МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра *Защиты информации*
(полное название кафедры)

Утверждаю

Зав. кафедрой *А. В. Иванов*

(подпись, инициалы, фамилия)

«___» _____ 202__ г.

ДИПЛОМНАЯ РАБОТА
НА ТЕМУ

Разработка приложения на основе графа знаний и генерации игрофицированных тестов для контроля осведомленности о политиках безопасности
(полное название темы)

Автор дипломной работы _____
(подпись студента, выполнившего дипломный проект или работу)

Лемякин А. В. Группа *АБс-723*
(фамилия, инициалы студента) (в которой обучался студент)

Факультет автоматики и вычислительной техники
(полное название факультета)

Специальность *10.05.03 «Информационная безопасность автоматизированных систем»*
(код и наименование специальности)

Руководитель работы _____ *Архипова А. Б.*
(подпись, дата) (фамилия, инициалы)

Новосибирск 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра *Защиты информации*
(полное название кафедры)

Утверждаю

Зав. кафедрой *А. В. Иванов*
(фамилия, имя, отчество)

_____
(подпись, дата)

ЗАДАНИЕ НА ДИПЛОМНУЮ РАБОТУ

студенту *Лемякину А. В.* группы *АБс-723*
(фамилия, инициалы)

1. Тема *Разработка приложения на основе графа знаний и генерации игрофицированных тестов для контроля осведомленности о политиках безопасности*
(полное название темы)

утверждена приказом по НГТУ № _____ от «____» _____ 202__ г.,
изменена приказом по НГТУ № _____ от «____» _____ 202__ г.

2. Дата представления работы к защите «____» _____ 202 _ г.

3. Цели работы (исходные данные): *создать программу принимающую на вход политику безопасности и выдающую автоматически сгенерированный тест с вопросами.*

4. Содержание работы: *Введение*

4.1. *Теоретические основы графов знаний. Основные понятия, стандарты представления данных, графовые БД, языки запросов, этапы создания графа знаний.*

4.2. *Базовые аспекты лингвистического анализа. Основные понятия, нейронные сети и машинное обучение, основные технологии машинного обучения.*

4.3. *Генерация вопросов на основе графов знаний. Основные понятия, классификации, оценка качества, примеры использования, проблемы генерации вопросов.*

4.4. *Введение в игрофикацию. Основные понятия, октализ, средства игрофикации, примеры использования игрофикации.*

4.5. *Техническая реализация. Схема технических функций программы, анализ и выбор средств разработки, извлечение текста из файлов, создание графа, клиент-серверная архитектура, пользовательский интерфейс, интеграция средств игрофикации.*

4.6. *Руководство пользователя. Введение, назначение и условия применения, подготовка к работе, описание операций.*

4.7. *Место работы в комплексной защите объекта информатизации.*

5. Перечень графического и (или) иллюстрационного материала *1. Теоретические основы графов знаний – 4 иллюстрации, 2. Базовые аспекты лингвистического анализа – 4 иллюстраций, 3. Генерация вопросов на основе графов знаний – нет иллюстраций, 4. Введение в игрофикацию – 1 иллюстрация, 5. Техническая реализация – 24 иллюстраций, 6. Руководство пользователя – 4 иллюстрации, 7. Место работы в комплексной защите объекта информатизации – нет иллюстраций.*

Руководитель работы _____ *Архипова А. Б.*

(подпись, дата) (фамилия, инициалы)

Задание принял к исполнению _____ *Лемякин А. В.*

(подпись студента, дата) (фамилия, инициалы студента)

Дипломная работа сдана в ГЭК № _____, тема сверена с данными приказа

_____

(подпись секретаря государственной экзаменационной комиссии по защите ВКР, дата)

_____

(фамилия, имя, отчество секретаря государственной экзаменационной комиссии по защите ВКР)

АННОТАЦИЯ

Пояснительная записка содержит 76 страниц текста, 37 рисунков, 8 таблиц, 45 использованных источников.

Дипломная работа на тему «Разработка приложения на основе графа знаний и генерации игрофицированных тестов для контроля осведомленности о политиках безопасности» выполнена студентом факультета Автоматики и вычислительной техники Новосибирского государственного технического университета Лемякиным Артёмом Викторовичем.

Место разработки – кафедра ЗИ, руководитель от НГТУ – Архипова Анастасия Борисовна, к.т.н., доцент кафедры ЗИ.

Цель работы: разработать приложение на основе графа знаний и генерации игрофицированных тестов для контроля осведомленности о политиках безопасности.

Ключевые слова: граф знаний, повышение осведомленности, политики безопасности, автоматическая генерация вопросов, NLP, контроль знаний, тесты, игрофикация, лингвистический анализ.

В первом разделе работы приводится теоретический материал о графах знаний.

Во втором разделе описываются теоретические данные о лингвистическом анализе.

В третьем разделе рассказывается про автоматическую генерацию вопросов.

В четвертом разделе рассматривается базовая теория касательно темы игрофикации.

В пятом разделе подробно описывается техническая реализация разработанного приложения, а также анализ и выбор средств разработки.

В шестом разделе дается краткое руководство пользователя.

В седьмом разделе описывается место работы в комплексной защите объекта информатизации. Описывается значимость разработанного приложения в рамках информационной безопасности.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ИБ – информационная безопасность;

NLP – natural language processing (обработка естественного языка);

NER – named-Entities Recognition (извлечение именованных сущностей);

RDF – resource description framework (структура описания ресурсов);

ОС – операционная система;

ФЗ – федеральный закон;

БД – база данных.

ВВЕДЕНИЕ

Самый опасный нарушитель в любой компании — это внутренний нарушитель. Так, например, по статистике компании «Infowatch» доля утечек данных по вине сотрудников за 2020 год составила 72% [1]. Для решения этой проблемы призван целый ряд мер. Так в статье [2] приводятся пять основных областей: повышение культуры ИБ организации и обучение персонала, ИТ-компетенция сотрудников, внедрение и соблюдение сотрудниками политик безопасности организации, управление рисками, менеджмент ИБ. Самыми эффективными были признаны: повышение культуры ИБ организации и обучение персонала, менеджмент ИБ. Обучение персонала также подразумевает тестирование. Это две взаимосвязанные вещи. Однако обычный контроль осведомленности о политиках безопасности имеет два значительных недостатка: ручной способ создания вопросов и невысокая мотивация сотрудников к самообучению.

Решением первой проблемы могла бы стать автоматическая генерация вопросов на основе графов знаний, способных выявить из анализируемого текста так называемые триплеты – картежи, состоящие из трех элементов: объект, субъект и предикат (отношение между объектом и субъектом). При этом в идеале необходимо соблюсти универсальность генератора для любых политик безопасности, чтобы любая организация могла без лишних усилий автоматически сгенерировать тесты на контроль знаний.

Для решения второй проблемы может подойти игрофикация тестов. Необходимо организовать процесс тестирования таким образом, чтобы он приносил удовольствие тем, кто его проходит. Нужно сформировать положительное отношение сотрудников к тестированию и самообучению. В идеале оно должно привести к добровольному стремлению повысить свою осведомленность о политиках безопасности.

В конечном итоге совокупность этих решений должна быть представлена в виде мобильного или десктопного приложения, в которое загружаются электронные документы политик безопасности, а на выходе формируются тесты в игровой форме.

Цель: разработать приложение на основе графа знаний и генерации игрофицированных тестов для контроля осведомленности о политиках безопасности.

Задачи, решение которых необходимо для достижения поставленной цели:

1. провести анализ и выбор программ, средств и методов для лингвистического анализа, построения графов знаний, генерации вопросов и игрофикации тестов;
2. реализовать извлечение текста из основных форматов (doc, docx, rtf, txt, pdf);
3. подготовить текст для лингвистического анализа;

4. выделить сущности и связи между ними в виде триплетов семантического графа;
5. из полученного графа сгенерировать вопросы и ответы;
6. создать тест и форму для его прохождения;
7. реализовать клиент-серверную архитектуру;
8. разработать адаптивный пользовательский интерфейс;
9. интегрировать в процесс прохождения тестов методы игрофикации.

1. Теоретические основы графов знаний

1.1 Понятие графа знаний

В области машинного обучения есть необходимость связать в единую коллекцию разрозненные, огромные объемы данных, генерируемых в интернете. Это необходимо, чтобы из большого объема данных автоматически получать полезную информацию. Для этой цели была разработана технология графов знаний, применение которой весьма широко. Вот некоторые примеры использования этой технологии [3]:

1. вывод концептуальных значений веб-запросов пользователя;
2. построение рекомендаций для пользователей или других приложений;
3. хранение структурированных знаний, которые поддерживают большое количество приложений, связанных с аналитикой больших данных.

Граф знаний – форма представления фактов посредством множества триплетов, имеющих вид вершин (сущностей) и ребр (отношений, предикатов) между ними. Термин введен компанией Google в 2012 году. Пример визуализации графа знаний приведен на рисунке 1.

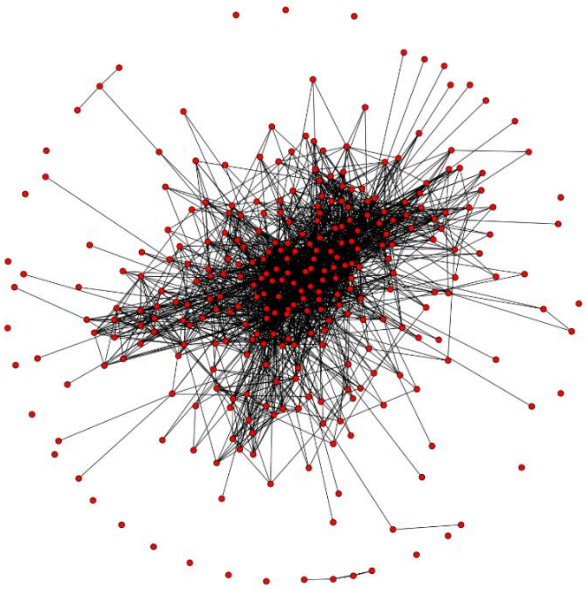

Рисунок 1 – Пример визуализации графа знаний

Граф знаний имеет ряд отличий от иных видов графов. Вот некоторые из них:

1) вершины и ребра имеют типы (классы),
2) ребра могут иметь разную значимость,
3) классы и отношения в графе имеют логический смысл.

1.2 Триплет. Его свойства и элементы структуры

Триплет – это структура из трех полей: <subject, predicate, object> где предикат – это семантическое отношение между субъектом и объектом. Пример условного триплета: <Петя> <ловит> <рыбу>, где Петя – субъект, ловит – предикат, рыбу – объект.

Элементы структуры триплета (субъект, объект и предикат) выражаются в виде URI, литералов и неименованных сущностей [3].

URI (Uniform Resource Identifier) — универсальный идентификатор ресурса. Это компактная последовательность символов, идентифицирующая абстрактный или физический ресурс. URI может быть представлен либо как URL, либо как URN [4]. Эта концепция введена глобальной информационной инициативой World Wide Web в 1990 году и описана в документе RFC 1630 (Request for Comments – серия информационных документов Интернета о технических спецификациях и стандартах, применяемых во всемирной сети).

URL — это URI, который, помимо идентификации ресурса, предоставляет ещё и информацию о местонахождении этого ресурса [4].

URN — это URI, который только идентифицирует ресурс в определённом пространстве имён, но не указывает его местонахождение [4]. Например, URN: ISBN:0-395-36341-1 — это URI, который указывает на ресурс (книгу) 0-395-36341-1 в пространстве имён ISBN.

Литерал – то, чему равна некая константа. Может иметь вид числа, текста, логического (bool) значения.

Неименованные вершины (Blank Notes) – сущности без URI или без литерала. Используются для логических аксиом и для реификации. Т.е. это вершины, буквально означающие «какая-то сущность».

С учетом вышеописанных терминов свойства триплета можно описать следующим образом:

1) субъектом может быть или URI, или неименованная сущность,
2) предикатом может быть только URI,
3) объектом может быть URI, неименованная сущность или литерал.

1.3 Стандарты представления данных

Данные (триплеты) в графах знаний имеют свою структуру, которая определяется стандартом представления данных RDF (Resource Description Framework). Пример такого RDF-триплета приведен на рисунке 2.

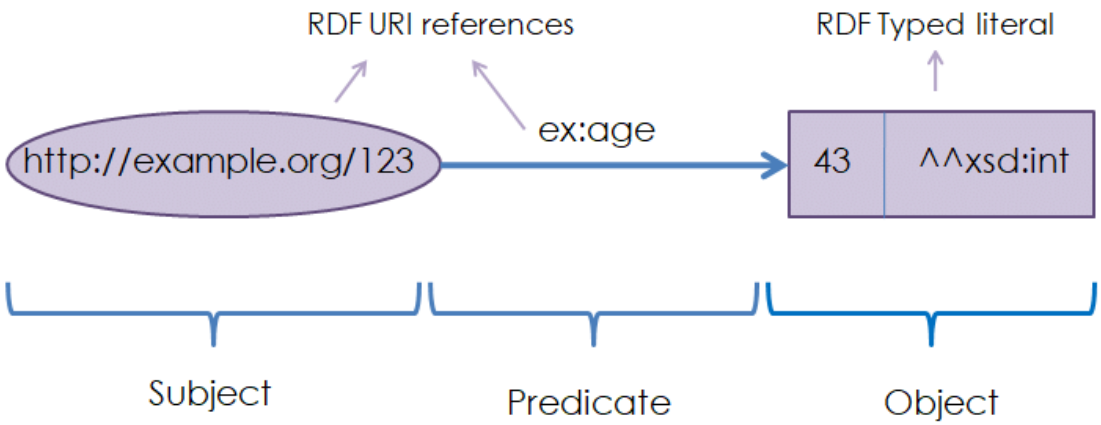

Рисунок 2 – Пример RDF триплета

В текстовом же варианте RDF-триплет имеет стандартизированную запись. Пример: <ex:NSTU_University> <ex:founding_date > <”19.08.1950”>.

