редактирование записи в базе | Пользователь имеет возможность редактировать записи в базе, сформированные агрегатором. |
| P4 | Пользователь | Удаление записи из базы данных | Пользователь имеет возможность удалять записи из базы релизов. |
| P5 | Пользователь | Поиск по базе данных | Этот вариант определяет возможность пользователя искать информацию в базе данных. |
| P6 | Пользователь | Задание фильтра поиска для базы | Функция, позволяющая задавать условия для поиска в базе данных. |
| P7 | Пользователь | Сортировка записей в базе | Используется пользователем для сортировки записей по одному из полей. |
| P8 | Пользователь | Просмотр записи в базе | Пользователь имеет возможность просмотра содержимого базы. |
| P9 | Пользователь | Добавление грамматики | Пользователь имеет возможность добавить новую КС-грамматику для работы с текстами. |

Данная таблица описывает возможности действий, которые предоставляет система для пользователя.

## **3.2 Модель данных**

Все данные о созданных пользователями системы моделях хранятся в базе данных Firebase. Firebase – это облачный сервис, сочетающий в себе множество функций: аутентификацию, базу данных в реальном времени, хранение файлов, уведомления и прочие.

Поскольку агрегатор предназначен для сбора разнородных данных, развернутая для него база данных изначально не состоит из конкретных таблиц. Благодаря возможностям Firebase в процессе работы агрегатора при работе с новой тематикой в базе данных создается таблица для нее, после чего при извлечении и интерпретации новых видов фактов в базе данных создаются поля для фактов и их атрибутов.

Таким образом, модель данных можно представить в следующем виде:

База данных состоит из таблицы источников (Sources), а также таблиц фактов.

Таблица Sources содержит 6 полей:

1. *ID*– уникальный номер источника.
2. *Name* – название источника.
3. *URL* – URL-адрес источника.
4. *Fact\_types* – описание типов фактов, предназначенных к извлечению из данного источника.
5. *Dict*– корневой словарь (перечень всех используемых для работы с источником словарей и грамматик)
6. *Grammar*– контекстно-свободная грамматика, используемая для извлечения фактов.

Пример таблицы фактов состоит из полей:

1. *Fact* – извлеченный из текста факт.
2. *Attribute1, Attribute2*и т.д. – атрибуты извлеченного факта, которых может быть несколько.

Подобная структура БД вкупе с возможностями платформы Firebase позволяет легко динамически создавать таблицы и изменять поля таблиц во время процесса агрегации данных из текстов на естественном языке и обеспечивать необходимую легкость и доступность к данным БД при работе с сервером.

## **3.3 Инструменты реализации**

В качестве основного языка для разработки агрегатора данных был выбран Python, язык программирования общего назначения, поддерживающий объектно-ориентированное программирование. Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование.

Python – активно развивающийся язык программирования, новые версии с добавлением/изменением языковых свойств выходят примерно раз в два с половиной года.

В качестве основного языка для разработки веб-интерфейса был выбран JavaScript. JavaScript – объектно-ориентированный язык программирования для написания сценариев. Таким образом, посредством языка JavaScript реализуется возможность программирования на стороне клиента. Язык JavaScript предоставляет возможность доступа к элементам разметки web-страницы посредством объектов.

Вся полученная информация выводится в понятном пользователю виде на Web-страницах. Web-страницы описываются на специальном языке, называемом HTML (HyperText Markup Language язык разметки гипертекстовой информации), который позволяет осуществлять простое форматирование текста, вставку графики, а также составление таблиц и цветовое оформление документов.

В качестве серверной части приложения будет использоваться стек технологий (Linux + Nginx + Firebase)

 Для хранения всех данных проекта была выбрана база данных Firebase. Основной сервис – облачная СУБД класса NoSQL, позволяющая разработчикам приложений хранить и синхронизировать данные между несколькими клиентами. Поддержаны особенности интеграции с приложениями под операционные системы Android и iOS, что важно для реализации кроссплатформенности, реализовано API для приложений на JavaScript, Java, Objective-C и Node.js, что критически важно для работы с веб-интерфейсом приложения, а также возможно работать напрямую с базой данных в стиле REST из ряда JavaScript-фреймворков. Предусмотрено API для шифрования данных.

Развертывание системы предполагается на сервере. Серверы, в сущности, представляют собой компьютеры со специальными характеристиками, ориентированными на непрерывную работу, эффективность и надежность. Кроме того, серверы балансируют вычислительную мощность с потреблением электроэнергии. Поэтому операционные системы Linux для серверов на первое место ставят надежность и потребление ресурсов.

Вероятно, наиболее известной из операционных систем Linux является Ubuntu. Имея множество вариантов, Ubuntu является стабильным дистрибутивом. Используя Ubuntu Server, можно разворачивать web-серверы, предоставлять контейнеры и делать многое другое. Более того, эти инструменты готовы к использованию сразу после установки.

Именно Ubuntu 16.0 была использована в качестве ОС для разворачивания на ней сервера. Firebase служит базой данных, которая изменяется в реальном времени и хранит данные в JSON. Любые изменения в базе данных тут же синхронизируются между всеми клиентами, или девайсами, которые используют одну и ту же базу данных. Другими словами, обновление в Firebase происходят мгновенно.

Вместе с хранилищем, Firebase также предоставляет пользовательскую аутентификацию, и поэтому все данные передаются через защищенное соединение SSL.

Nginx – мощный инструмент для развертывания веб-сервера, который при правильной настройке превосходит Apache. Области применения Nginx весьма обширны – от кэширования HTTP до создания инвертированного прокси-сервера. Nginx работает на ОС Unix-типа и был успешно протестирован на OpenBSD, FreeBSD, Linux, Mac OS X, Solaris.

В отличие от обычного веб-сервера, Nginx не создаёт один поток под каждый запрос, а разделяет его на меньшие однотипные структуры, называемые рабочими соединениями. Каждое такое соединение обрабатывается отдельным рабочим процессом, а после выполнения они сливаются в единый блок, возвращающий результат в основной процесс обработки данных. Одно рабочее соединение может обрабатывать до 1024 запросов одного вида одновременно. Для конфигурации Nginx задействуется директория /etc/nginx/. При дальнейшей работе с сервером важны файл nginx.conf и папка sites-available.

Основные настройки можно в файле nginx.conf. Благодаря этому файлу, все параметры можно настроить по своему усмотрению. Работать можно и с настройками по умолчанию.

Root-каталог Nginx по умолчанию находится в директории /usr/share/nginx/html. Все файлы, которые размещаются в нем, автоматически обслуживаются веб-сервером. Место определяется файлом конфигурации, который можно найти в /etc/nginx/conf.d/default.conf.

Новые блоки server создаются через конфигурационные файлы в /etc/nginx/conf.d. Они буду загружаться при запуске Nginx, если заканчиваются на .conf.

Основной конфигурационный файл сервера находится в /etc/nginx/nginx.conf. Через него изменяются любые настройки.

**3.4 Поведение системы**

Алгоритм работы основного компонента системы, инструмента «Томита-парсер», следующий:

1. Парсер ищет в тексте вхождения всех ключей из газеттира. В том случае, если ключ состоит из нескольких слов, создается новое искусственное слово, условно называемое «мультиворд».
2. Из всех найденных в тексте вхождений ключей газеттира отбираются те, которые упоминаются в грамматике.
3. Среди отобранных ключей могут встретиться и мультиворды, пересекающиеся друг с другом или включающие в себя одиночные ключевые слова. В этом случае парсер попытается покрыть предложение непересекающимися ключевыми словами так, чтобы ими были охвачены как можно большие части предложения.
4. Линейная цепочка, состоящая из слов и мультивордов, подается на вход GLR-парсеру. Терминалы грамматики отображаются на входные слова и мультиворды.
5. На последовательности множеств терминалов GLR-парсер строит все возможные варианты. Из всех построенных вариантов также отбираются те, которые как можно шире покрывают предложение.
6. Затем парсер запускает процедуру интерпретации на построенном синтаксическом дереве. Он отбирает специально помеченные подузлы, а слова, которые им соответствуют, записываются в порождаемые грамматикой поля фактов.

### *Каскады*

В качестве ключевых слов могут быть использованы результаты работы других грамматик. Тогда на шаге 2 предыдущего алгоритма парсер смотрит, какие ключевые слова задаются не списком слов, а конкретными грамматиками. Если такие ключи есть, то он рекурсивно запускает тот же алгоритм на всех грамматиках, упомянутых в статьях для этих ключевых слов. Выделенные этими грамматиками цепочки становятся мультивордами и смешиваются с остальными ключевыми словами. Таким образом, можно строить целые деревья парсеров, подавая результаты работы одного в качестве входных символов другого.

### *Адресация*

Главным элементом для парсера является статья из газеттира. Статью можно представлять в виде функции на предложении, результатом работы которой всегда является выделенная подцепочка. Имя статьи тогда является именем этой функции, а тип статьи – множеством функций, применяемых в случайном порядке. Внутри этой статьи-функции явно указано, каким способом выделить нужную цепочку: грамматикой или поиском указанных слов и словосочетаний. Вся связь между грамматиками происходит через статьи газеттира, в которых в ключе вместо слов указана грамматика. В параметрах к парсеру указываются также имена газеттирных статей, т.е. вызываются функции, выделяющие подцепочки.

### *Веса*

Парсер может порождать много вариантов разбора одной и той же цепочки, которые отличаются только деревом разбора. Различные деревья могут порождать разные варианты заполнения полей фактов. Для этой цели правилу можно приписывать вес (помета weight). По умолчанию вес у всех правил равен 1. Его можно искусственно уменьшить, чтобы парсер выбрал вариант с большим весом, если такой построился.[25]

Работа агрегатора основана на работе инструмента «Томита-парсер» по вышеописанному алгоритму, основываясь на заданных пользователем через веб-интерфейс источниках и КС-грамматиках.

Диаграмма последовательности для общего алгоритма работы программы приведена на рисунке 3.2.

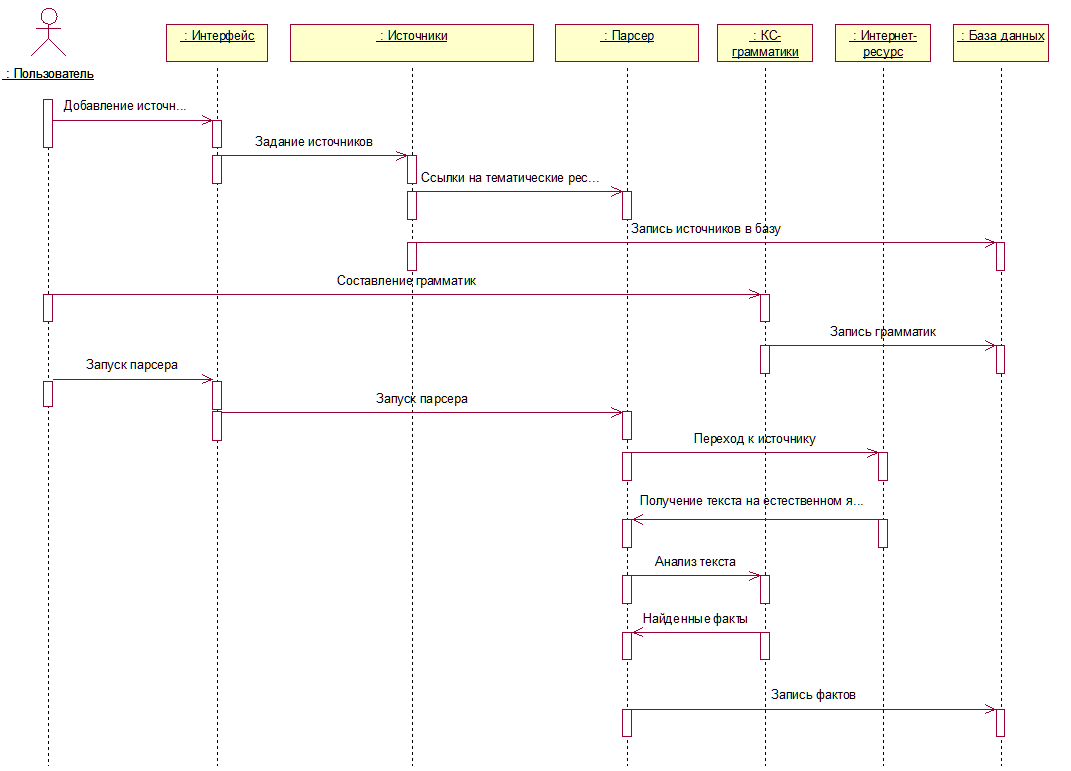


Рисунок 3.2 – Диаграмма последовательности

При работе с системой пользователь посредством веб-интерфейса добавляет источники для анализа и составляет контекстно-свободные грамматики, соответствующие тематике информации, которую предполагается из них извлекать. Информация об источниках и грамматики сохраняются в базу данных. После этого пользователь запускает работу парсера, который обрабатывает найденные в источниках тексты на естественном языке, основываясь на заданных пользователем КС-грамматиках.

Выделенные из текста факты и их атрибуты, а также результаты их интерпретации сохраняются в таблицы базы данных, соответствующие тематикам, после чего пользователь имеет возможность просматривать и редактировать их посредством веб-интерфейса.

Диаграмма классов приложения представлена на рисунке 3.3. Она демонстрирует общую структуру иерархии классов системы, обеспечивающих ее быстродействие и удобство работы пользователя.

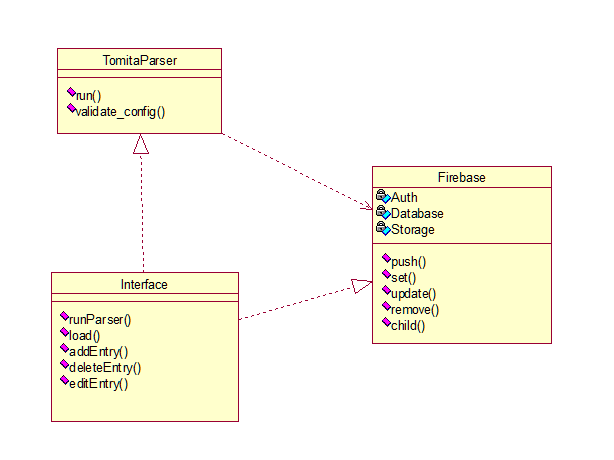


Рисунок 3.3 – Диаграмма классов

Система состоит из 9 основных классов и 3 вспомогательных (описаны в ER-диаграмме):

* *TomitaParser* – класс, описывающий главный модуль системы – модуль извлечения именованных сущностей, т.е. программную оболочку инструмента «Томита-парсер», и обеспечивающий работу с ним. Данный класс не содержит атрибутов.

Методы класса:

run() – метод, запускающий работу Томита-парсера и на выходе предоставляющий набор выделенных парсером фактов и атрибутов.

validate\_config() – вспомогательный метод, проверяющий правильность конфигурации парсера и файлов словарей и грамматик.

* *Firebase* – класс, описывающий модуль взаимодействия с базой данных Firebase.

Атрибуты класса:

Auth – ключ аутентификации для доступа к сервису Firebase.

Database – база данных Firebase, подключенная к системе.

Storage – облачное хранилище, в котором размещается используемая в приложении база данных.

Методы класса:

push() – метод, позволяющий пользователю добавить записи в базу данных.

set() – метод, позволяющий пользователю изменять записи в базе данных.

update() – метод, позволяющий пользователю обновить таблицы базы данных.

remove() – метод, позволяющий пользователю удалять записи из базы данных.

child() – метод, позволяющий порождать новые таблицы и подтаблицы в базе данных, что необходимо при работе с различными типами фактов и разными тематиками.

* *Interface* – класс, описывающий модуль обработки действий пользователя, также через него реализуются модули работы с источниками и КС-грамматиками. Фактически организует взаимодействие между пользователем и двумя вышеописанными классами. Данный класс не содержит атрибутов.

Методы класса:

runParser() – инициирует работу модуля извлечения именованных сущностей.

load() – загружает в интерфейс содержимое таблиц из базы данных, позволяя пользователю просмотреть их содержимое.

addEntry() - инициирует работу модуля взаимодействия с базой данных, позволяя добавлять записи в таблицы (фактов или источников)

editEntry() – инициирует работу модуля взаимодействия с базой данных, позволяя редактировать записи в таблицах (фактов или источников)

deleteEntry() – инициирует работу модуля взаимодействия с базой данных, позволяя удалять записи из таблиц (фактов или источников)

**3.5 Разработка прототипа**

Обычно под прототипом понимают поведенческую модель, в которой не реализуются все слои архитектуры системы, но которая обладает предполагаемым интерфейсом пользователя. Такой прототип называется горизонтальным, или поведенческим.

Горизонтальный прототип не реализует функциональности системы в действительности, он создает только ее видимость, поэтому трудоемкость разработки горизонтального прототипа невелика. Экраны интерфейса пользователя, навигация между ними показывают функциональные возможности и структуру доступа к информации, что позволяет пользователю исследовать поведение системы в различных ситуациях, выяснить возможность выполнения необходимой работы и уточнить требования.

Вертикальный, или структурный прототип служит для проверки концепции, реализует часть функциональности системы на всех уровнях от интерфейса пользователя до сервисных функций. Такой прототип действует как настоящая система и позволяет оптимизировать алгоритмы, оценить базу данных, определить критические временные требования. Вертикальный прототип полезен для исследования критически важных требований к интерфейсу и времени исполнения, для сокращения рисков на этапе проектирования системы.

Точкой входа «А» в приложение является начальный экран, который пользователь видит при загрузке веб-приложения. Дизайн этого экрана представлен на рисунке 3.4.

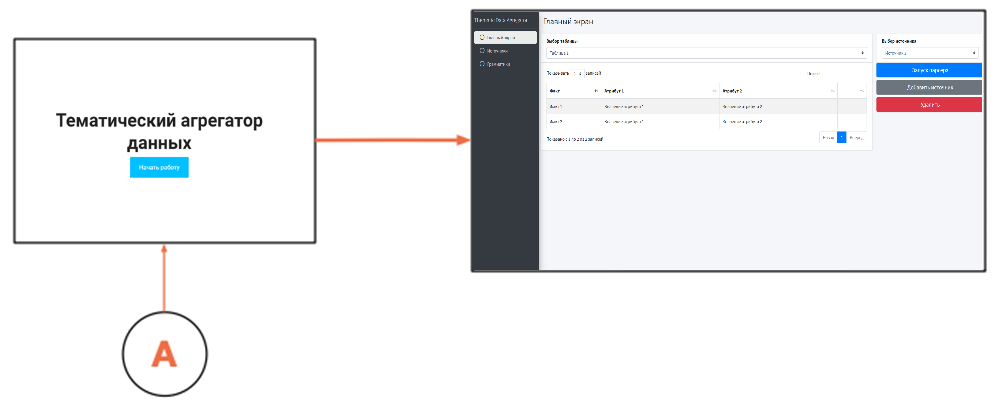


Рисунок 3.4 – Точка входа в приложение

Начальная страница приложения состоит из двух элементов – его названия и кнопки «Начать работу», которая призывает пользователя начать работу с тематическим агрегатором данных (рисунок 3.5).

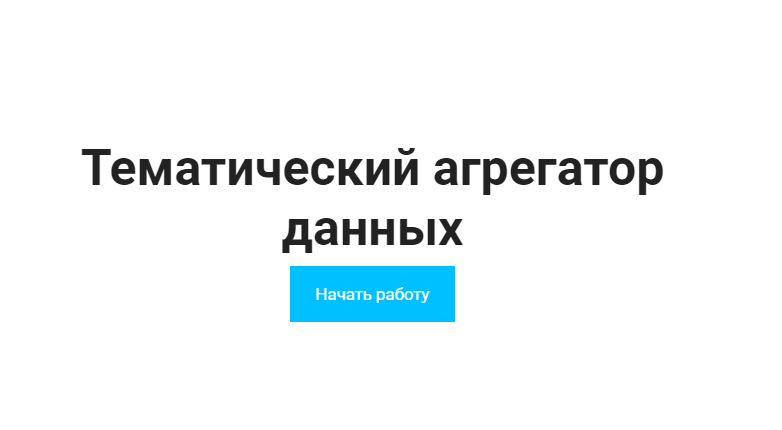


Рисунок 3.5 – Дизайн начального экрана приложения

После нажатия на кнопку пользователь попадает на главный экран приложения. Здесь пользователь может просматривать таблицы базы данных, содержащие факты и атрибуты, извлеченные из уже обработанных парсером источников, проводить поиск по и сортировку по таблице. Для запуска поиска по таблице нужно ввести запрос в поле поиска. Поиск по загруженной из базы данных таблице произойдет сразу же в реальном времени, и результаты будут выведены пользователю в таблице. Содержимое таблицы сортируется по возрастанию или убыванию значения одного из столбцов таблицы путем нажатия на заголовок столбца.