Моделей представления данных в графах знаний бывает несколько: RDF, RDFS, OWL и онтологии. Все они при этом являются надстройками друг на другом и имеют свои особенности. Рассмотрим каждую из них [3].

1. **RDF** – модель представления данных, которая приводит некие утверждения о ресурсах к виду, пригодному для машинной обработки. Ресурсом в RDF может быть любая сущность – как информационная (например, веб-сайт или изображение), так и неинформационная (например, человек, город или некое абстрактное понятие). Утверждение же о ресурсе имеет вид триплета.
2. **RDFS** (RDF Schema) – базовое расширение RDF, включающее возможность объявлять классы (rdfs:class) и их экземпляры (rdf:type); объявлять предикаты, их область определения и область значений (rdf:Property, rdfs:domain, rdfs:range); создавать иерархии классов (rdfs:subClassOf) и иерархию предикатов (rdfs:subPropertyOf); создавать аннотации: человекочитаемое имя ресурса (rdfs:;label), текстовый комментарий (rdfs:comment), ссылка на объясняющий ресурс (rdfs:seeAlso).
3. **OWL** (Web Ontology Language) – ещё более выразительное надмножество (расширение) RDFS. Основан на дескрипционной логике. Эта модель данных:

представляет реальность в виде «объект – свойство». Позволяет описывать классы и отношения между ними, присущие веб-документам и приложениям.

4. **Онтология** – это обобщенная модель данных, представляющая только общие типы вещей, которые имеют определенные свойства, но не включают персонифицирующая информацию о конкретном объекте. Это позволяет использовать онтологию повторно для описания других аналогичных объектов. Если применить онтологию к набору отдельных элементов данных (информация о книге, авторе, издателе), то получится уже граф знаний. Таким образом: ontology + data = knowledge graph. Это не самостоятельная модель данных, т.к. для её формирования используются RDF, RDFS, OWL [5].

Стандартом RDF также определен ряд базовых предикатов, которые используются в языках запросов к графовым базам данных. Примеры самых часто используемых базовых предикатов [6]:

1) rdf:type – указание того, что ресурс является экземпляром класса;
2) rdf:property – это класс свойств RDF;
3) rdf:subject – определяет предмет утверждения;
4) rdf:predicate – определяет предикат утверждения;
5) rdf:object – определяет объект утверждения (одна из составляющих триплета);
6) rdf:first – указывает первый элемента списка;
7) rdf:rest – указывает подсписок, который содержит элементы списка, отличные от первого;
8) rdf:value – используется при описании структурированных значений;
9) rdf:nil – пустой лист;
10) rdf:list – это класс списков RDF.

1.4 Графовые базы данных и языки запросов

Графовые базы данных (графовые БД) – используются для хранения данных графа знаний и логики, описывающей взаимосвязи и контекст. Поддерживают запросы управления и извлечения данных.

Преимущества графовых БД [7]:

1. **Универсальность**. Возможность хранить и реляционные, и документарные и сложные семантические данные.

2. **Производительность**. В отличие от реляционных баз данных, где учет взаимосвязей интенсивно ухудшает производительность запросов на больших наборах данных, производительность графовых баз данных остается неизменной с увеличением объема хранимых данных. Это связано с тем, что запросы локализуются в определенной части графа. В результате время выполнения каждого запроса зависит от размера части графа, которую требуется обойти для удовлетворения запроса, а не от общего размера графа.
3. **Гибкость**. Можно добавлять новые виды взаимосвязей, новые узлы, новые метки и новые подграфы в существующую структуру, не нарушив при этом существующих запросов и функционала приложения. Благодаря этому нет необходимости планировать задачу в мельчайших деталях.

Типы графовых БД [8]:

1. **Property graph** – данные организованы в виде узлов, связей и свойств (данные, хранящиеся на узлах или связях). Одно ребро может соединять только две вершины. Пример property graph изображен на рисунке 3.

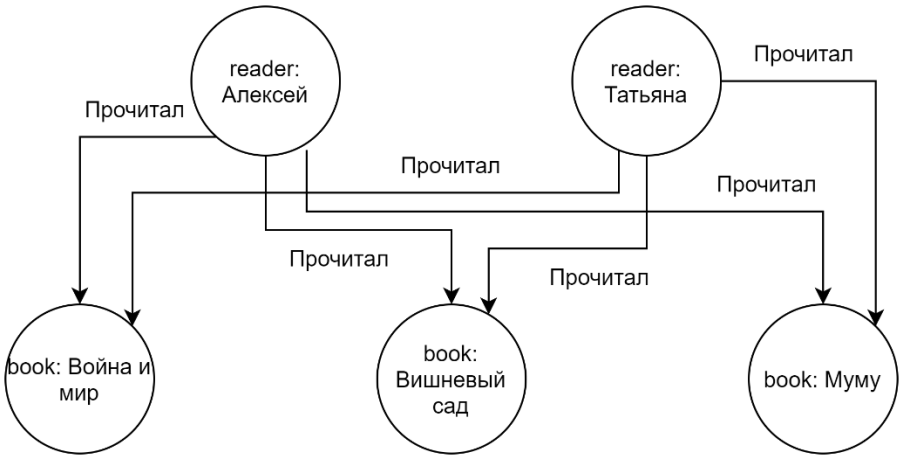

Рисунок 3 – Пример property graph

2. **Hypergraph** – расширение обычной концепции графа, позволяющей ребру графа иметь более 2-х вершин. Такое ребро называют гиперребром. Это полезно, когда данные содержат большое количество взаимосвязей "многие ко многим". Пример hypergraph изображен на рисунке 4.

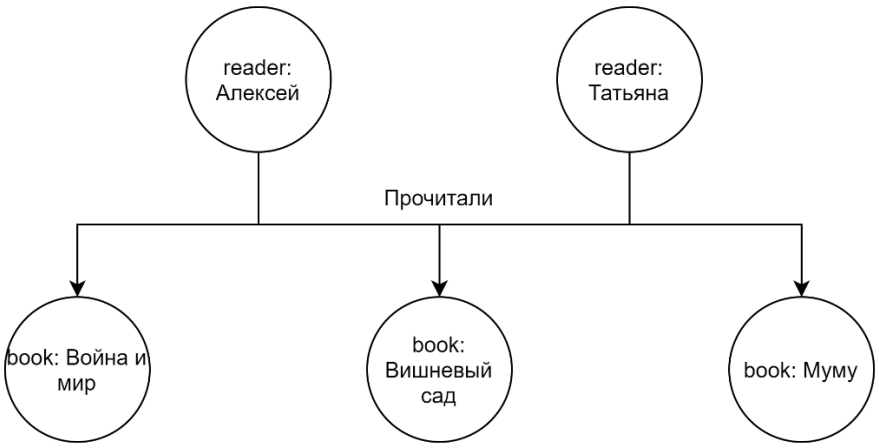

Рисунок 4 – Пример hypergraph

3. **Triple store** – не является самостоятельной графовой базой данных, т.к. не поддерживает смежность без индексов и его механизмы хранения не оптимизированы для хранения property graph. Такая система хранит триплеты как независимые элементы, что позволяет им масштабироваться по горизонтали, но не позволяет им быстро пересекать взаимосвязи. Чтобы выполнять графовые запросы, triple store должен создавать соединения из отдельных независимых фактов, что увеличивает задержку для каждого запроса.

Для обработки (извлечение и управление) RDF-данных используются декларативные языки запросов.

Декларативный язык – язык программирования высокого уровня, в котором программистом не задается пошаговый алгоритм решения задачи ("как" решить задачу), а некоторым образом описывается, "что" требуется получить в качестве результата. В противовес декларативным существуют императивные языки программирования.

Императивный язык – язык, в котором записываются инструкции, которые должны выполняться последовательно, т.е. это запись того «как» решить задачу.

Наиболее распространенными языками запросов к графовым БД являются: SPARQL, Cypher, Gremlin.

SPARQL (Protocol and RDF Query Language) – язык запросов к данным, представленным по модели RDF, а также протокол для передачи этих запросов и ответов на них. SPARQL является рекомендацией консорциума W3C и одной из технологий семантической паутины. Поддерживаемые графовые базы данных: Amazon Neptune, Apache Marmotta, AllegroGraph, Eclipse RDF4J, Apache Jena with ARQ, Blazegraph, Cray Urika-GD, IBM

Db2 – Removed in v11.5, KAON2, MarkLogic, Mulgara, NitrosBase, Ontotext GraphDB, Oracle DB Enterprise Spatial & Graph, RDFLib Python library, Redland / Redstore, Virtuoso [9].

Cypher – декларативный язык запросов к графам, который позволяет выполнять выразительные и эффективные запросы к данным в графе свойств. Поддерживаемые графовые базы данных: Janus Graph, InfiniteGraph, Cosmos DB, DataStax Enterprise(5.0+) и Amazon Neptune [10].

Gremlin – это язык обхода графов, разработанный Apache TinkerPop и принятый многими решениями для графовых баз данных. Оно может быть как декларативным, так и императивным. Поддерживаемые графовые базы данных: Neo4j, AgensGraph и RedisGraph [10].

1.5 Создание графа знаний

Граф знаний возможно создать как из структурированных, так и неструктурированных данных.

Структурированные данные – данные, отражающие отдельные факты предметной области и упорядоченные определенным образом с целью обеспечения возможности применения к ним различных методов обработки. Большинство алгоритмов машинного обучения, статистического и интеллектуального анализа данных работают только со структурированными данными.

Неструктурированные данные – данные, которые не соответствуют заранее определённой модели представления и, как правило, представлены в форме текста с датами, цифрами, фактами, расположенными в нём в произвольной форме.

Для создания графа знаний из неструктурированных данных необходимо выполнение следующих шагов [11]:

1. извлечение полезной информации из различных источников (текст на веб-страницах, аудио, видео, книги, документы);
2. определение сущностей и отношений между ними с использованием обработки естественного языка;
3. интеграция полученных данных;
4. анализ структуры графа, прогнозирования ссылок и использование методов встраивания, основанных на машинном обучении и методах глубокого обучения.

Рассмотрим каждый этап подробнее.

1.5.1 Этап 1. Извлечение сущностей

Извлечение сущностей или NER (named-entities recognition) – это обнаружение и классификация информационных элементов из неструктурированных данных.

Для выполнения данной задачи используются различные методы:

1. **тегировщик фрагментов** – метод основан на скрытой Марковской модели;
2. **KNN** (K-nearest neighbors) с линейной моделью CRF – простой контролируемый алгоритм машинного обучения, который можно использовать для решения задач как классификации, так и регрессии. Он прост в реализации и понимании, но его производительность сильно снижается по мере роста размера используемых данных;
3. **TJE** (Transfer Joint Embedding) **–** метод на основе трансферного обучения, для междоменной классификации нескольких классов [12];
4. **NELL** (Never-Ending Language Learning) – идентифицирует базовый набор фундаментальных семантических связей между несколькими сотнями предопределенных категорий данных, таких как города, компании, эмоции и спортивные команды.

Проблема контекста. Одна сущность может иметь разное значение в разных источниках. Необходимо соотнести такие текстовые упоминания сущностей с их узлами в графе. Например, слово «Тесла» может ассоциироваться с учёным Николой Тесла и в то же время оно может означать название компании Илона Маска «Тесла».

Решениями проблемы контекста являются следующие модели [11]:

1. **модель Хоффарта** – данная модель базируется на использовании нейронной сети для получения представлений сущностей и упоминания контекста для применения к ним привязки сущностей;
2. **модель Ямады** – совместно отображает слова и сущности в одно и то же непрерывное векторное пространство и применяет вложения для изучения функций для EL;
3. **DSMM** (Deep Semantic Match Model) – модель глубокого семантического соответствия. Помогает сопоставить текстовое упоминание с референтным объектом в графе знаний, применяя двунаправленную сеть долговременной кратковременной памяти (BiLSTM) с мультигранулярностью. Превосходит все предыдущие современные модели.

1.5.2 Этап 2. Извлечение отношений

Методы установления отношений между сущностями [11]:

1. Метод основанный на признаках (основан на методах NLP - тегирование частей речи, анализ синтаксиса, распознавание именованных объектов). Существует несколько наборов признаков:
    - словесные признаки:
        - заголовки сущностей,
        - слова или биграммы слева,
        - количество слов, разделяющих две сущности и т.д.
    - сущностные признаки:
        - типы именованных сущностей (например, лицо, местонахождение)
        - их объединение,
        - уровни упоминаний (например, имя, нарицательное или местоимение).
    - функции анализа:
        - последовательность синтаксических фрагментов,
        - путь между двумя сущностями в дереве синтаксического анализа.
2. Методы основанные на «зерне». Они требуют значительного объема данных, которые помечаются человеком, что требует огромного количества времени и сил для крупномасштабного извлечения веб-отношений.
3. Методы на основе текстовых шаблонов.
4. Модели для изучения более глубоких семантических признаков, такие как PCNN и CNN с объединением внимания, граф LSTM и модель LFDS.
5. Методы основанные на word embeddings. Встраивание слов может быть определено методами NLP, где слова или фразы представляются как векторы в низкоразмерном пространстве. Обычно для создания таких карт используются глубокое обучение, тематическое моделирование, матричная факторизация и другие методы.

Вышеописанные методы позволяют выделить необходимые для графа отношения, однако, если ограничиться лишь ими, то возникнет так называемая проблема кореферентности.

Кореферентность (Coreference) – это задача поиска всех выражений, которые ссылаются на одну и ту же сущность в тексте. Пример: «Доктор сказал, что он не против предложения, ему оно вполне подходит» – слова «он», «ему» в данном контексте явно

ссылаются на одну и ту же сущность «доктор». В таком случае ссылающиеся слова называют референтными выражениями, а объект ссылок референтом (или сущностью). Два или более референтных выражения, используемых для обозначения одного объекта, называются кореферентными [13].

Так, благодаря кореферентности, например, диалоговая система, сообщившая: «В Екатеринбург есть рейс в 12:00 и в 14:00», будет знать, какой именно рейс пользователь имеет в виду, говоря: «Я возьму второй».

Инструменты для решения этой задачи [11]:

1. Stanford Core NLP - позволяет пользователям создавать лингвистические аннотации к тексту, включая границы лексем и предложений, части речи, именованные объекты, числовые и временные значения, синтаксический анализ зависимостей и составляющих, корреляцию, настроение, атрибуцию цитат и отношения.
2. AllenNLP - исследовательская библиотека Apache 2.0 NLP, построенная на PyTorch, для разработки современных моделей глубокого обучения для широкого спектра лингвистических задач.

1.5.3 Этап 3. Интеграция полученных данных

Интеграция триплетов (полученных данных) происходит с использованием выходных данных трех компонентов: Entity Mapping, Coreference и непостредственно Triple Extraction.

Компонент Triple Extraction формирует триплеты из отношений и сущностей, но не учитывает кореферентность. Это приводит к неоднозначности и отсутствию взаимосвязей между сущностями в графе знаний. Именно поэтому необходима интеграция с выходными результатами ещё двух компонентов, перечисленных выше. Как это происходит:

1. идентичные сущности группируются с использованием кореферентных цепочек;
2. с помощью алгоритма голосования выбирается представитель группы кореферентных сущностей;
3. все сущности, принадлежащие группе в отношении триплета, заменяются представителем своей группы;
4. отношение этого триплета напрямую преобразуется в предикат путем присвоения нового URI;
5. если объект триплета отношений не является сущностью, он остается литералом;
6. предфинальное формирование триплетов.

Рассмотрим пример предфинального создания триплета. Имеется триплет <DBpedia: ИлонМаск, ex: владелец, DBpedia: Тесла>, в котором предикат «ex: владелец» несопоставлен ни с одним из предикатов графа знаний. Если сопоставить его с отношением в пространстве имен DBpedia (например, Property), тогда триплет примет окончательный вид: <DBpedia: ИлонМаск, ex: Property, DBpedia: Тесла>.

1.5.4 Этап 4. Прогнозирование связей

Существует проблема поиска недостающих отношений в графе. Она включает в себя задачу прогнозирования сущностей и задачу прогнозирования отношений.

Методы для заполнения графа знаний недостающими сведениями [11]:

1. **Внедрение состязательной сети** – она состоит из двух компонентов: компонента сохранения структуры и компонента состязательного обучения. Первый направлен на получение структурных свойств сети, а второй способствует изучению надежных представлений путем сопоставления апостериорного распределения скрытых представлений с заданными априорными данными.
2. **struc2vec**. Это фреймворк, который используется для изучения скрытых представлений структурной идентичности узлов (когда узлы сети идентифицируются в соответствии со структурой сети и их отношением к другим узлам). struc2vec использует иерархию для измерения сходства узлов в различных масштабах и строит многослойный граф для кодирования структурных сходств и создания структурного контекста для узлов. В отличии от DeepWalk и node2vec данная модель способна уловить понятие структурной идентичности. Также было показано, что struc2vec лучше справляется с задачей классификации, где метки узлов больше зависят от структурной идентичности (т. е. сети воздушных путей с метками, представляющими деятельность аэропорта).
3. **TransE** – модель, которая непрерывно учится низкоразмерному встраиванию сущностей и отношений.
4. **LINE** – метод оптимизирует тщательно разработанную целевую функцию, которая сохраняет как локальную, так и глобальную сетевые структуры. Предлагается алгоритм выборки краев, который устраняет ограничения классического стохастического градиентного спуска и повышает как эффективность, так и результативность вывода. Эмпирические эксперименты доказывают эффективность LINE в различных реальных информационных сетях, включая языковые сети,

социальные сети и сети цитирования. Алгоритм очень эффективен, он может научиться встраиванию сети с миллионами вершин и миллиардами ребер за несколько часов на типичной одиночной машине. Эмпирические эксперименты доказывают эффективность LINE в различных реальных информационных сетях, включая языковые сети, социальные сети и сети цитирования. Он может изучить вложение сети с миллионами вершин и миллиардами ребер за несколько часов даже на не очень мощной машине. В отличие от большинства существующих моделей неконтролируемого обучения на графоструктурированных данных и прогнозирования ссылок, она может естественным образом включать функции узлов, что значительно повышает производительность прогнозирования на нескольких эталонных наборах данных.

5. **node2vec** – алгоритмическая структура для изучения непрерывных представлений признаков для узлов в сетях. Изучается отображение узлов в низкоразмерное пространство признаков, которое максимизирует вероятность сохранения сетевых окрестностей узлов. Определяется гибкое понятие окрестности сети анода и разрабатывается процедура случайного блуждания со смещением, которая эффективно исследует различные окрестности. Предлагаемый алгоритм обобщает предыдущую работу, основанную на жестких представлениях о сетевых окрестностях. Он обладает дополнительной гибкостью при изучении окрестностей является ключом к изучению более богатых представлений.
6. **Logical queries** – в данном методе узлы графа были встроены в низкоразмерное пространство и представляют логические операторы как изученные геометрические операции (например, перемещение, вращение) в это пространство вложения. Путем выполнения логических операций в низкоразмерном пространстве вложений была достигнута временная сложность, линейная по количеству переменных запроса, по сравнению с экспоненциальной сложностью, требуемой нативным подходом, основанным на перечислении. Конъюнктивные запросы, соответствующие подмножеству логики первого порядка, использующие только операторы конъюнкции и квантификации существования. Конъюнктивные запросы позволяют рассуждать о существовании отношений подграфов между наборами узлов, что делает конъюнктивные запросы естественным фокусом для приложений графов знаний.) в этом пространстве вложения. После обучения мы можем использовать модель, чтобы предсказать, какие узлы, скорее всего, удовлетворят любому

правильному конъюнктивному запросу, даже если запрос включает в себя несущствующие ребра.

7. **Autoencoder** – эта модель полагается на автоэнкодер для изучения нелинейных вложений узлов из локальных окрестностей графа. Модель автоэнкодера нацелена на изучение набора низкоразмерных скрытых переменных для узлов, которые могут производить приблизительные выходные данные реконструкции, так что ошибка между матрицей смежности и выходными данными минимизируется, тем самым сохраняя глобальную структуру графа.
8. **DeepWalk** – в данном методе вместо смешивания пространства меток как части пространства признаков, происходит изучение признаков, фиксирующих структуру графа, независимую от распределения меток.
9. **Learning Structural Node Embeddings via Diffusion Wavelets** – этот метод обеспечивает математические гарантии оптимальности изученных структурных вложений. Используя спектральную теорию графов, структурно эквивалентные (или похожие) узлы имеют почти идентичные (или похожие) вложения в GraphWave. Различные эксперименты с реальными и синтетическими сетями предоставляют эмпирические доказательства аналитических результатов и дают существенный прирост производительности по сравнению с современными базовыми уровнями.

2. Базовые аспекты лингвистического анализа

2.1 Понятие лингвистического анализа и его этапы

В контексте построения графов знаний из неструктурированных данных ключевым элементом является лингвистический анализ, позволяющий выделить сущности и отношения между ними.

Лингвистический анализ (NLP, Natural Language Processing) – обработка естественного языка. Подраздел искусственного интеллекта и математической лингвистики, изучающий проблемы компьютерного анализа и синтеза текстов на естественных (человеческих) языках.

NLP подразделяется на четыре последовательных шага [14]:

1. **Графематический анализ** – выделяет элементы структуры текста. Для этого происходит учет числа символов, определяющих новый абзац, и отношения количества каждой последовательности в буфере к его размеру. Такой анализ позволяет разделить текст на графы, выделить слова и разделители, выделить заголовки, абзацы и примечания, а также найти границы предложений и определить устойчивые обороты или же сокращения.
2. **Морфологический анализ** – определить морфологический характер слова и словоформы. Выделяет леммы на основе заданного токена или морфологических параметров;
3. **Синтаксический анализ** – определение синтаксических зависимости слов в приложении. Данный анализ позволяет узнать, что предложение корректно и позволяет сформировать структуру (синтаксический граф), наглядно показывающую какие есть между словами синтаксические отношения. Пример данного этапа виден на рисунке 5.

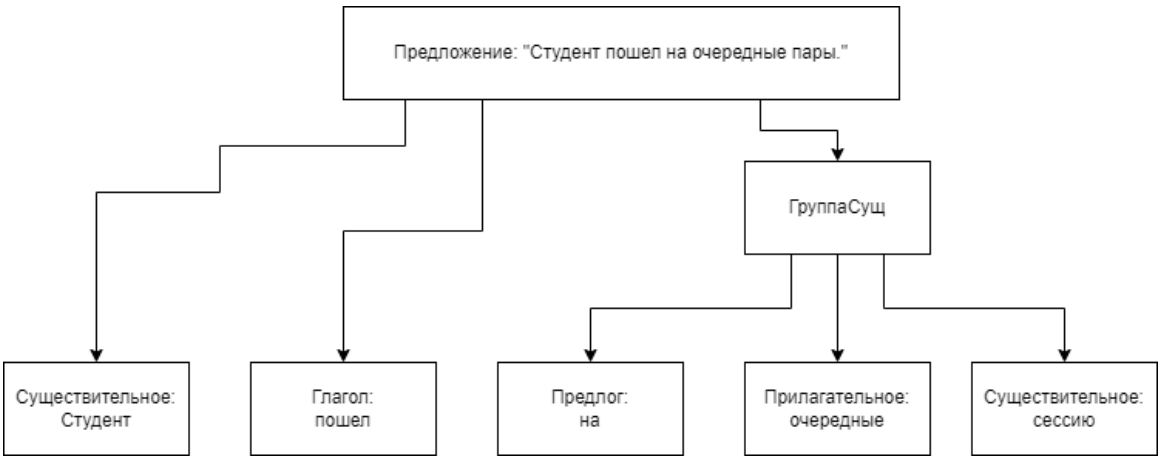

Рисунок 5 – Пример результата синтаксического анализа

4. **Семантический анализ** – находит связи, которые не смог обнаружить синтаксический анализ, а также устраняет из обработки слова, не имеющие «семантическую связность». Результатом такого анализа является семантический граф, отражающий виды отношений между узлами (смысловыми единицами текста). Примеры отношений: Agent (кто действует), Pacient (на кого действуют), Anafor (анафорическая ссылка), Time (время) и др. Такой граф напоминает граф знаний с его триплетами, но в отличии от него он не демонстрирует информацию о фактах, содержащихся в тексте. При этом семантический граф является основой для формирования графа знаний. Более наглядно различия между семантическим графом и графом знаний видны по рисункам 6 и 7.