В панели справа пользователь выбирать источник для агрегации в выпадающем списке, удалять записи из таблицы, добавлять новые источники, а также запускать работу парсера по выбранному источнику для обновления базы данных с помощью соответствующих кнопок. С помощью навигационной панели слева пользователь может переключаться между экранами приложения.

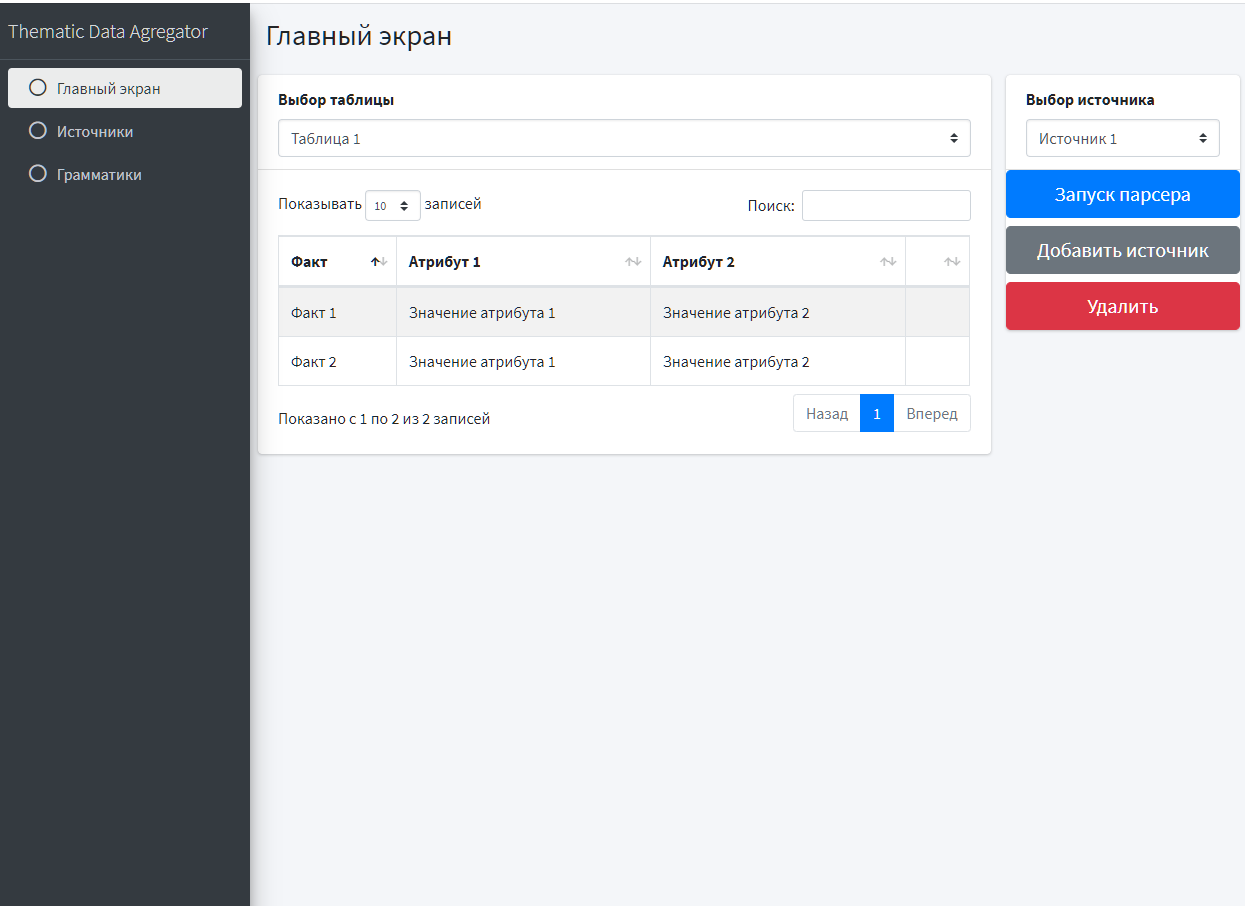


Рисунок 3.6 – Главный экран приложения

При нажатии на кнопку «Запуск парсера» запускается агрегация данных из текстов на естественном языке в выбранном источнике.

При нажатии на кнопку «Добавить источник» открывается карточка добавления нового источника. Карточка содержит пять полей для заполнения – название источника, «URL источника», куда вносится адрес интернет-ресурса, с которого предполагается производить агрегацию данных, «Типы фактов», куда вносятся описания типов фактов для извлечения в формате, используемом Томита-парсером, «Корневой словарь», куда вносятся статьи газеттира, используемого при работе с источником, и «Грамматика», где выбирается применяемая при работе с источником контекстно-свободная грамматика.

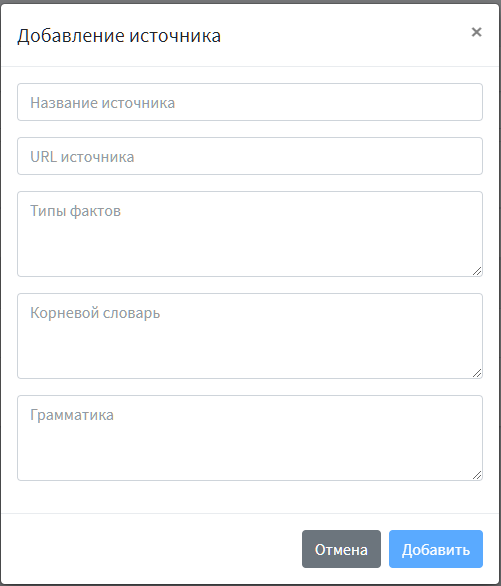


Рисунок 3.7 – Карточка создания нового источника

Кнопка «Добавить» является недоступной до тех пор, пока пользователь не заполнит как минимум поля «Название источника», «URL источника» и «Грамматика». После этого кнопка станет доступной для нажатия (рисунок 3.8)

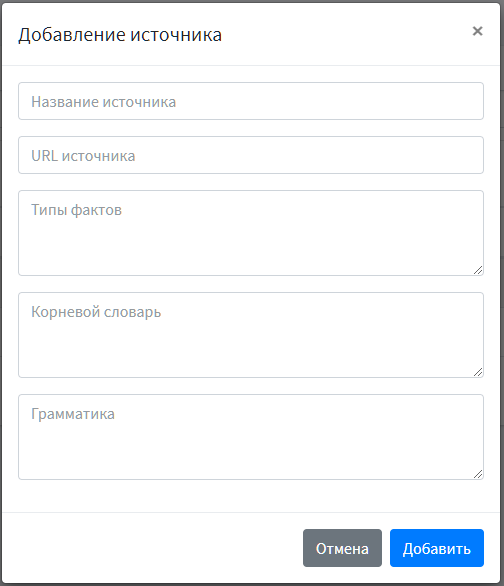


Рисунок 3.8 – Активная кнопка «Добавить» на карточке создания нового источника.

По нажатии на активную кнопку «Добавить» заданный источник сохранится в базу данных и станет доступен для выбора в выпадающем меню на главном экране приложения, находящемся прямо над кнопкой запуска парсера в правой части главного экрана приложения.

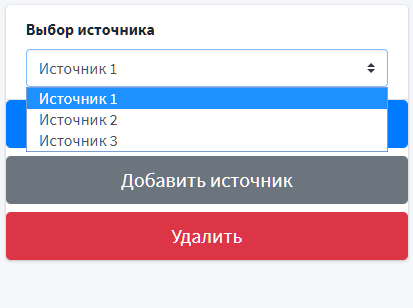


Рисунок 3.9 – Выбор источника на главном экране приложения

Вся информация приложения обрабатывается и сохраняется на сервер, на котором используется база данных Firebase. Логика взаимодействия приложения и сервера с БД представлена на рисунке 3.10.



Рисунок 3.10 – Логика взаимодействия приложения и Firebase

При нажатии на кнопку «Удалить» запись, выбранная в данный момент в таблице, будет удалена из таблицы и из базы данных.

В боковой панели на главном экране можно перейти на экран источников и экран грамматик, нажав на соответствующие кнопки.

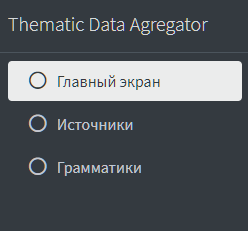


Рисунок 3.11 – Боковая навигационная панель

На рисунке 3.12 изображен экран работы с источниками.

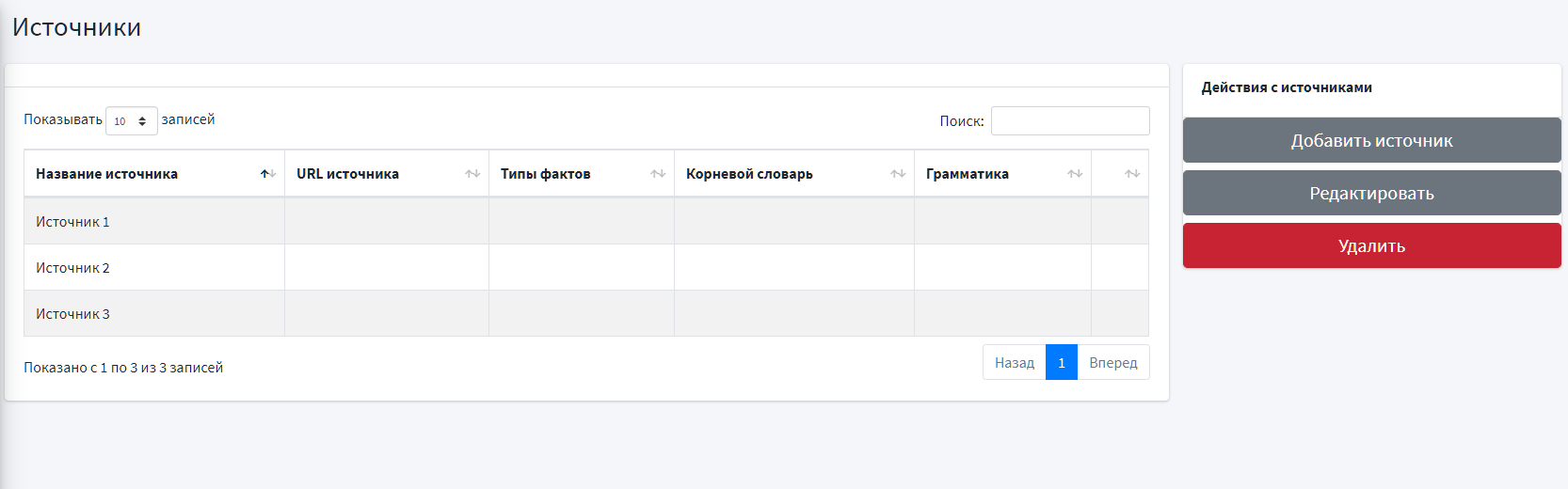


Рисунок 3.12 – Экран работы с источниками

Экран работы с источниками содержит таблицу источников из базы данных проекта, кнопки для работы с источниками «Добавить источник» и «Редактировать источник», а также кнопка удаления источников.