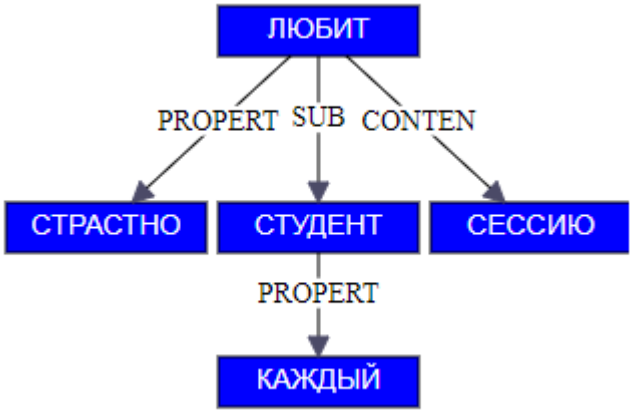

Рисунок 6 – Пример семантического графа, сформированного из предложения «Каждый студент страстно любит сессию»

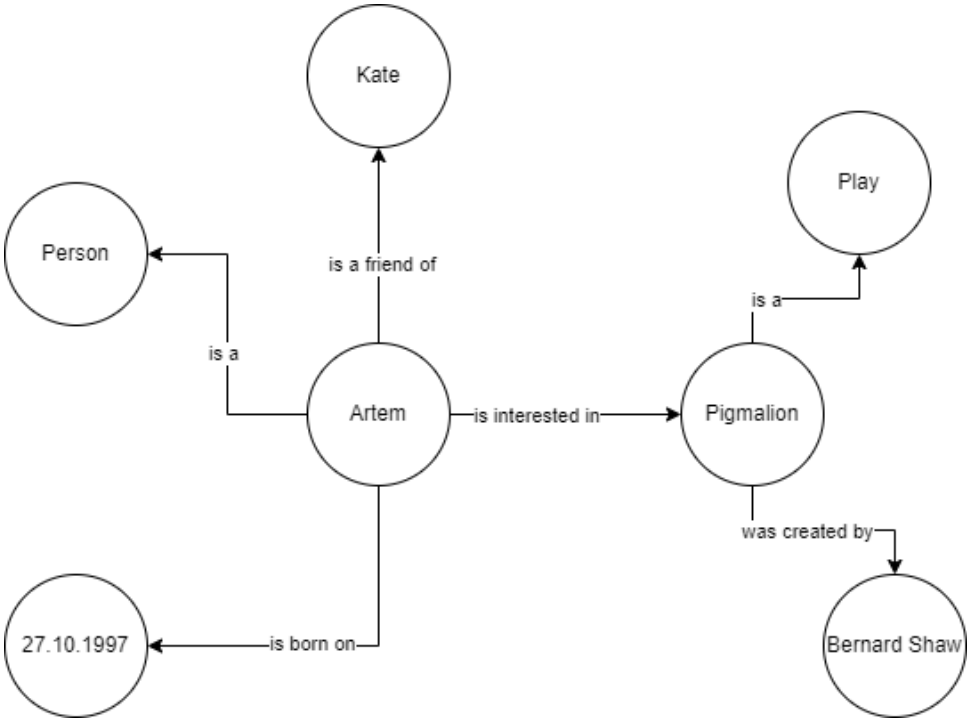

Рисунок 7 – Пример графа знаний

Такой подход анализа в 4 этапа имеет недостаток: одни и те же получаемые результаты каждого шага анализа могут быть получены из разной входной информации, а одна и та же входная информация может дать разные результаты.

В статье [15] предлагается пересмотр традиционного анализа в пользу двух этапного процесса:

1. общение сканирование, дающее информацию лишь о ключевых частях текста;
2. подробный анализ, выявляющий уточняющие данные через обращение к вышерасположенных модулей к нижерасположенным.

2.2 Основные термины NLP

Понятия, используемые в морфологическом анализе [16, 17]:

Грамматические категории (параметры) – множество из взаимоисключающих друг друга граммем, характеризующих какой-то общий признак (например, род, время, падеж).

Граммемы – конкретные значения одной из грамматических категорий слова (например, прошедшее время, единственное число, мужской род).

Токенизация (иногда – сегментация) – процесс разделения последовательности слов на более мелкие сегменты. Например, разбитие предложения на слова.

Токен (англ. Token – знак) – объект, создающийся из лексемы в процессе лексического анализа (токенизации). Иными словами, это слово, отделено от других пребелом или иным знаком препинания.

Метатокен – являет собой более крупную конструкцию, сущность. Формируется посредством объединения токенов. Пример: «яркое солнце», «большой медведь».

Нормализация (Лемматизация) – приведение слова к нормальной форме (**лемме**), т.е. к начальной форме. Например, приведение слова «молекул» к форме «молекула».

Лексема – набор всех форм одного слова.

Словоформа – совокупность из токена, леммы и грамматических параметров.

Понятия, используемые в семантическом анализе:

NER (Named-Entities Recognition) – поиск и классификация информационных элементов из текста. Например, это могут быть имена людей или названия компаний, названия географических объектов (города, реки, улицы), даты, страны, указы, url ссылки, номера телефонов и т.д. Конкретный состав извлекаемых категорий именованных сущностей зависит от частной реализации.

Отношения (Relationships) – лексическая функция, выражающая смысловую связь между словами.

Word Embeddings – представление слов для анализа текста в форме вектора с действительным знаком, который кодирует значение слова таким образом, что слова, находящиеся ближе в пространстве вектора, как ожидается, будут схожими по смыслу. На рисунке 8 наглядно демонстрируется данный принцип [18].

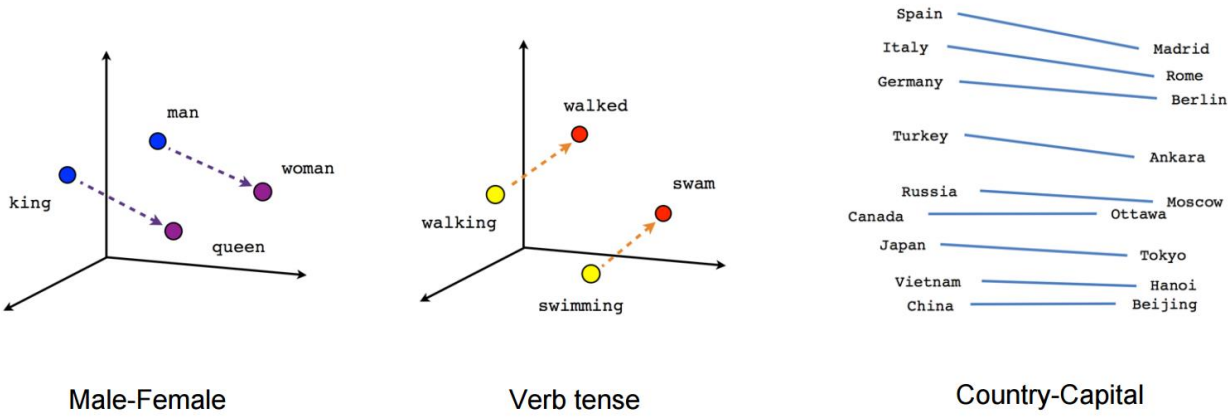

Рисунок 8 – Принцип word embeddings

2.3 Нейронные сети и машинное обучение

Есть две метода реализации NLP [15]: инженерный и на базе машинного обучения.

Инженерный, основанный на правилах (rule-based) – суть метода в создании набора формальных правил, выражающих лингвистическую информацию. Такие правила составляются лингвистами или же специалистами в определенной проблемной области.

Недостатки:

- большие затраты ручного труда на создание правил,
- необходимость в высокой квалификации составителя правил.

Достоинство: легкость изменения и расширения составленных правил в виду их декларативности и просты в понимании.

Метод, основанный на машинном обучении (machine learning) – происходит отбор текстов проблемной сферы, которые будут служить источником информации. Существует следующие вариации данного метода:

- обучения с учителем (supervised),
- методы обучения без учителя (unsupervised),
- методы частичного обучения с учителем (bootstrapping).

Недостатки:

- необходимо наличие заранее размеченного корпуса текстов,
- отсутствие явной лингвистической интерпретации у результирующих моделей.

Достоинства:

- минимизация затрат времени на разработку системы,
- отсутствия ручного труда.

Помимо вышеописанных подходов существуют гибридные системы лингвистического анализа, которые реализуют разные этапы анализа разными методами, например, графематический анализ делается на базе машинного обучения, а морфологический анализ – на базе правил.

Также имеются системы, использующие признаковую модель текста. В такой модели признаками могут являться следующие характеристики текста:

- статистические (повторяемость букв, биграмм, m-грамм и др.);
- структурные (соотношение слов с точки зрения частей речи, присутствие конкретных синтаксических конструкций, разделов текста и др.).

2.4 Основные технологии машинного обучения

word2vec. Данный алгоритм определяет смысл слова через часто встречающийся контекст его употребления. Таким образом, смысл слова определяется по его соседям или наоборот, определяются соседи слова. При этом данный алгоритм способен без специальной разметки обучаться на любом множестве текстов. Помимо определение семантически близких слов, этот алгоритм позволяет обнаружить опечатки и произвести оценку важности слов в составленном пользователе запросе [19].

Рекуррентные нейронные сети. Они умеют создавать языковые модели (распределение вероятностей следующего, за неоконченной частью предложения, слова).

Принцип действия рекуррентной нейронной сети [19]:

1. Сеть обрабатывает предложение.
2. На каждом шаге сеть формирует скрытое представление в виде вектора, содержащего закодированную информацию об уже обработанной части текста.
3. Благодаря скрытому представлению делается предсказание следующего слова.
4. Далее сеть принимает на вход текущее слово и предыдущее скрытое представление, после чего вычисляет на их основе следующее скрытое представление, предсказывает слово, и т.д.

Существуют две самые эффективные функции, вычисляющие последующее скрытое представление: GRU (gated recurrent unit) и LSTM (long short-term memory). Их особенность в том, что они делают сложные преобразования входных векторов, позволяющие сохранить информацию, которая была выявлена в тексте множество шагов назад. Это позволяет следить за длинным контекстом предложения.

Механизм «внимания» – этот механизм составляет векторное представление всего предложения как сумму векторных представлений отдельных слов. Это значительно улучшает качество алгоритмов ответов на вопросы и машинного перевода [19].

Латентно-семантический анализ текста (Latent semantic analysis, LSA). Представляет документы и отдельно взятые слова в «семантическое пространство». Алгоритм вычисляет частоту встречаемости слов в документе. При этом порядок слов игнорируется, а семантический смысл документа устанавливается посредством составления набора слов, употребляемых вместе [15].

TF/IDF (term frequency – частотность вхождения термина, inverse document frequency – инвертированная частота документа). Это отношение, выражающее весомость слова в тексте всего документа, который является частью коллекции документов. TF отражает важность слова в рассматриваемом документе. IDF обратное значение частоты встречаемости слова в коллекции документов (позволяет снизить весомость предлогов, союзов, общих терминов и понятий) [15].

3. Генерация вопросов на основе графов

3.1 Основные понятия

В общем смысле генерация вопросов является задачей синтеза текста, на ряду с задачей анализа текста. Анализ может быть представлен следующими этапами [20]:

1) считывание текстовой информации из входного файла;
2) выделение отдельных слов, реализуемое любым из множества доступных способов, например, отделением слов от пробела до пробела и добавлением в массив;
3) определение программой принадлежности части речи для каждого слова;
4) подготовка семантических моделей, представленных в виде графов, где узлами являются слова, а ребрами их отношения.

Синтез текста (вопросов) может быть представлен комбинацией двух способов из ниже приведенных классификаций (по методам генерации и по структуре исходных данных).

3.2 Классификация способов генерации вопросов

По методам генерации вопросов [21]:

1. **Логические методы (LOGIC)** – методы, основанные на представлении функций и моделей в виде заранее заданных формул, на основании которых задача генерации вопросов сводится к применению этих формул для преобразования исходных данных.
2. **Методы на основе машинного обучения (ML)** – методы, способные обучиться по прецедентам, т.е. сделать общие выводы на основе частных наблюдений.
3. **Гибридные методы (HM)** – методы, сочетающие в себе как логические методы, так и методы на основе машинного обучения.

По структуре исходных данных [21]:

1. **Использование текстовых данных (TEXT)** – методы, которые используют входные данные в виде текста на естественном языке.
2. **Использование графовых данные (GRAPH)** – методы, входные данные которых представляют собой знания, использующие графовое описание. Примером является формат представления знаний RDF.

3. **Использование гибридных данные (HD)** – методы, для которых входные данные подаются в гибридном представлении описанных выше структур исходных данных. Примером такого представления может быть текст на естественном языке, дополненный онтологией в формате RDF, которая описывает предметную область текста.

В работе [22] для интеллектуальной обучающейся системы (ИОС) используется технология OSTIS (Open Semantic Technology for Intelligent Systems) и автоматическая генерация субъективных и объективных вопросов. Генерация вопросов происходит логическими методами, которые реализуются аппаратом дискретной математики. Относительного этого подхода можно выделить следующие виды стратегии генерации вопросов [22].

1. На основе элементов:
    - на основе ролевого отношения – то, которое задает роль элементов в рамках некоторого множества;
    - на основе бинарного отношения – множество отношений на множестве M, являющихся подмножеством декартова произведения множества M на самого себя.
2. На основе классов:
    - отношения «включение», когда некоторые классы в базе знаний содержат несколько подклассов. Пример: «Частным случаем бинарного дерева является: вариант 1, вариант 2, и т. д.»;
    - отношения «разбиение», когда в результате разбиения множества получается множество попарно непересекающихся множеств, объединение которых есть исходное множество;
    - отношения «строгое включение» – частный случай отношения включения.
3. На основе идентификаторов, когда некоторые сущности в базе знаний могут иметь несколько идентификаторов (синонимов). Пример: «Ориентированное множество также называется __?».
4. На основе аксиом.
5. На основе свойств отношений – использование свойств рефлексивности, симметричности и транзитивности, которым могут соответствовать некоторые отношения в базе знаний. Пример: «Если A = B, и при этом B = C, то равны ли A и C?».

6. На основе примеров изображений, когда в базе знаний хранятся некоторые понятия, отношения, теоремы и поясняющие их изображения.
7. Генерация субъективных вопросов – вопросы на определение понятий и доказательства теорем.

Вопросы, автоматически генерируемые с использованием перечисленных выше стратегий, могут содержать повторные и неправильные вопросы, поэтому для обеспечения качества генерируемых вопросов необходимо сначала сохранить эти автоматически генерируемые вопросы в базе знаний подсистемы автоматической генерации вопросов, а затем использовать ручные или автоматические подходы (сравнение подобия между вопросами) для фильтрации повторных и неправильных вопросов [22].

3.3 Оценка качества сгенерированных вопросов

Существует два принципиальных способа оценки [23]: субъективные и машинный.

Первый может быть реализован с помощью пятибалльной бальной шкалы Макото Нагао, в которой количество баллов означает следующее:

1 балл – смысл предложения понятен и не возникает никаких вопросов, грамматика, словоупотребление и стиль соответствуют общей структуре текста и не требуют постредактирования;

2 балла – смысл предложения понятен, но возникают большие проблемы с грамматикой, словоупотреблением и стилем;

3 бала – общий смысл предложения понятен, но смысл некоторых его частей вызывает сомнение из-за неправильного грамматического строя;

4 балла – присутствуют ошибки словоупотребления и стилистики, требуется обращение к оригиналу;

5 баллов – в предложении имеется большое количество грамматических, словоупотребительных и стилистических ошибок, смысл предложения с трудом можно понять после внимательного изучения.

Второй способ может быть реализован с помощью автоматической системы оценки на основе метода N-грамм (вероятности появления цепочки букв N-го порядка (N-грамм) в анализируемых текстах). Он представлен следующими метриками [24]: BLEU, NIST, METEOR.

BLUE (Bilingual Evaluation Understudy) – основная идея: «Чем ближе машинный текст к профессиональному человеческому тексту, тем лучше». Происходит вычисление точности

выявленных n-грамм, так как, например, при переводе текста с одного языка на другой машинным способом оригинал часто оказывается более емким.

NIST (Национальный институт стандартов и технологий США) – в отличие от метода BLEU, помимо точности вычисляется еще и то, насколько информативным является конкретный N-грамм. Так, например, предлог будет иметь значительно меньший вес, чем последовательность из прилагательного и существительного.

METEOR (Metric for Evaluation of Translation with Explicit Ordering) использует функции сопоставления синонимов вместе с точным соответствием слов

3.4 Примеры использования генерации вопросов

В работе [25] приводится самостоятельно разработанное решение на основе уже существующего приложения linkeddata trivia. Результаты приведены в таблице 1.

Таблица 1 – Сравнение решений

	linkeddata trivia	Самостоятельная разработка
Способ построения вопросов	По случайному субъекту	По поиску субъекта; случайному субъекту, определенному по классу; случайному субъекту, определенному по тематике вопросов
Время на поиск субъектов	Около 23 с	< 1 с
Время на построение вопросов	-	Около 5 с
Полный перевод	Нет	Неполный (возможна ситуация, когда литерал на русском отсутствует в базе данных и тогда он заменяется на английский, поэтому некоторые предикаты и субъекты могут быть на английском)
Примеры результатов	Что является частью военного конфликта операции «Радуга»?	Кто, или что, или какой автор Мона Лиза?
Особенности	-	Поиск реализован за счет Google Custom Search. Есть возможность поиска по географической карте.
Источник исходных данных	Семантическая сеть DBPedia	

В статье [26] приводится модель Graph2Seq, которая улучшает современный показатель BLEU-4 с 11,57 до 29,40 и с 25,99 до 59,59 в тестах WQ (WebQuestions) и PQ (PathQuestions) соответственно.

3.5 Проблемы генерации вопросов

Распространённые проблемы в области автоматической генерации вопросов [20]:

1. Сложность моделирования семантики. Наличие в текстах синонимов, омонимов, наличие неточности, неопределенности высказывания, все это вызывает определенные проблемы при анализе текста.
2. Использование контекста. Зачастую человек воспринимает информацию лишь благодаря тому, что понимает контекст, в ходе которого эта информация была получена. Программа же контекста не понимает.
3. Трудности формирования сложных вопросов. Большинство решений позволяют генерировать только самые простые вопросы (вопросы на выбор, вопросы на заполнение пробелов и т. д.). [22]

Проблему контекста можно решить разными способами. Так, в статье [27] используется генерация последовательности вопросов на основе заранее известной последовательности ответов.

Для генерации вопросов выполняется взаимодействие с двумя графами:

1) об информации из отрывка текста; используется для лучшего захвата контекстных зависимостей;
2) об информации из ответов; используется, чтобы сделать сгенерированные вопросы более релевантными заданным ответам.

В работе [27] приводится пример исходного отрывка текста с отметками ответов (в квадратных скобках выделены ответы, относительно которых генерируются вопросы): «A small boy named [John]1 was at the park one day. He was [swinging]2 [on the swings]3 and [his friend]4 named [Tim]5 [played on the slide]6. John wanted to play on the slide now. He asked Tim [if he could play on the slide]7. Tim said [no]8, and he cried». Сгенерированные вопросы к ответам представлены в таблице 2

Таблица 2 – Пример контекстного построения вопросов, приведенный в работе [27]

Сгенерированный вопрос	Ответ
Who was at the park?	John
What was he doing there?	Swinging
On what?	on the wings
Who was he with?	his friend
Named?	Tim
What was he doing?	played on the side
What did John asked him?	if he could play on the slide
What did he say?	no

В статье [28] используется модель Refine Network (RefNet), которая пытается имитировать человеческий процесс создания вопросов, сначала создавая первоначальный черновик, а затем уточняя его. Модель обращает внимание как на исходный отрывок, так и на черновой вопрос и создает грамматически более правильный и полный вопрос. Реализуется это за счет двух декодеров: один формирует черновой вариант, второй формирует конечный вариант.

При этом второй декодер поддается улучшению по таким показателям, как беглость и возможность быть отвеченным (answerability) за счет использования дополнительной модели Reward-RefNet. Она проводит сравнение уточненного вопроса с результатами базового алгоритма (baseline) и вознаграждает систему за изменения, которые улучшают соответствующую метрику во время обучения. Для оценки беглости используется балл BLEU. Для оценки возможности быть отвеченным, используется балл, фиксирующий, содержит ли вопрос требуемые именованные сущности, важные слова, служебные слова, типы вопросов.

Для оценки беглости используется балл BLEU. Для оценки возможности быть отвеченным используется балл, фиксирующий, содержит ли вопрос требуемые именованные сущности, важные слова, служебные слова, типы вопросов.

Пример исходного отрывка текста (жирным шрифтом выделен ответ): «Исполненные заявки передаются администратору информационной безопасности, и хранятся в архиве **в течение 5 лет** с момента окончания предоставления доступа к информационному ресурсу». Результаты генерации вопросов по разным моделям представлены в таблице 3.

Таблица 3 – Результаты для разных моделей

Модель	Сгенерированный вопрос
Baseline	Кому передаются исполненные заявки?
RefNet	Сколько хранятся исполненные заявки?
Reward-RefNet	Сколько хранятся в архиве исполненные заявки?

Проблема генерации сложных вопросов также поддается решению. Например, в статье [29] предлагается комплексное решение DQG (Deep Question Generation), состоящее из трех частей:

1) новый кодировщик графа, который включает механизм внимания в нейронную сеть Gated Graph (GGNN); используется для динамического моделирования взаимодействия между различными семантическими отношениями;

2) усовершенствование вложений отрывков текста на уровне слов и представлений семантического графа на уровне узлов; используется для получения унифицированных представлений отрывков с учетом семантики для декодирования вопросов;

3) введение вспомогательной задачи выбора контента, которая тренируется совместно с декодированием вопросов; используется для выбора подходящих контекстов в семантическом графе для формирования правильной цепочки рассуждений.

Поверхностный вопрос генерируется из одного предложения и апеллирует лишь к одному факту, который можно извлечь из такого предложения (таблица 4).

Таблица 4 – Пример генерации простого вопроса

Исходный текст	Дистанционное техническое обслуживание должно осуществляться только со специально выделенных автоматизированных рабочих мест, конфигурация и состав которых должны быть стандартизованы, а процесс эксплуатации регламентирован и контролироваться.
Вопрос	Какими должны быть автоматизированные места, выделенные для дистанционного технического обслуживания?
Ответ	Стандартизированными.

Сложный (глубокий) вопрос, требующий рассуждений и понимания текста в целом, формируется из нескольких непересекающихся релевантных предложений (таблица 5).

Таблица 5 – Пример генерации глубокого вопроса

Отрывок *A*	Работодатель должен осуществлять передачу персональных данных сотрудника в пределах одной организации в соответствии с локальным нормативным актом организации, с которым сотрудник должен быть ознакомлен под расписку.
Отрывок *B*	Разрешать доступ к персональным данным сотрудников работодатель должен только уполномоченным лицам, при этом указанные лица должны иметь право получать только те персональные данные сотрудника, которые необходимы для выполнения конкретных функций.
Вопрос	Согласно какому нормативно-правовому акту работодатель должен осуществлять передачу персональных данных сотрудника в пределах одной организации и разрешать доступ к персональным данным сотрудников только уполномоченным лицам?
Ответ	Согласно Трудовому Кодексу.

4. Введение в игрофикацию

4.1 Основные понятия

Игрофикация (геймификация) – искусство извлекать из игр элементы и механизмы, которые делают их увлекательными, и применять их в других видах деятельности людей, повышая их внутреннюю и внешнюю мотивацию.

Игра – это процесс добровольного решения поставленных проблем через определенные правила и приносящий удовольствие от результатов.

Мотивация (от лат. movēre «двигать») – побуждение к действию или движущая сила; психофизиологический процесс, задающий поведение человека, его устойчивость, активность, организацию, направленность; совокупность мотивов (причин), обуславливающих решение или поступок человека.

Одной из главных идей в игрофикации, которая значительно повлияла на данную сферу является концепция октализа, придуманная практикующим с 2003 года специалистом в игрофикации Ю Кай Чоу.

Октализ – схема восьми мотивационных стимулов, позволяющая анализировать и создавать геймифицированные системы, которые мотивируют людей на определенные действия [30].

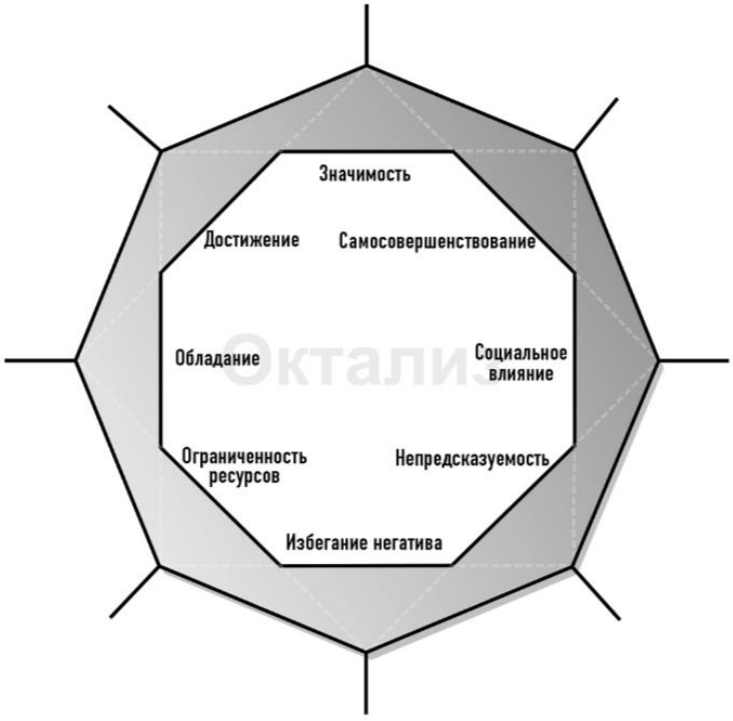

Рисунок 9 – Схема «Октализ» [30]

Мотивация поступков людей основана на одном или сочетании сразу нескольких стимулах [30]:

1. **Значимость** – ощущение, что ты делаешь нечто важное, глобальное, сосредоточенное не на личной выгоде и самом себе, а на чем-то более высоко духовном, благородном. Пример: добровольцы, что заполняют Википедию.
2. **Самосовершенствование** – люди нуждаются в способах самовыражения, видеть результат своих трудов, получать обратную реакцию и совершенствовать что-то в ответ.
3. **Социальное влияние** – зависть, восхищение, ностальгия, признание, наставничество, обратная связь и конкуренция мотивируют людей.
4. **Непредсказуемость** – когда что-то происходит незапланированно, ваш мозг начинает работать на высокой скорости и обращает внимание на новое событие. Именно из-за этого стимула многие люди смотрят фильмы или читают романы.
5. **Избегание негатива** – вероятность что-либо потерять или испытать какое-то неприятное ощущение или понимание, что возможности ускользают от нас, активирует этот стимул. Люди чувствуют, что, если они не будут действовать немедленно, то навсегда потеряют открывшуюся возможность. Пример: специальное предложение действующее ограниченное количество времени.
6. **Ограниченность ресурсов** – «запретный плод сладок». Невозможность обладать чем-то сразу стимулирует нас желать его еще больше.
7. **Обладание** – чувство собственности мотивирует людей к накоплению чего-либо. Так же если человек много тратит времени на настройку профиля или аватара, то он автоматически воспринимает их как собственность.
8. **Достижение** – ощущение прогресса, получаемого в ходе преодоления неких препятствий и вызовов. А также получение наград, фиксирующих этот результат (очки, баллы, трофеи, рейтинги).

Расположение этих стимулов выражает характер мотивации, которую они вызывают. В правой части стимулы **внутренней мотивации** (сам процесс уже награда), включающие в себя стимулы творчества, самореализации и социализации. В левой части стимулы **внешней мотивации**, включающие в себя стимулы логики, аналитического мышления и чувства собственности.

Эффект переоценки. Если уделять внимание только внешней мотивации, то в случае прекращения её стимулов, пользователи теряют интерес и их общий уровень мотивации оказывается даже ниже, чем до введения внешних стимулов.