При нажатии на кнопку «Добавить источник» открывается карточка добавления источника, аналогичная такой же карточке на главном экране приложения.

При нажатии на кнопку «Редактировать» открывается карточка для редактирования источника, выбранного пользователем в таблице источников в центре экрана. Карточка содержит пять полей – название источника, «URL источника», куда вносится адрес интернет-ресурса, с которого предполагается производить агрегацию данных, «Типы фактов», куда вносятся описания типов фактов для извлечения в формате, используемом Томита-парсером, «Корневой словарь», куда вносятся статьи газеттира, используемого при работе с источником, и «Грамматика», где выбирается применяемая при работе с источником контекстно-свободная грамматика. Уже существующие и хранящиеся в базе данных значения полей для конкретного источника подтягиваются в поля карточки автоматически.

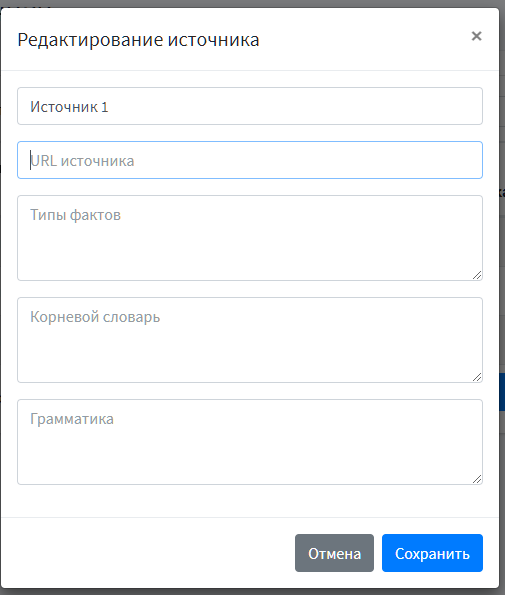


Рисунок 3.13 – Карточка редактирования источника

При нажатии на кнопку «Удалить» запись, выбранная в данный момент в таблице, будет удалена из таблицы и из базы данных.

В боковой панели экрана источников можно перейти на главный экран приложения и экран грамматик, нажав на соответствующие кнопки.

На экране работы с грамматиками пользователь может составлять контекстно-свободные грамматики для последующего использования при агрегации данных из новых источников на нужные темы, придерживаясь правил, описанных в разделе 2. Созданные в интерфейсе для работы с грамматиками КС-грамматики становятся доступны для применения в приложении. Их можно выбрать для использования с новым или уже существующим источником в карточке добавления или редактирования источника.

# **4 Разработка тематического агрегатора данных**

Разработанная система «Тематический агрегатор данных с применением алгоритмов анализа естественного языка» состоит из пяти модулей:

* Модуль работы с источниками
* Модуль работы с базой данных
* Модуль извлечения именованных сущностей
* Модуль работы с КС-грамматиками
* Модуль обработки действий пользователя

## **4.1 Модуль работы с источниками**

Основополагающая задача разрабатываемого агрегатора данных заключается в сборе контента с интернет-ресурсов. Таким образом, при разработке агрегатора данных в первую очередь необходимо разработать модуль приложения, обеспечивающие чтение данных с веб-страниц интернет-ресурсов, с которого производится агрегация, и их разбор с целью выделения и интерпретации фактов для последующего сохранения.

В процессе чтения текстов на естественном языке, расположенных на веб-странице, происходит работа Томита-парсера, который разбирает полученный текст и выделяет из него факты, извлекая именованные сущности с помощью контекстно-свободных грамматик, библиотеки которых составлены для каждой темы.

## **4.2 Модуль работы с базой данных**

Работа с базой данных – это ключевой функционал для тематического агрегатора данных, поэтому соответствующий модуль является одним из критических и наиболее объемных в приложении. Для подключения базы данных к приложению необходимо было выполнить следующий алгоритм действий:

* Зарегистрировать приложение на сайте Google
* Получить ключ авторизации и параметры удаленного подключения к базе и инициализировать его. При работе с веб-приложениями инициализация происходит прямо в HTML-коде веб-страницы, но в случае разрабатываемого тематического агрегатора необходимо работать с языком Python, поэтому для работы с базой данных Firebase в Python необходима обертка.

В листинге ниже описана инициализация API Firebase, необходимая для доступа к базе данных из приложения на Python.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | class Firebase: | |  |  | |  | def \_\_init\_\_(self, config): | |  | self.api\_key = config["apiKey"] | |  | self.auth\_domain = config["authDomain"] | |  | self.database\_url = config["databaseURL"] | |  | self.storage\_bucket = config["storageBucket"] | |  | self.credentials = None | |  | self.requests = requests.Session() | |  | if config.get("serviceAccount"): | |  | scopes = [ | |  | 'https://www.googleapis.com/auth/firebase.database', | |  | 'https://www.googleapis.com/auth/userinfo.email', | |  | "https://www.googleapis.com/auth/cloud-platform" | |  | ] | |  | service\_account\_type = type(config["serviceAccount"]) | |  | |  |  | |  |  | |  |  | |  | |  | |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |  |  | |

Для того, чтобы API Firebase в тематическом агрегаторе данных могло получить доступ к заданным атрибутам (аутентификационному ключу API, ссылке на адрес приложения в сервисе Firebase, ссылке на саму базу данных и на используемое ей облачное хранилище), и при этом инкапсулировать эти поля от постороннего доступа, нужно реализовать доступ к этим полям через методы. В следующем листинге определяются методы для доступа к полям аутентификатора, базы данных и хранилища данных.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | |  | if service\_account\_type is str: | |  | self.credentials = ServiceAccountCredentials.from\_json\_keyfile\_name(config["serviceAccount"], scopes) | |  | if service\_account\_type is dict: | |  | self.credentials = ServiceAccountCredentials.from\_json\_keyfile\_dict(config["serviceAccount"], scopes) | |  | if is\_appengine\_sandbox(): | |  | |  | |  | |  | adapter = appengine.AppEngineAdapter(max\_retries=3) | |  | else: | |  | adapter = requests.adapters.HTTPAdapter(max\_retries=3) | |  |  | |  | for scheme in ('http://', 'https://'): | |  | self.requests.mount(scheme, adapter) | |  |  | |  | def auth(self): | |  | return Auth(self.api\_key, self.requests, self.credentials) | |  |  | |  | def database(self): | |  | return Database(self.credentials, self.api\_key, self.database\_url, self.requests) | |  |  | |  | def storage(self): | |  | return Storage(self.credentials, self.storage\_bucket, self.requests) | |

После этого экземпляр класса firebase() будет доступен для использования в приложении. Теперь нужно разработать методы для непосредственного взаимодействия с базой данных – работы с записями базы и создания подтаблиц.

В листинге ниже определяется подкласс Database, реализующий базу данных как непосредственный объект, с которым и будет взаимодействовать приложение:

|  |
| --- |
| **class** Database:**def** \_\_init\_\_(self, credentials, api\_key, database\_url, requests):  **if not** database\_url.endswith(**'/'**):  url = **''**.join([database\_url, **'/'**])  **else**:  url = database\_url  self.credentials = credentials  self.api\_key = api\_key  self.database\_url = url  self.requests = requests   self.path = **""** self.build\_query = {}  self.last\_push\_time = 0  self.last\_rand\_chars = [] |

Внутри этого подкласса создадим методы для работы с таблицами и записями. Далее определяется функция child, с помощью которой приложение будет порождать новые подтаблицы, что изначально предусмотрено моделью данных тематического агрегатора – в случае работы с новыми типами фактов или новой тематикой необходимо создание новой таблицы, так как атрибуты и взаимосвязь фактов в случае разных тематик будут заметно различаться.

Затем определяются базовые необходимые методы для работы с записями таблиц базы данных – методы, реализующие запись информации в базу и удаление существующих записей. Для работы с базой данных также необходимо иметь возможность редактировать записи и обновлять таблицы. Для этих действий также были разработаны методы.

Таким образом, основной функционал для записи и редактирования информации, полученной модулем извлечения именованных сущностей, уже реализован, но для комфортной работы пользователя с результатами работы агрегатора данных необходимо определить методы, позволяющие более эффективно находить нужные записи в таблице – сортировку и фильтры. Реализация всех вышеописанных методов приведена в листинге ниже:

|  |
| --- |
| **def** sort(self, origin, by\_key):pyres = origin.each()  new\_list = []  **for** pyre **in** pyres:  new\_list.append(pyre.item)data = sorted(dict(new\_list).items(), key=**lambda** item: item[1][by\_key])  **return** PyreResponse(convert\_to\_pyre(data), origin.key())  **def** order\_by\_key(self):  self.build\_query[**"orderBy"**] = **"$key"  return** self  **def** order\_by\_value(self):  self.build\_query[**"orderBy"**] = **"$value"  return** self  **def** order\_by\_child(self, order):  self.build\_query[**"orderBy"**] = order  **return** self |

Для работы с базой данных необходимы и утилитарные методы, реализующие отправку запросов из приложения, разработанного на языке Python, в базу данных Firebase, рассчитанную изначально на работу с веб-приложениями. Так как Firebase хранит свои базы данных, а соответственно, и обрабатывает запросы, в формате JSON, то разрабатываемому приложению для полноценного взаимодействия с базой данных необходима возможность генерировать такие запросы.

Кроме метода, обеспечивающего построение запросов к базе данных в формате JSON, необходим и метод, непосредственно отправляющий эти запросы в базу данных Firebase и, что критически важно, возвращающий полученный ответ базы данных, так как необходимо в реальном времени представлять изменения в таблицах базе данных пользователю. Реализация этого метода приведена в нижеследующем листинге:

|  |
| --- |
| **def** get(self, token=None, json\_kwargs={}):  build\_query = self.build\_query  query\_key = self.path.split(**"/"**)[-1]  request\_ref = self.build\_request\_url(token)headers = self.build\_headers(token) request\_object = self.requests.get(request\_ref, headers=headers)  raise\_detailed\_error(request\_object)  request\_dict = request\_object.json(\*\*json\_kwargs) **if** isinstance(request\_dict, list):  **return** PyreResponse(convert\_list\_to\_pyre(request\_dict), query\_key)  **if not** isinstance(request\_dict, dict):  **return** PyreResponse(request\_dict, query\_key)  **if not** build\_query:  **return** PyreResponse(convert\_to\_pyre(request\_dict.items()), query\_key) |

Для метода также нужно разработать поведение при получении ответа от базы данных на разные варианты запросов, в частности, форматы представления и сортировки записей, являющихся результатом запроса к базе.

В качестве примера программной реализации работы тематического агрегатора данных с базой данных Firebase будет рассмотрено сохранение извлеченных из текста фактов и атрибутов в базу данных.

|  |
| --- |
| auth = firebase.auth()  user = auth.sign\_in (email, password)  db = firebase.database()  data = {  fact: factValue  attribute: attribValue  }  results = db.child("facts").push(data, user['idToken']) |

В примере выше происходит авторизация в сервис Firebase, инициализируется база данных, затем извлеченный агрегатором факт и его атрибут загружаются в базу данных. В БД при этом также создается новая таблица, как в случае обнаружения новой тематики.

**4.3 Модуль извлечения именованных сущностей**

Разработанный агрегатор данных использует для извлечения именованных сущностей инструмент Томита-парсер. Томита-парсер, использующий алгоритм GLR-анализатора, является эффективным инструментом для извлечения именованных сущностей из текста на естественном языке при помощи контекстно-свободных грамматик и словарей ключевых слов. Особо стоит отметить, что Томита-парсер позволяет выполнять фрагментарный анализ. Это означает, что факт группируется из фраз, стоящих в разных частях предложения. Следовательно, необходимо задать множество КС-грамматик, описывающих синтаксическую структуру извлекаемых фактов, и словарь ключевых слов, которые используются в процессе анализа КС-грамматиками. Каждая отдельная статья в словаре – это группа ключевых слов с общими свойствами.

Когда Томита-парсер интерпретирует факты, они передаются Python-оболочке для дальнейшего сохранения в базу данных. Листинг, реализующий этот функционал, приведен ниже:

|  |
| --- |
| facts = [] leads = [] etree\_root = ElementTree.fromstring(out) **if** etree\_root.find(**"document"**) **is not** None:  **if** with\_facts:  self.debug(**"Parsing facts..."**)  **for** fact **in** etree\_root.find(**"document"**).find(**"facts"**):  facts.append({  **"fact\_id"**: fact.attrib.get(**"FactID"**),  **"lead\_id"**: fact.attrib.get(**"LeadID"**),  **"pos"**: fact.attrib.get(**"pos"**),  **"len"**: fact.attrib.get(**"len"**),  **"fact"**: fact.find(**"Name"**).attrib.get(**"val"**)  })  **if** with\_leads:  self.debug(**"Parsing leads..."**)  **for** lead **in** etree\_root.find(**"document"**).find(**"Leads"**):  leads.append({  **"lead\_id"**: lead.attrib.get(**"id"**),  **"lead"**: lead.attrib.get(**"text"**)  }) **return** facts, leads |

**4.4 Модуль работы с КС-грамматиками**

Данный модуль позволяет пользователю составлять и редактировать КС-грамматики для нужных тематик и источников, придерживаясь синтаксиса КС-грамматик, используемых Томита-парсером. КС-грамматика – это множество правил, описывающих синтаксическую структуру выделяемых цепочек. Одна цепочка соответствует одному предложению в тексте. Перед запуском терминалы грамматики отображаются на слова (или словосочетания, об этом будет сказано ниже) предложения. Одному слову может соответствовать много терминальных символов. Таким образом, парсер получает на вход последовательность множеств терминальных символов. Из цепочки выделяются подцепочки, которые, в свою очередь, интерпретируются в разбитые по полям факты.

КС-грамматики составляются по нижеследующим правилам.

Грамматики, которые выделяют подцепочки, выглядят так:

S -> Verb;

Это правило выделяет из цепочки все глаголы. В данном правиле Verb является терминалом. Терминал может стоять только в правой части правила. В Томите к терминалам относятся названия частей речи (Noun, Verb, Adj) , некоторые символы (Comma, Punct, Ampersand, PlusSign), а также морфологические леммы, выделяемые одинарными кавычками.

S – это нетерминал, который строится из терминалов и должен хотя бы один раз встретиться в левой части правила. Если нетерминал встречается только в левой части и никогда – в правой, значит, это вершина грамматики. Ее также можно указать явно при помощи директивы #GRAMMAR\_ROOT.

Чтобы извлекать из текста не только отдельные существительные, но и, например, именные группы, применяется операция конкатенации. В качестве оператора конкатенации выступает обычный пробел:

S -> Adj Noun;

Файл грамматики состоит из двух частей: блока с директивами, влияющими на работу грамматики в целом, и множества правил.

Директивы начинаются со знака # и заканчиваются переносом строки. Некоторые директивы совпадают по смыслу с аналогичными директивами препроцессора языка C++.

Указывает кодировку данного файла с грамматикой. Кодировка по умолчанию: utf-8. Кодировка указывается в кавычках.

Включает в данную грамматику текст другой грамматики. При этом фильтры грамматик суммируются, а корень включаемой грамматики (см. #grammar\_root) игнорируется. Называние включаемой грамматики указывается в кавычках.

Директива #GRAMMAR\_ROOT указывает нетерминал, который является корнем данной грамматики. Корень грамматики можно не задавать явно, если в грамматике есть только один нетерминал, который ни разу не встречается в правой части правил грамматики.

Директива #GRAMMAR\_KWSET позволяет явно указать имена или типы статей из газеттира, чьи найденные в предложении ключи должны передаваться парсеру в качестве терминалов. C точки зрения текущей

грамматики, эти цепочки становятся multiword-ами. В частности директива #GRAMMAR\_KWSET может использоваться для уточнения значения пометы kwtype=none.

Синтаксис описания статей аналогичен синтаксису пометы kwset: перечисления в квадратных скобках [ ] через запятую.

Фильтры позволяют ускорить работу грамматик. Если предложение на входе не соответствует ни одному из объявленных фильтров, то грамматика на них не запускается.

Фильтр записывается как последовательность терминалов. Предложение проходит фильтр если в нем в указанном порядке встретились слова описываемые этими терминалами. Расстояние между словами может быть любым.

Перед названиями терминалов в фильтрах ставится амперсанд &. У терминалов могут быть их обычные пометы: kwtype, wff и другие. Операторы (+, \* и другие) в фильтрах использовать нельзя. Между терминалами может быть указано максимально допустимое расстояние в квадратных скобках.

Если в грамматику включается другая грамматика директивой #include, то фильтры включаемой грамматики тоже учитываются.

Директива #define также используется для реализации подстановок. В этом случае у директивы два аргумента: имя подстановки и ее значение. Именем подстановки может быть любая последовательность из букв, цифр и символа подчеркивания, первый символ имени подстановки не может быть цифрой. Значением подстановки может быть любое завершенное выражение, в синтаксисе, разрешенном в правой части правил грамматики: список граммем, цепочка нетерминалов с ограничениями и интерпретациями и т.п. Знак переноса строки означает конец значения переменной.

После объявления имя подстановки можно использовать с помощью синтаксиса в фигурных скобках со знаком доллара перед ним: ${ … }. Между знаком $, фигурными скобками и именем подстановки пробелы не допускаются.

Следует понимать, что подстановка «подставляется» не буквально (как, например, в языке C++), а сама по себе является сложным токеном в тексте грамматики. Поэтому подстановки нельзя использовать внутри строкового литерала (там она просто не будет распознана), а также внутри идентификаторов и других длинных токенов.

Все определенные с помощью #define имена действуют дальше по тексту грамматики, в том числе во всех включенных позже файлах. Подстановку можно отменить, указав ее имя после директивы #undef. Рекомендуется всегда отменять подстановки явно, чтобы избежать неожиданных эффектов от включения одной грамматики в другую.

Данный модуль делает программный комплекс более универсальным, создавая для пользователя возможность извлечения фактов из текстов новых тематик.

## **4.5 Модуль обработки действий пользователя.**

Все действия пользователя во время пользования системой реализуются через модуль обработки действий. В общем виде действия пользователя можно представить в виде Use-case диаграммы (рисунок 4.1).

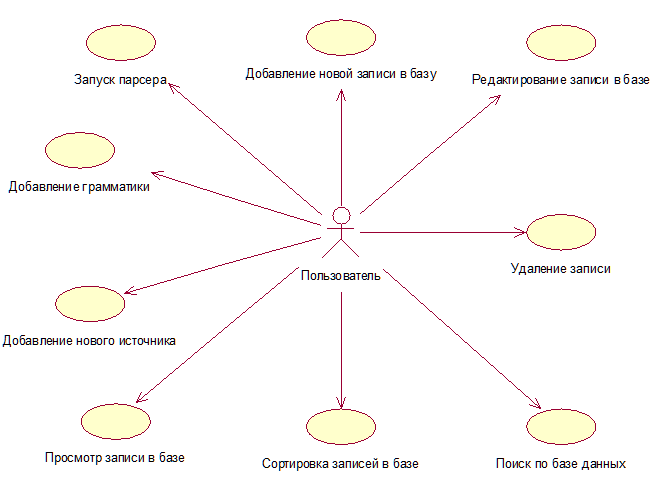


Рисунок 4.1 – Use-case диаграмма приложения

При нажатии на кнопку «Запуск парсера» происходит запуск агрегатора, и база данных обновляется в реальном времени – собранные факты добавляются в базу данных по мере их выделения и интерпретации модулем извлечения именованных сущностей. Если агрегатор ранее не работал с указанной тематикой, то, закончив работу, модуль извлечения именованных сущностей создаст в базе данных новую таблицу. Затем таблица загрузится в веб-интерфейс и откроется на главном экране приложения.