В долгосрочной перспективе игрофикация за счет только или преимущественно внешних стимулов может иметь неблагоприятный эффект, поскольку она снижает внутреннюю мотивацию и вызывает своего рода аддиктивное состояние, причиной которого является эффектом чрезмерного оправдания (или эффект невозвратных затрат). Это психологическая ловушка, когда естественный интерес сменяется гонкой за наградами. Но игроки при этом мотивированными не наградами, а неприятием отказа от деятельности, в которую они вложили большое количество времени и усилий. Им кажется, что если они прекратят эту деятельность, то вложенные ресурсы окажутся зря потраченными. Тем самым они оправдывают дальнейшее вложение своих ресурсов [31].

4.2 Средства игрофикации

Динамики — это абстрактные аспекты игрофицированной системы. Их нужно учитывать и контролировать, но их невозможно непосредственно внедрить в игровой процесс [32].

Распространенные динамики:

1) ограничения (лимиты или вынужденные компромиссы);
2) эмоции (любознательность, дух соперничества, разочарование, счастье);
3) повествование (последовательная, непрерывная сюжетная линия);
4) продвижение (рост игрока и его развитие);
5) отношения (социальные взаимодействия, формирующие чувства товарищества, статуса и альтруизма).

Механики – способ достижения одной или нескольких описанных динамик [32].

Распространённые механики:

1) задания – загадки, требующие усилий для их решения;
2) шанс – элементы случайности;
3) соревнование – один игрок или группа игроков побеждает, а вторая — проигрывает;
4) сотрудничество – игроки должны работать вместе, чтобы достичь общей цели;
5) обратная связь – информация об успехах игрока;
6) накопление ресурсов – получение полезных или коллекционных предметов;

7) вознаграждения – награды за определенные действия и достижения;
8) сделки – торговые операции между игроками, напрямую или через посредников;
9) ходы – поочередное участие меняющихся игроков;
10) состояние победы – показатели, которые превращают игрока или команду в победителя; состояния выигрыша и проигрыша — связанные понятия.

Компоненты — это более конкретная форма, которую принимают механики и динамики [32].

Распространенные компоненты:

1) достижения (определенные цели);
2) аватары (визуализация характера игрока);
3) бейджи (визуализация достижений);
4) битвы с боссами (особенно сложные испытания для перехода на следующий уровень);
5) коллекционирование (накопление наборов предметов или бейджей);
6) сражения (конкретная борьба, обычно быстрая);
7) доступ к контенту (то, что открывается игрокам, когда они достигают определенных показателей);
8) подарки (возможность делиться ресурсами с другими);
9) рейтинги лидеров (визуализация развития и достижений игрока);
10) уровни (определенные шаги в развитии игрока);
11) очки (количественное отображение развития игры);
12) квесты (конкретные задачи со своими целями и наградами);
13) социальный профиль (визуализация игры в социальной сети игрока);
14) команды (определенные группы игроков, работающих вместе ради общей цели);
15) виртуальные товары (игровые активы с субъективной или реальной денежной ценностью).

4.3 Примеры игрофикации

В статье [33] приводится ряд приложений, использующих средства игрофикации.

Duolingo (платформа для изучения естественных языков). Процесс обучения делится на небольшие этапы. Пока пользователи проходят уроки, они получают очки опыта (XP, eXperience Points) и слитки (вымышленный драгоценный камень). Их накопление дает

пользователю возможность получить доступ к специальным функциям и занять более высокие места в списках лидеров.

Codecedemy. (платформа для обучения написанию программного кода для создания веб-сайтов или приложений). Пользователи получают значки за прохождение курсов, а также могут всегда следить за своим прогрессом на индикаторе выполнения для каждого изучаемого навыка.

В сфере спорта такие веб-компаний как, например, Nike+, Strava, Endomondo, Runkeeper используют мгновенную обратную связь с пользователем, предоставляя статистику и достижения на основе его активности. А также они применяют социальное соревнование, где пользователи, соревнуясь видят, как их друзья оцениваются по сравнению с ними.

В сфере здравоохранения и медицины есть приложения Pain Squad и MySugr. Pain Squad нацелено на детей от 8 до 18 лет, страдающих от заболевания раком. Врачам нужна обратная связь от пациентов, чтобы знать, как работает лечение. Врачи могут устанавливать связи только в том случае, если им регулярно поступают сообщения о боли, которую испытывает пациент, иначе они не могут сделать никаких выводов. Игрофикация здесь необходима, чтобы мотивировать детей каждый день сообщать о своей боли, чтобы найти лучшие способы лечения рака.

Приложение MySugr представляет собой всесторонний уход за людьми с диабетом. Оно реализует игрофикацию только в одном аспекте – «монстр-компаньон». Пользователь может дать ему имя и попытаться «приручить» его, посредством ежедневных действий, регистрируя данные, оставаясь в пределах нормальных показателей или просто проявляя активность. Кроме того, «монстр-компаньон» обеспечивает обратную связь. Он выдает различные предупреждающие комментарии, когда пользователь сохраняет данные и его показатели оказываются выше или ниже установленных границ. Это приложение мотивирует пользователя изменить свое поведение в лучшую сторону, стать здоровее.

Платформа Habitica (помогает людям достичь своих целей). Приложение стилизованно под такой жанр компьютерных и настольных игр, как RPG (Role-Playing Game). Пользователи имеют собственных аватаров, создают свои цели или привычки, которые им хочется развить. Выполняя или пропуская регулярные действия, достигая или проваливая поставленные цели пользователи соответственно получают вознаграждение (очки опыта, золото (внутриигровая валюта)) и штрафы (потеря очков здоровья у созданного аватара). Так же некоторые функции доступны только за золото, таким образом, пользователи могут «тратить» накопленный ресурс.

5. Техническая реализация

5.1 Схема технических функций программы

В ходе реализации технических функций, изложенных на рисунке 10, часть из них, получилось выполнить несколько иным образом (см. раздел 5.4 о генерации вопросов), а другую часть отложить на дальнейшую перспективу разработки (магазин наград, сами награды и менеджмент тестов).

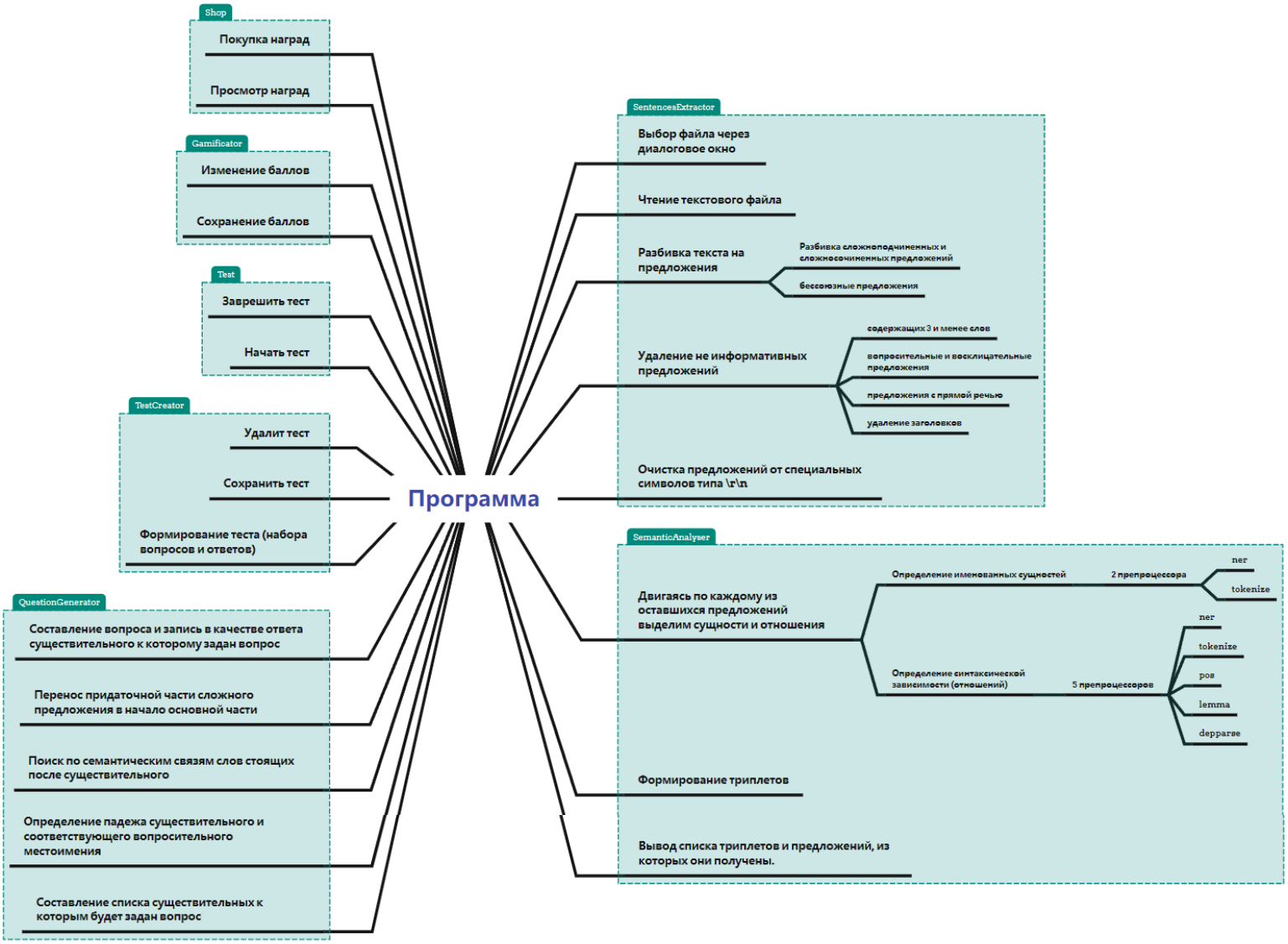

Рисунок 10 – Изначальная схема технических функций, выполняемых разрабатываемой программой

5.2 Анализ и выбор средств разработки

Среди инструментов для реализации NLP существуют автономные библиотеки (таблица 6) доступные для скачивания, а есть веб-сервисы, но их функционал недостаточно гибкий для интеграции с разрабатываемым приложением.

Таблица 6 – Сравнение программных библиотек для NLP [17, 16, 34, 35, 36, 37]

Библиотека	Язык программирования	Функционал	Поддерживаемые языки
DeepMorhy	C#	Морф. анализ, нормализация	RU
Catalyst NLP	C#	Морф. анализ, нормализация, word embeddings	60 языков, включая русский
Pullenti	C#, Java, Python JavaScript	NER, Tokenize, нормализация, relationships (sem. graph)	RU, UK, EN
Natasha	Python	NER, морф. анализ, нормализация, сегментатор синтаксическая модель	RU
Tomitaparser	C++	NER, морф. анализ, Tokenize, сегментатор	RU
Stanza	Python	NER, Depparse, POS, LEMMA, Tokenize, MWT	66 языков, включая русский

Программы для построения графов знаний (таблица 7). Большинство из них имеют следующий базовый набор свойств: масштабируемость и расширяемость, безопасность на корпоративном уровне, визуализация графов.

Таблица 7 – Сравнение онлайн-программ для построения графов знаний [28, 29, 30]

Программа	Доступ	Особенности	Язык запросов	ОС
Hume от Graph Aware	По личному запросу	- геопространственный анализ отображает данные в соответствующем местоположении; - временной анализ показывает, как данные меняются с течением времени. - обработка неструктурированных данных.	Cypher	Linux; Windows; MacOS.
Neo4J	Бесплатное скачивание и регистрация + наличие google billing account для NLP функций	- управление доступом на основе ролей; - интеграция с облачной средой.	Cypher	MacOS 10.10 (Yosemite)+; Windows is 8.1+ w/Powershell; Ubuntu 12.04+; Fedora 21; Debian 8.
Nebula Graph	Бесплатное скачивание	- автоматическое аварийное восстановление без простоев; - нет привязки к облачной среде; - открытый исходный код под управлением Apache 2.0.	nGQL	CentOS 7, 8; Ubuntu 1604, 1804, 2004.

Для удобства восприятия построенные графы можно визуализировать. Для этого три наиболее популярные программы, рассмотренные в таблице 8.

Таблица 8 – Сравнение программ для визуализации графов знаний [31, 32, 33, 34]

Программа	ОС	Форматы экспорта	Преимущества	Недостатки
NetworkX	Cross-platform	Adjacency List, Multiline Adjacency List, Edge List, GEXF, GML, Pickle, GraphML, JSON, LEDA, SparseGraph6, Pajek, GIS Shapefile, Matrix Market.	- простота освоения; - возможность визуализации объемных и плоских графиков.	- тяжело построить сложный граф.
Graphviz	Windows, Linux, MacOS, Solaris, Other Unix	BMP, CGImage, DOT, EPS, EXR, FIG, GD/GD2, GIF, GTK, ICO, JPEG, JPEG 2000, JSON, PDF, PIC, PICT, Plain Text, PNG, POV-Ray, PS, PS/PDF, PSD, SGI, SVG, TGA, TIFF, Tk, VML, VRML, WBMP, WebP, X11	- удобство в создании графов разной сложности; - широкий инструментарий.	- трудности для работы во внутренней закрытой сети, т.к. программа использует Интернет-соединение.
Gephi	MacOS, Windows, Linux	CSV, GDF, GEXF, GraphML, Pajek NET, Spreadsheet, PDF, SVG	- удобство внесения изменений в графе с помощью мыши; - бизнес-аналитика; - интеграция с Excel; - интерактивное исследование графиков.	- отсутствует откат назад, что существенно осложняет редактирование графа; - низкая производительность при создании больших графов (около миллиона вершин).