При введении запроса в поле поиска пользователь может осуществлять поиск нужных записей в таблице. Поиск по загруженной из базы данных таблице произойдет в реальном времени, и результаты будут выведены пользователю в таблице.

Содержимое таблицы можно также отсортировать по возрастанию или убыванию значения одного из столбцов таблицы. Для этого нужно нажать на заголовок столбца, один раз для сортировки по возрастанию и второй раз для сортировки по убыванию.

При нажатии на кнопку «Источники» открывается экран, содержащий список источников из базы данных агрегатора, а также элементы управления, необходимые для добавления, редактирования и удаления источников.

При нажатии на кнопку «Добавить источник» для пользователя откроется карточка добавления источника с полями для заполнения, соответствующими необходимым параметрам источника (название, URL источника и т.д.). Когда пользователь заполнит эти поля и подтвердит добавление источника нажатием на кнопку «Добавить», в таблицу источников в базе данных будет добавлен источник с параметрами, соответствующими значениям в полях ввода для названия, URL источника и пр.

При нажатии на кнопку «Редактировать источник» пользователь может изменить записанные в базе параметры источника (название, URL источника и пр.) в карточке редактирования, в общих чертах аналогичной карточке добавления источника, описанной выше.

При нажатии на кнопку «Удалить» выбранная в данный момент запись таблицы или источник удаляется из базы данных.

При нажатии на кнопку «Добавить грамматику» в базу данных будет добавлен источник с параметрами, соответствующими значениям в полях ввода для названия, URL источника и прочих атрибутов грамматики.

**5 Апробация разработанного тематического агрегатора данных**

**5.1 Постановка задачи и подготовка к апробации агрегатора данных**

Для проверки корректности работы созданной системы необходимо протестировать ее работу на практическом примере. Для тестирования возьмем пример текста на естественном языке, содержащий информацию о музыкальной группе и выпущенных ей альбомах.

Этапы тестирования включат в себя:

* Определение типов фактов, которые будут извлечены из текста. Если мы подразумеваем извлечение фактов об альбомах, выпущенных музыкальной группой, в типы фактов должны войти название группы, названия и года выпуска альбомов. Составим описание типов фактов в установленном формате:

|  |
| --- |
| import "base.proto";  import "facttypes\_base.proto";  message Group: NFactType.TFact  {  required string Name = 1;  }  message Album: NFactType.TFact  {  required string Name = 1;  required string Year = 2;  } |

* Для работы с контекстно-свободной грамматикой потребуется не только грамматическая, но и семантическая информация о словах из текста. Для этого нам надо построить словарь. Следовательно, следующим этапом будет составление корневого словаря-газеттира. В него войдут ключевые слова, по которым парсер будет искать в тексте на естественном языке нужные факты.

Словарь для Томита-парсера называется газеттиром и пишется в отдельном файле с расширением gzt. Газеттир состоит из статей.

|  |
| --- |
| encoding "utf8";  import "base.proto";  import "articles\_base.proto";  import "facttypes.proto";  TAuxDicArticle "слово\_группа"  {  key = "группа"  key = "ансамбль"  key = "выступает"  }  TAuxDicArticle "слово\_альбом"  {  key = "альбом"  key = "релиз"  }  TAuxDicArticle "группы"  {  key = { "tomita:group.cxx" type=CUSTOM }  } |

В данном случае TAuxDicArticle – типы статей, в кавычки взяты их названия, а в поле key перечисляются все слова, входящие в эту статью. Теперь на созданную статью можно будет ссылаться из грамматики. Для этого используются пометы kwtype и kwset, в качестве значения которых выступает тип или имя статьи. Тип TAuxDicArticle является типом по умолчанию и используется во многих статьях, поэтому при составлении грамматики мы будем ссылаться на уникальные названия статей.

* + Теперь перейдем к самому важному этапу: составлению контекстно-свободной грамматики для парсинга текста. Составим КС-грамматику, придерживаясь правил и синтаксиса, описанных выше, в разделе 2.

|  |
| --- |
| #encoding "utf-8"  #GRAMMAR\_ROOT S  AlbumDate -> AnyWord<wff = "(19[0-9]{2})|(20[0-1][0-9])">;  AlbumName -> AnyWord<kwtype='слово\_альбом'> AnyWord<h-reg1, l-quoted> AnyWord<~r-quoted>\*  AnyWord<r-quoted> | AnyWord<kwtype='слово\_альбом'> AnyWord<h-reg1, l-quoted, r-quoted>;  GroupName -> Word<kwtype="слово\_группа", nc-agr[1]> Word<h-reg1, nc-agr[1]>+;  S -> GroupName interp(Group.Name::not\_norm);  S -> AlbumDate interp(Album.Year) OrdinalNumeral\* 'год' AnyWord\* AlbumName interp(Album.Name) |  AlbumName interp(Album.Name) OrdinalNumeral\* AlbumDate interp(Album.Year) 'год';  AlbumDate -> OrdinalNumeral interp(Album.Year); |

Теперь все необходимое для тестирования системы готово.

**5.2 Тестирование тематического агрегатора данных**

Когда готовы описание типов фактов, корневой словарь-газеттир и контекстно-свободная грамматика, создадим новый источник на экране источников веб-приложения.

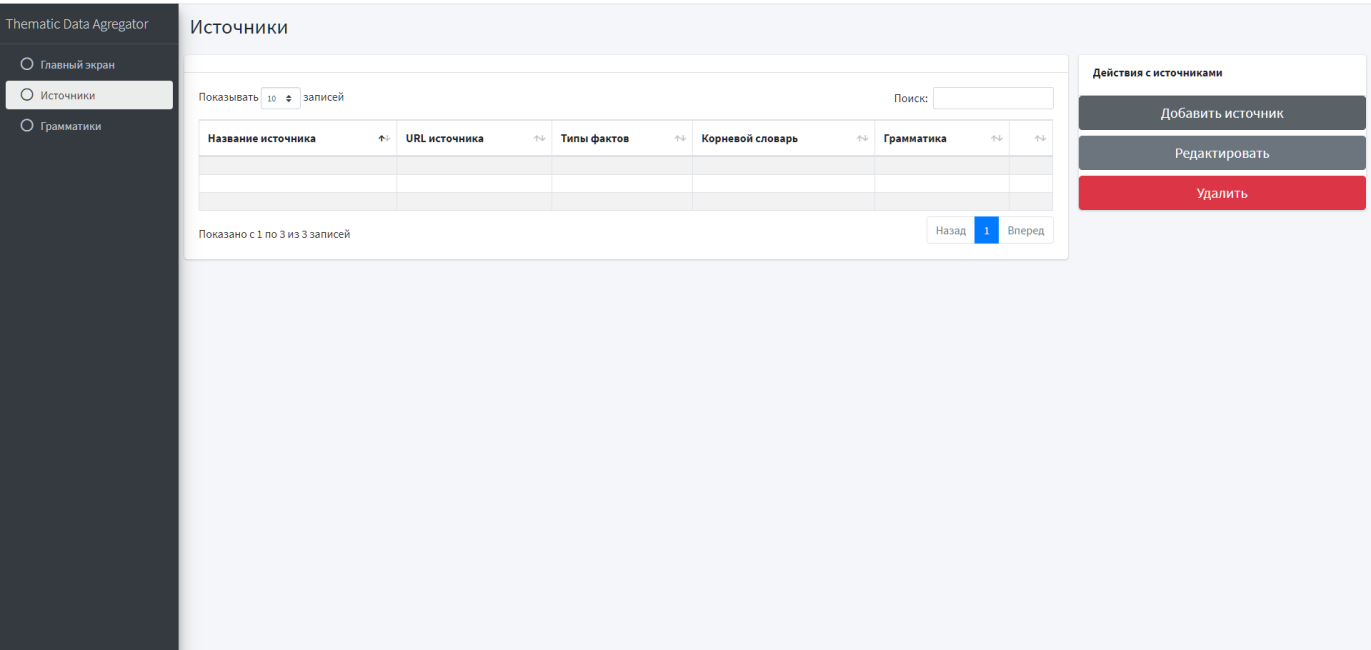


Рисунок 5.1 – Экран источников

Откроем карточку создания источника и внесем в нее необходимые данные.

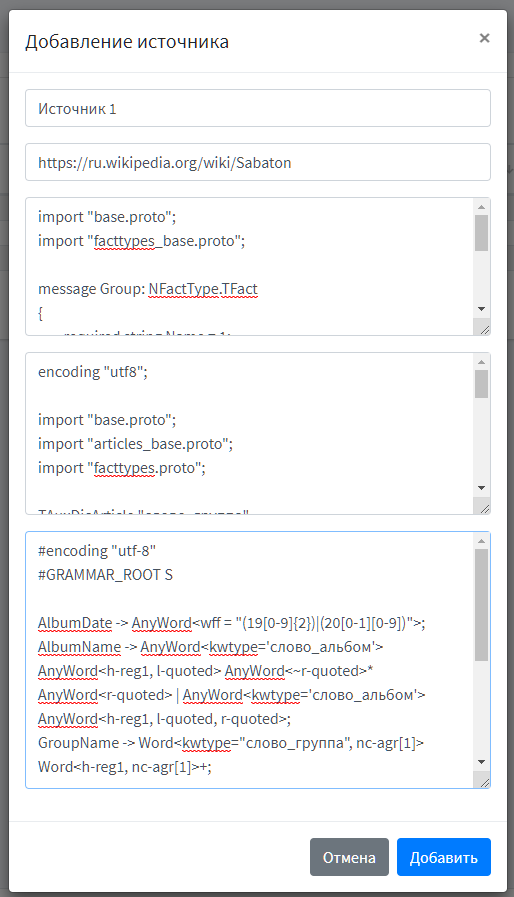


Рисунок 5.2 – Создание источника

Нажав кнопу «Добавить», мы сохраним источник в базу данных.

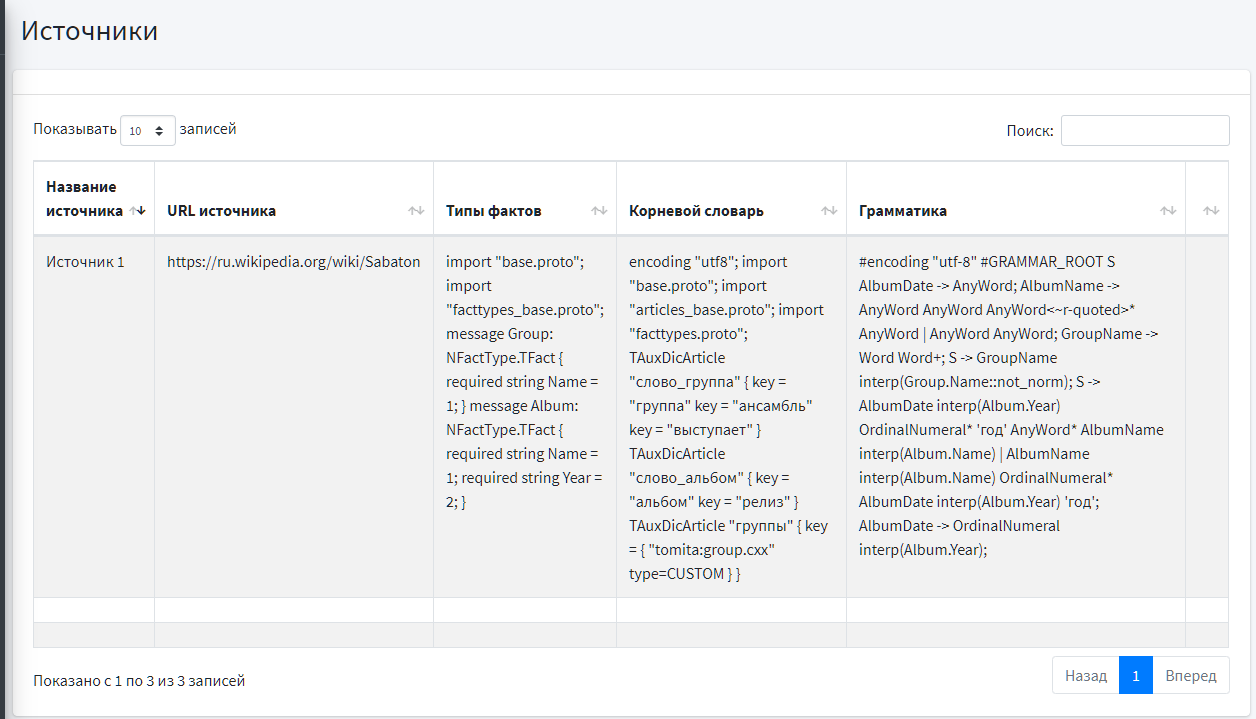


Рисунок 5.3 – Таблица с добавленным источником

Теперь перейдем на главный экран приложения. Перед началом агрегации выберем нужный источник, из которого будет взят текст на естественном языке, подлежащий разбору парсером. Для этого откроем выпадающий список, расположенный над кнопкой запуска парсера в панели справа, выберем источник и запустим работу парсера, нажав на кнопку «Запуск парсера».

Закончив работу, модуль извлечения именованных сущностей создает в базе данных новую таблицу (так как приложение раньше не работало с такой тематикой), открывает созданную таблицу, и добавляет в нее полученные факты и атрибуты. Затем таблица загружается в веб-интерфейс и сразу открывается на главном экране приложения. Впоследствии отображаемую на главном экране приложения таблицу из базы данных можно переключить, воспользовавшись выпадающим списком, расположенным в центральной части главного экрана, прямо над таблицей.

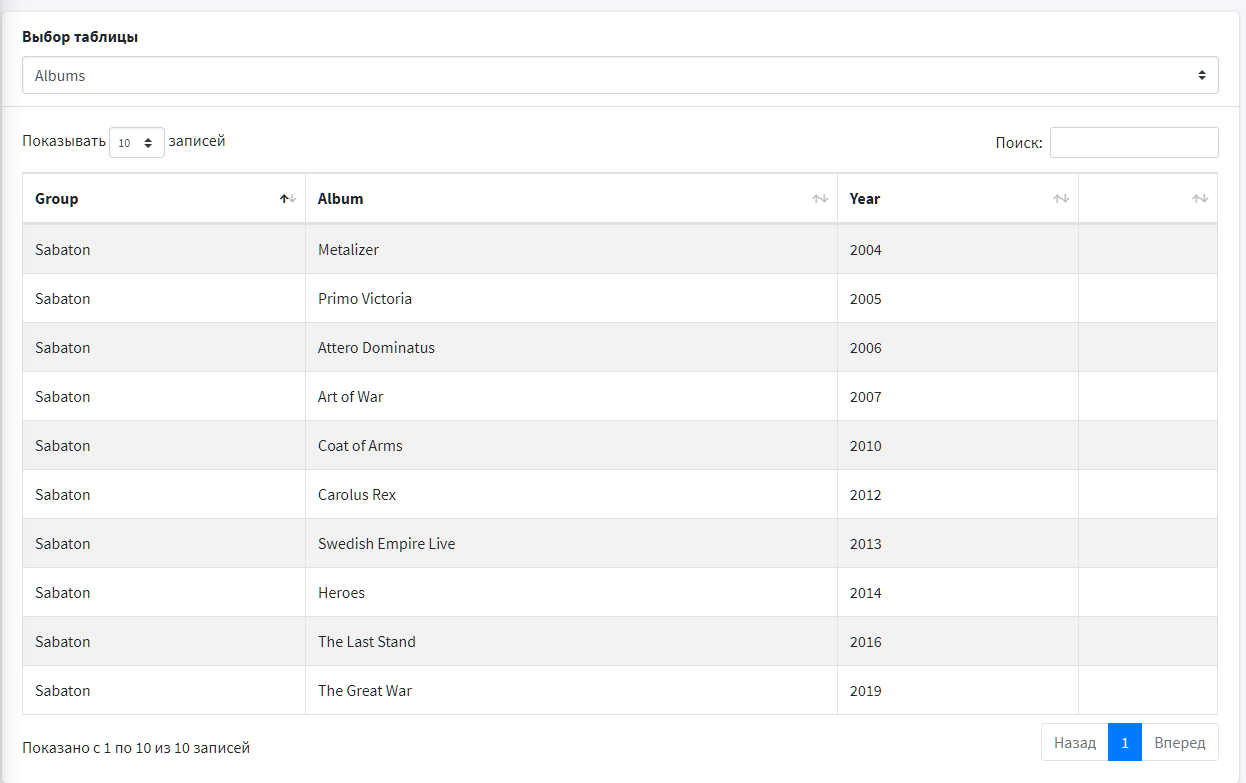


Рисунок 5.4 – Результат работы тематического агрегатора данных

Мы можем видеть, что агрегатор, основываясь на заданной контекстно-свободной грамматике, выделил из текста на естественном языке (статьи на интернет-ресурсе) названия музыкальных альбомов в том порядке, в каком они встречались в исходном тексте, и связал их с годами их выхода. В данном примере не только извлечение, но и интерпретация фактов сработали успешно.

На полученных данных протестируем вспомогательную функцию разработанного приложения – поиск по таблице. Для этого введем в поле поиска запрос. Поиск по загруженной из базы данных таблице произойдет сразу же в реальном времени, и результаты будут выведены пользователю в таблице. Содержимое таблицы можно также отсортировать по возрастанию или убыванию значения одного из столбцов таблицы.

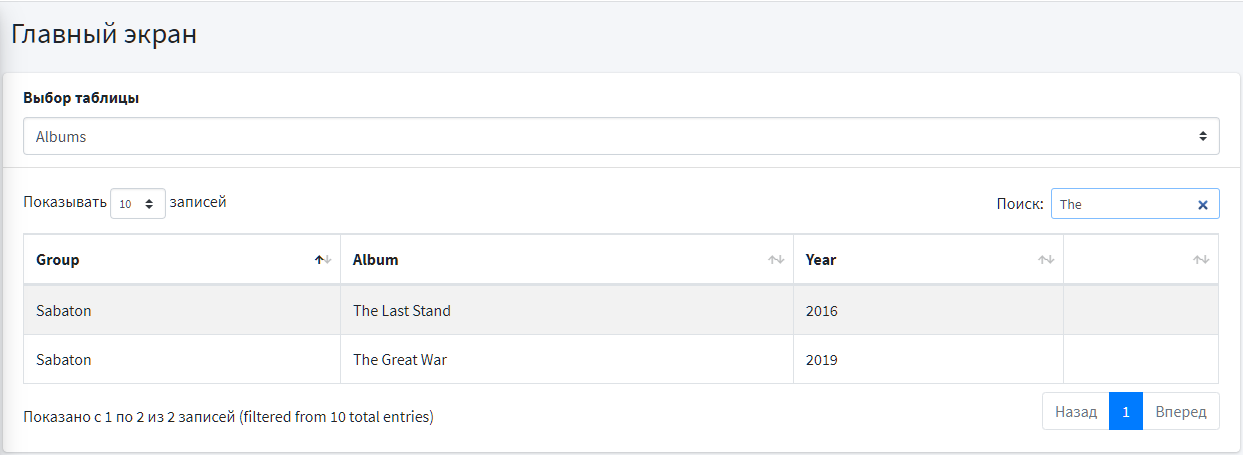


Рисунок 5.5 – Результаты тестового поиска по таблице из базы данных

Из рисунка видно, что функция поиска по таблице базы данных работает корректно, обеспечивая пользователю агрегатора удобство работы с содержимым.

## **Результаты тестирования тематического агрегатора данных**

Результатом работы системы на данном тестовом примере стало успешное добавление источника извлечение и интерпретация фактов по заданной тематике, а также последующая работа с полученным контентом. Программный комплекс «Тематический агрегатор данных» показал себя полностью функциональным и готовым к работе.