После ряда проб и ошибок был выбран наиболее подходящий под имеющиеся ресурсы и навыки инструмент: библиотека Pullenti. Из всех схожих библиотек (таблица 6) создать семантический граф могли только Pullenti и Stanza. А т.к. именно C# является основным языком программирования для автора данной работы, то выбор пал не в пользу последней. Что касается средств создания графов знаний (таблица 7), то все они завязаны на онлайн режиме и соответственно оказались недоступны в виду санкционно-политической обстановки на момент написания работы. Необходимости в использовании приложений для визуализации графов (таблица 8) не возникло в связи с отсутствием возможности построить сам граф знаний.

5.3 Извлечение текста из основных форматов

В качестве первого шага необходимо извлечь текстовые данные из заранее записанного файла. Для этого используется библиотека SautinSoft.Document, позволяющая отделить интересующие нас текстовые данные от служебно-технической информации необходимой для работы текстовых форматов (doc, docx, pdf и т.д.). Применение данной библиотеки демонстрируется на рисунках 11-13.

```
39          private void butt_OpenFile_Click(object sender, RoutedEventArgs e)
40          {
41              OpenFileDialog openFileDialog = new OpenFileDialog();
42              openFileDialog.Multiselect = false;
43              openFileDialog.Filter = "All files|*.*|Word|*.docx|Word 97-2003|*.doc|PDF files|*.pdf|Text files|*.txt";
44              Nullable<bool> dialogOk = openFileDialog.ShowDialog();
45
46              if (dialogOk == true)
47              {
48                  filePath = openFileDialog.FileName;
49              }
50          }
```

Рисунок 11 – Метод butt_OpenFile_Click(), вызываемый при нажатии кнопки «Открыть файл»

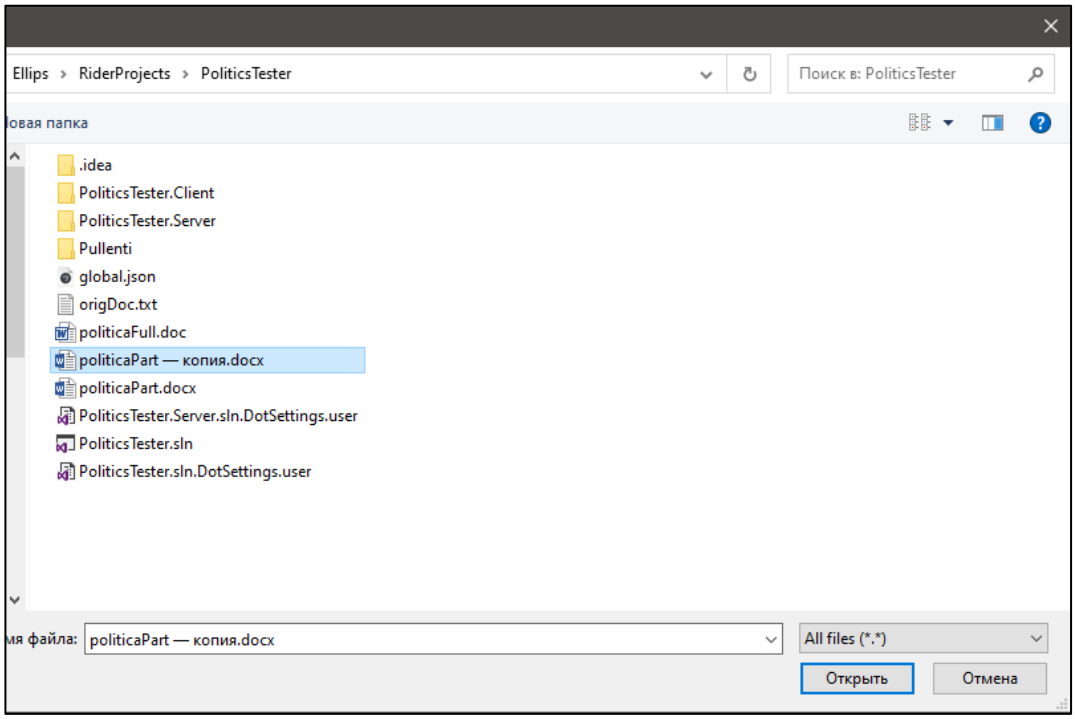

Рисунок 12 – Окно выбора файлов, открывающееся после нажатия кнопки «Открыть файл»

```
93          private string GetTextFromFile()
94          {
95              if (File.Exists(filePath))
96              {
97                  DocumentCore dc = DocumentCore.Load(filePath);
98                  return dc.Content.ToString();
99              }
100             else
101                 return "";
102         }
```

Рисунок 13 – Код метода GetTextFromFile(), возвращающего непосредственно интересующий нас текст из выбранного файла

5.4 Создание семантического графа и вопросов

Извлеченный текст необходимо подготовить для лингвистического анализа, что включает в себя следующие задачи: разбить текст на предложения, удалить служебные символы возврата каретки и начала новой строки и удалить слишком короткие предложения (из 1-2 слов).

```
10          public string[] GetSentences(string text)
11          {
12              text = DeleteSpecialSymbols(text);
13              text = ReplaceDotsBetweenNumbers(text);
14
15              char[] separators = {'|'};
16
17              List<string> splitedText = new List<string>();
18
19              text = text.Replace( oldValue: ".", newValue: ".|").Replace( oldValue: "?", newValue: "?|").Replace( oldValue: "!", new
20                  .Replace( oldValue: ".|.|.|", newValue: "...|"); // string
21
22              splitedText.AddRange( collection: text.Split(separators, StringSplitOptions.RemoveEmptyEntries));
23
24              splitedText = splitedText.Select(sent :string => sent.Trim()).ToList();
25
26              DeleteExcessSentences(splitedText);
27
28              return splitedText.ToArray();
29          }
```

Рисунок 14 – Код метода GetSentences(), возвращающего подготовленные предложения

Как видно из рисунка 14, для разбития текста на предложения используется стандартный для языка C# метод Split, для работы которого необходим набор сепараторов (символов, являющихся маркером для выделения подстроки). В нашем случае это знаки препинания: «.», «,», «?», «!», «!?», «…». При этом сами сепараторы не попадают в выделенные подстроки. Чтобы этого избежать, ко всем знакам препинания добавим символ «|», который нигде в тексте не используется. Он и будет сепаратором.

Получив набор предложений, их нужно обработать в трех методах: ExtractEntities() – извлечение сущностей, ExtractRelationships() – извлечение отношений и построение семантического графа, GenerateQuestions() – генерация вопросов (рисунок 15).

```
32              var sentences:string[] = _extractorSentences.GetSentences(textModel.Text);
33              var questionsAndAnswers = new List<QA>();
34
35              foreach (var sentence:string in sentences)
36              {
37                  var basicResult = _semanticAnalyser.ExtractEntities(sentence);
38                  var semDoc:SemDocument = _semanticAnalyser.ExtractRelationships(basicResult);
39                  var currentQuestionsAndAnswers:List<QA> = _questionsGenerator.GenerateQuestions(semDoc);
40                  questionsAndAnswers.AddRange(currentQuestionsAndAnswers);
41              }
42
43              _questionSets.SetsValues = questionsAndAnswers;
```

Рисунок 15 – Код, где происходит обработка каждого предложения

Промежуточные результаты обработки предложений изображены на рисунке 16.

```
АДМИНИСТРАЦИЯ АЛТАЙСКОГО КРАЯ УПРАВЛЕНИЕ АЛТАЙСКОГО КРАЯ ПО КУЛЬТУРЕ И
АРХИВНОМУ ДЕЛУ политика информационной безопасности
Содержание Вводные положения4 1,1.
*****NamedEntities:
Алтайский
Администрация
Управление
*****TokensNameGroups:
КУЛЬТУРЕ
АРХИВНОМУ ДЕЛУ
политика
информационной безопасности
Содержание
Вводные положения
>>>>>>>>>>>Triples:
<КУЛЬТУРА ><Detail - чего><политика >
<политика ><Detail - чего><информационная безопасность >
<КУЛЬТУРА ><Detail - чего><информационная безопасность >
<Содержание ><Naming - ><Вводные Положения >
<Вводные Положения ><Detail - какой><4 1,1 >
<Содержание ><Detail - какой><4 1,1 >

Период действия и порядок внесения изменений 5 2.
*****NamedEntities:
NOT FOUND
*****TokensNameGroups:
Период
действия
порядок
внесения
изменений
>>>>>>>>>>>Triples:
<Период ><Detail - чего><действие >
<Период ><Detail - чего><порядок >
<действие ><Detail - чего><внесение >
<внесение ><Detail - чего><изменения >
<внесение ><Detail - какой><5 2 >
```

Рисунок 16 – Результат построения семантических графов для предложений

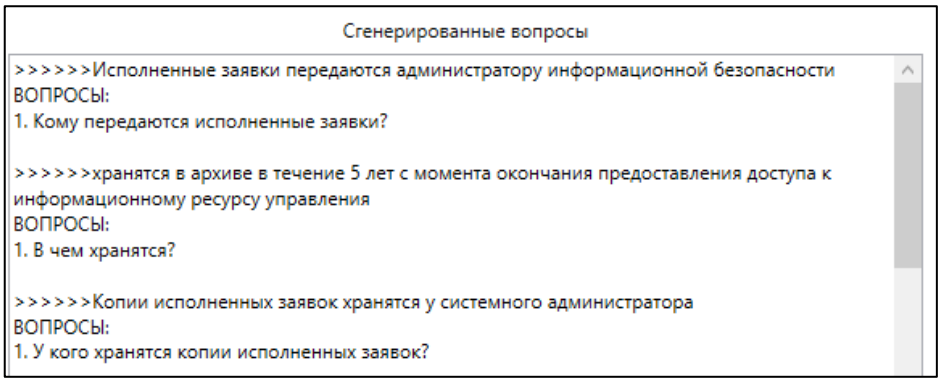

Рисунок 17 – Результат генерации вопросов

Как видно из рисунка 17, не все вопросы получаются достаточно корректными. Причиной этому является несовершенство алгоритма генерации, основанного на простой логике применения таких правил как: вопросительное местоимение должно стоять на первом месте, затем должен обязательно располагаться глагол, придаточная часть сложного предложения должна выноситься перед основной и др.

5.5 Клиент-серверная архитектура

Дальнейшим шагом была реализации архитектуры «Клиент-сервер», в которой задания (сетевая нагрузка) отдаются серверу, а клиенты лишь делают запросы, не производя ни каких вычислений. При этом сервер не имеет никакого пользовательского интерфейса (кроме отладочного, как на рисунке 21). Для реализации архитектуры использовалась платформа ASP.NET Core 5.0.

Перед написанием кода следует учесть следующий момент: программа будет передавать данные клиентам через протокол HTTPS, для которого является обязательным использование технологии сертификатов, а значит необходимо произвести соответствующую настройку. Программа выдаст ошибку если не установить в системе SSL сертификат. Для этого достаточно в окне командной строки, запущенной от администратора, ввести команду «dotnet dev-certs https –trust» (рисунок 18).

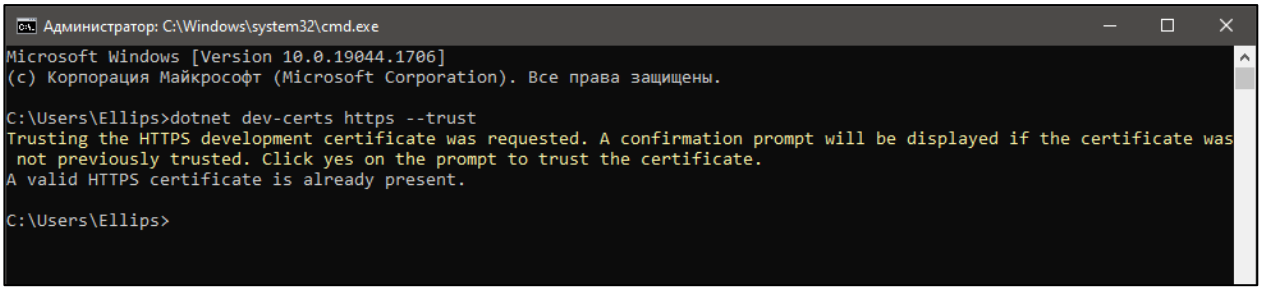

Рисунок 18 – Установка сертификата SSL

Рассмотрим реализацию серверной части приложения (рисунок 19).