Апробация программного комплекса на 42 источниках показала, что из 603 извлеченных фактов 472 были извлечены и интерпретированы верно, что составляет приблизительно 78%. Таким образом, точность работы комплекса достаточно высока. Проблема неверно извлеченных и интерпретированных фактов может быть решена с помощью модерации базы извлеченных фактов вручную, для чего разработанный программный комплекс предоставляет инструментарий.

Более подробная статистика по тестированию программного комплекса приведена в таблице 2.

Таблица 2 – Статистика по апробации тематического агрегатора данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Выборка по тематике текстов | Число источников | Число извлеченных фактов | Фактов извлечено верно | Процент верного извлечения фактов |
| Музыка | 10 | 142 | 109 | 76% |
| Фильмы | 10 | 129 | 98 | 75% |
| Литература | 10 | 150 | 128 | 85% |
| Новости | 12 | 182 | 137 | 75% |
| Итого | 42 | 603 | 472 | 78% |

В процессе тестирования были сделаны выводы о том, что работа приложения с текстами на естественном языке различных тематик требует большого количества контекстно-свободных грамматик, вследствие чего было принято решение подготовить библиотеку КС-грамматик, охватывающую широкий спектр тем и типов фактов, извлекаемых из текста.

# **ЗАКЛЮЧЕНИЕ**

Реализованный программный комплекс планируется использовать для решения задачи автоматического извлечения и структурирования (то есть агрегации) данных из документов на естественном языке, что позволит обеспечить оптимальную организацию процесса сбора данных, а также увеличит простоту поиска нужного контента для пользователя.

В первой главе были проведен обзор современной библиографии, научных исследований в области агрегации данных, обработки естественного языка и извлечения именованных сущностей. Было проанализировано положение дел в области извлечения именованных сущностей, рассмотрены основные направления анализа естественных языков и возможность их применения для решения поставленной задачи, методы извлечения именованных сущностей и их возможное применение в агрегации данных, алгоритмы парсинга данных на естественном языке, в том числе алгоритм с использованием контекстно-свободных грамматик, проведен сравнительный анализ программных средств для извлечения именованных сущностей из текстов на естественном языке.

Во второй главе было рассмотрено использование контекстно-свободных грамматик в инструменте «Томита-парсер» для извлечения данных из текстов на естественном языке, изучены принципы работы и синтаксис контекстно-свободных грамматик, используемых в инструменте «Томита-парсер», возможности инструмента «Томита-парсер» по интерпретации извлеченных фактов с помощью контекстно-свободных грамматик, в том числе интерпретация КС-грамматикой нескольких фактов одновременно.

В третьей главе были проанализированы функциональные требования к разрабатываемому программному комплексу, проведено проектирование приложения. Были построены UML-диаграммы: диаграмма вариантов использования, диаграмма классов, диаграммы последовательности, а также была проведена конкретизация вариантов использования с выделением потоков событий. Была спроектирована модель данных, используемая в приложении, выбраны инструменты реализации программного комплекса, смоделировано поведение системы, разработан прототип тематического агрегатора данных

В четвертой главе были разработаны модули и компоненты системы, которые в совокупности при взаимодействии друг с другом обеспечивают надежную и бесперебойную работу приложения в целом, удобство и комфорт работы с ним пользователя. Были разработаны модуль извлечения именованных сущностей, позволяющий извлекать и интерпретировать факты из текстов на естественном языке, модуль работы с источниками, позволяющий добавлять и настраивать источники контента, модуль работы с базой данных, позволяющий приложению взаимодействовать с NoSQL-базой данный Firebase, модуль работы с КС-грамматиками, позволяющий задавать контекстно-свободные грамматики для работы с новыми источниками, модуль обработки действий пользователя, посредством которого пользователь взаимодействует со всем комплексом.

В пятой главе была апробирована работа программного комплекса – тематического агрегатора данных с применением анализа естественного языка, были сделаны выводы о корректной работе разработанного приложения. В ходе апробации система была протестирована на 42 источниках (интернет-ресурсах), точность извлечения фактов из текстов на естественном языке составила 78%.

Таким образом, приложение позволяет производить агрегацию данных из текстов на естественном языке, производить выделение фактов, их интерпретацию. В будущем планируется расширить библиотеки КС-грамматик с целью обеспечить охват более широкого спектра тем и типов фактов, извлекаемых из текста.

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Беззуб М. Д. Парсинг сайтов как универсальное средство добавления контента / М. Д. Беззуб, О. И. Чуйко // Современные тенденции и проекты развития информационных систем и технологий. – 2016. – С. 292-296.
2. Барсегян А. Методы и модели анализа данных: OLAP и Data Mining. / А. Баргесян // – 2004.
3. Большакова Е. И. Автоматическая обработка текстов на естественном языке и компьютерная лингвистика. / Е. И. Большакова // – 2015.
4. Анисимов А. В. Система обработки текстов на естественном языке / А. В. Анисимов, А. А. Марченко // Искусственный интеллект. – 2002. – №. 4. – С. 157-163.
5. Евграфов Д. А. Разработка алгоритмического и программного обеспечения модуля автоматизированного извлечения контента веб-ресурсов. / Д. А. Евграфов // – 2018.
6. Диковицкий В. В. Обработка текстов естественного языка в моделях поисковых систем / В. В. Диковицкий, М. Г. Шишаев // Труды Кольского научного центра РАН. – 2010. – №. 3.
7. Зеленский И. С. Извлечение структурированного описания объектов недвижимости из пользовательских записей на естественном языке. / И. С. Зеленский //Известия Волгоградского государственного технического университета. – 2017. – №. 14. – С. 41-46.
8. Пальчунов Д. Е. Поиск и извлечение знаний: порождение новых знаний на основе анализа текстов естественного языка / Д. Е. Пальчунов // Философия науки. – 2009. – №. 4. – С. 70-90.
9. Тихомиров И. А. Метод динамической контентной фильтрации сетевого трафика на основе анализа текстов на естественном языке / И. А. Тихомиров, И. В. Соченков // Вестник Новосибирского государственного университета. Серия: Информационные технологии. – 2008. – Т. 6. – №. 2.
10. Виолентов И. А. Анализ подходов и инструментов для извлечения именованных сущностей / И. А. Виолентов // Труды 20-ой Международной научно-технической конференции «Информационные системы и технологии». – 2014. – С. 227.
11. Судоплатова С. Н. Анализ технологии создания системы извлечения именованных сущностей из текстов с использованием Томита-парсера / С. Н. Судоплатова, М. А. Домрачев // МНСК-2017: Прикладная лингвистика. – 2017. – С. 32-33.
12. Степанов П. А. Автоматизация обработки текстов естественного языка / П. А. Степанов // Вестник Новосибирского государственного университета. Серия: Информационные технологии. – 2013. – Т. 11. – №. 2
13. Дударь З. В. Семантическая нейронная сеть, как формальный язык описания и обработки смысла текстов на естественном языке / З. В. Дударь, Д. Е. Шуклин // Радиоэлектроника и информатика. – 2000. – №. 3
14. Найденова К. А. Машинное обучение в задачах обработки естественного языка: обзор современного состояния исследований / К. А. Найденова, О. А. Невзорова // Ученые записки Казанского университета. Серия Физико-математические науки. – 2008. – Т. 150. – №. 4.
15. Золотарев О. В. Методы выделения процессов, объектов, отношений из текстов естественного языка / О. В. Золотарев // Проблемы безопасности российского общества. – 2014. – №. 3-4. – С. 276-283.
16. Степанов П. А. Системы анализа текстов естественного языка / П. А. Степанов // Альманах современной науки и образования. – 2013. – №. 6. – С. 159-161.
17. Мельников А. В. Об Использовании машинного обучения в задачах обработки естественного языка на примере анализа образовательного контента / А. В. Мельников, Д. С. Ботов, Ю. Д. Кленин // Онтология проектирования. – 2017. – Т. 7. – №. 1 (23). – С. 34-47.
18. Рубайло А. В. Программные средства извлечения информации из текстов на естественном языке / А. В. Рубайло, М. Ю. Косенко // Альманах современной науки и образования. – 2016. – №. 12. – С. 87-92.
19. Харабет Я. К. Автоматическое выделение количественных конструкций в русскоязычных научно-популярных текстах / Я. К. Харабет //XVIII Объединенная научная конференция «Интернет и современное общество»(IMS‐2015), Санкт-Петербург. – 2015. – С. 23-25.
20. Tomita M. An efficient augmented-context-free parsing algorithm / M. Tomita //Computational linguistics. – 1987. – Т. 13. – №. 1-2. – С. 31-46
21. Ермакова Л. М. Методы извлечения информации из текста. / Л. М. Ермакова // – 2015.
22. Большакова Е. И. Инструментальные средства для разработки систем извлечения информации из русскоязычных текстов / Е. И. Большакова, Н. Э. Ефремова, Г. Ф. Шариков // Новые информационные технологии в автоматизированных системах. – 2015. – №. 18.
23. Артемов М. А. Обзор систем анализа естественного текста на русском языке / М. А. Артемов, А. Н. Владимиров, К. Е. Селезнев // Вестник Воронежского государственного университета. Серия: Системный анализ и информационные технологии. – 2013. – №. 2. – С. 189-194.
24. Тутов Е. Б. Применение контекстно-свободных грамматик для обработки текстов на естественном языке / Е. Б. Тутов // Известия Юго-Западного государственного университета. Серия: Управление, вычислительная техника, информатика. Медицинское приборостроение. – 2012. – №. 2-1. – С. 97-101.
25. Мильчаков А. Е. Технология Томита Парсер / А. Е. Мильчаков // Информационные системы и технологии в образовании, науке и бизнесе (ИСиТ-2014). – 2014. – С. 226-226.
26. Сулейманов Р. С. Извлечение метаданных из полнотекстовых электронных русскоязычных изданий при помощи Томита-парсера / Р. С. Сулейманов // Программные продукты и системы. – 2016. – №. 4 (116).
27. Огородник Р. В. Обработка текста с помощью Томита-парсера. / Р. В. Огородник, Л. В. Серебряная // – 2014.
28. Бенгфорт Б. Прикладной анализ текстовых данных на Python / Б. Бенгфорт, Р. Билбро, Т. Охеда // Машинное обучение и создание приложений обработки естественного языка. СПб. Питер. – 2019.