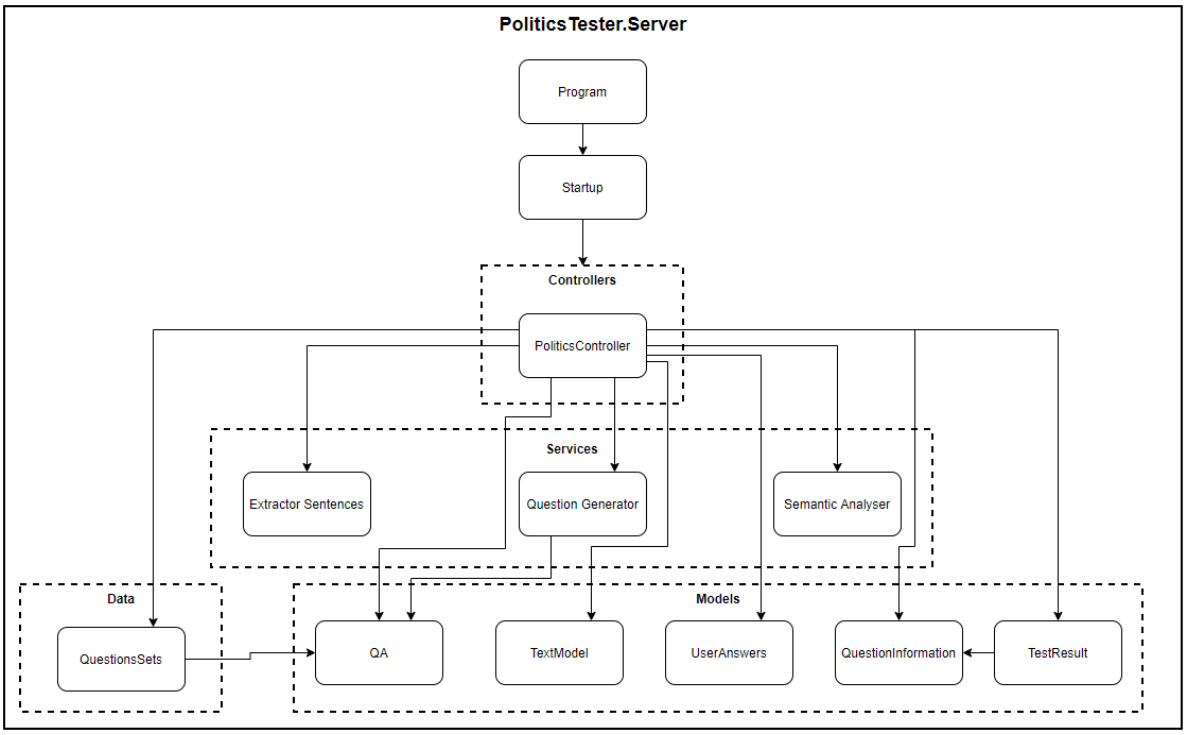

Рисунок 19 – Диаграмма взаимодействия классов серверной части приложения

Класс Program – инициирует работу программы. Он развертывает веб-приложение через создание объекта IHost посредством стандартного метода CreateHostBuilder.

Класс Startup – является продолжением инициации работы программы. Задачи класса, следующие:

- конфигурация приложения;
- настройка сервисов, используемых приложение сервисов (в нашем случае это Extractor Sentences, Question Generator, Semantic Analyser);
- установка middleware (компонентов для обработки запроса).

Класс PoliticsController – главный элемент архитектуры ASP.NET Core. Этот класс отвечает за обработку запросов и возврат результатов обработки. Он содержит в себе два ключевых метода: GetQuestions и GetTestResult. Первый имеет атрибут [HttpPost], второй [HttpGet]. Это разделение запросов на типы методов действий.

Методы действия – методы, всегда имеющие модификатор доступа public. Это именно те методы, которые вызываются при получении контроллером запросов. Виды стандартных атрибутов:

- [HttpGet] – отправка данных на сервер;
- [HttpPost] – чтение данных с сервера;

- [HttpPut] – обновление данных;
- [HttpDelete] – удаление данных по ID;
- [HttpHead] – проверка на наличие данных по указанному адресу и проверка на присутствие их изменения с момента последнего обращения;
- [HttpPatch] – частичное изменение данных.

Таким образом при вызове GetQuestions серверу передаются ещё не обработанные текстовые данные, извлеченные из файла политики безопасности. Эту входную информацию метод передает сервисным методам Extractor Sentences, Question Generator и Semantic Analyser.

Реализованные методы действия доступны для вызова без запросов от клиента через стандартный отладочный интерфейс Swagger (рисунок 21), который был установлен при конфигурации сервера в классе PoliticsController (рисунок 20).

```
29          services.AddSwaggerGen(c :SwaggerGenOptions =>
30          {
31              c.SwaggerDoc( name: "v1", new OpenApiInfo {Title = "PoliticsTester.Server", Version = "v1"});
32          });
```

```
38      if (env.IsDevelopment())
39      {
40          app.UseDeveloperExceptionPage();
41          app.UseSwagger();
42          app.UseSwaggerUI(c :SwaggerUIOptions => c.SwaggerEndpoint( url: "/swagger/v1/swagger.json", name: "PoliticsTester.Server v1"));
43      }
```

Рисунок 20 – Код позволяющий проводить вызов методов действий сервера вручную, без запросов клиента

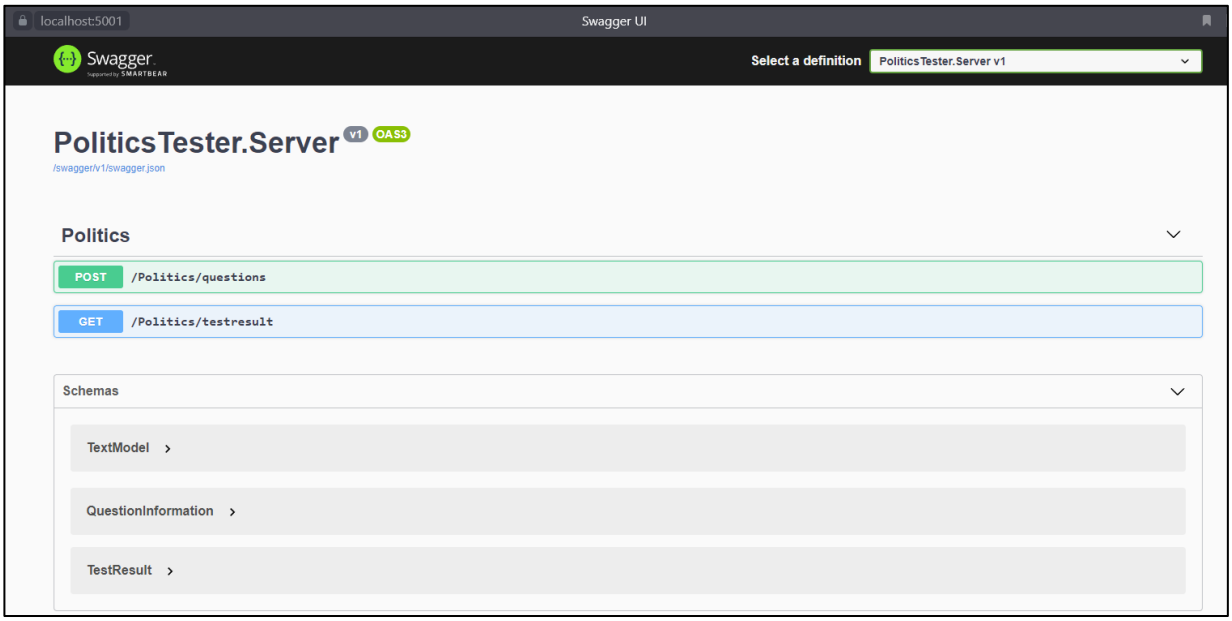

Рисунок 21 – Окно отладки функций сервера в браузере

Теперь рассмотрим клиентскую часть приложения (рисунок 22).

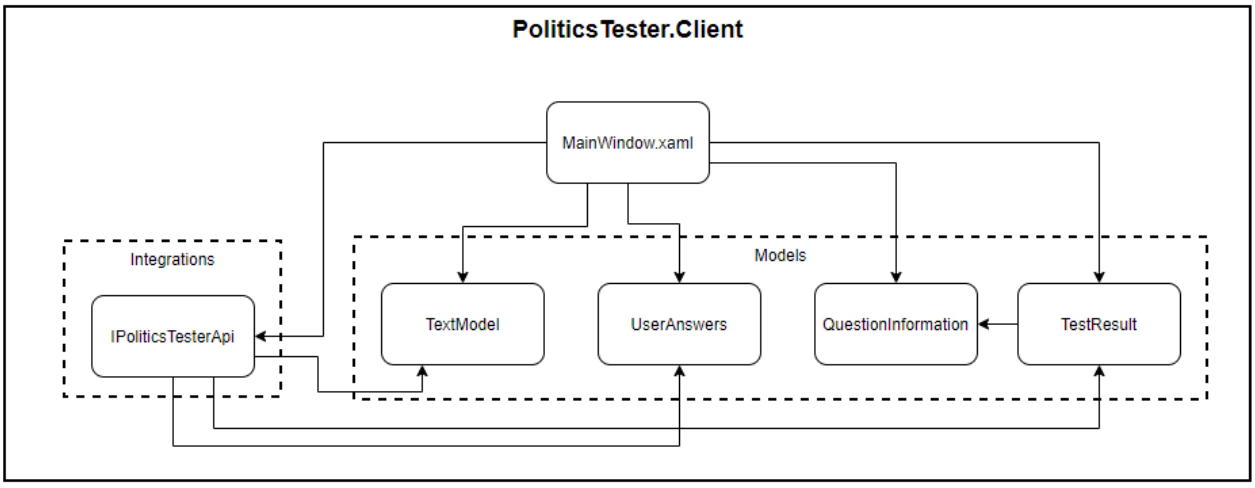

Рисунок 22 – Диаграмма взаимодействия классов клиентской части приложения

Для того, чтобы клиент и сервер друг друга «видели» и могли обмениваться данными необходимо произвести ряд шагов. Для этого мы используем библиотеку Refit, которая позволяет относительно просто выполнять вызовы REST API без написания большого количества кода.

REST API – архитектурный стиль взаимодействия элементов распределенного приложения в сети. Это перечень правил по организации написания кода серверной части приложения. Эти правила обеспечивают легкость обмена данными между системами и простоту масштабирования приложения.

При инициализации клиентской программы в файле MainWindow.xaml.cs в 32 строке (рисунок 23) укажем инструкцию, которая генерирует реализацию IGitHubApi, используемую HttpClient'ом для выполнения своих вызовов. В нашем случае, для простоты отладки, обмен данными будет происходить через localhost, т.е. и сервер и клиент находятся на одном компьютере.

```
25          public MainWindow()
26          {
27              UserAnswers = new UserAnswers();
28              UserAnswers.Answers = new List<string>();
29
30              InitializeComponent();
31
32              iPoliticsTesterApi = RestService.For<IPoliticsTesterApi>( hostUrl: "https://localhost:5001");
33
34              questions = new List<string>();
35              SummPoints = 0;
36              RefreshUI( idCase: 0);
37          }
```

Рисунок 23 – Код необходимый для установки соединения между клиентом и сервером

Интерфейс IPoliticsTesterApi помимо методов описывает также их атрибуты [Post] и [Get], в которых прописывается путь, благодаря которому и происходит установка связи с соответствующими методами действиями на сервере (рисунок 24 и 25).

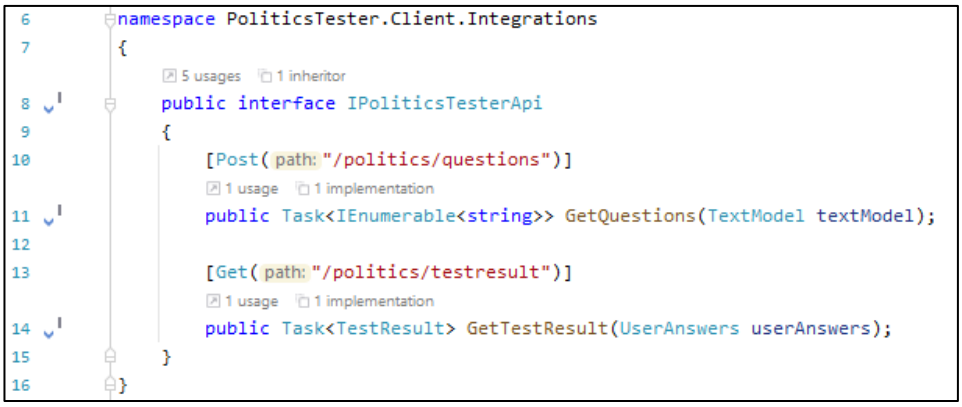

Рисунок 24 – Атрибуты методов, прописанные в интерфейсе IPoliticsTesterApi

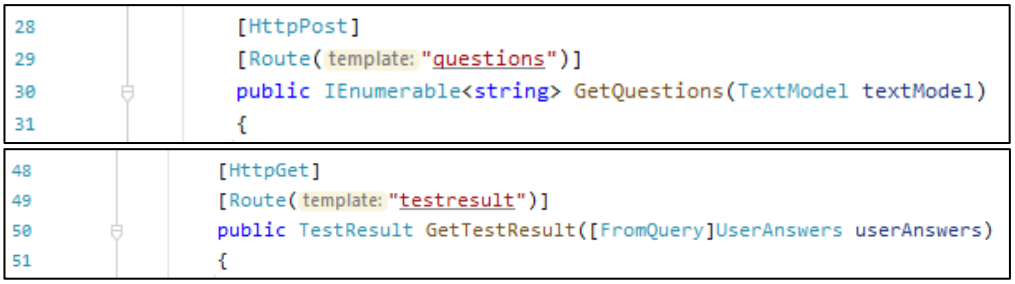

Рисунок 25 – Атрибуты методов, прописанные в контроллере PoliticsController

Далее во все том же файле MainWindow.xaml.cs реализуем методы интерфейса IPoliticsTesterApi для получения вопросов и получения результатов прохождения теста, GetQuestions и GetTestResult соответственно (рисунок 26).

```
196         public TestResult GetTestResult()
197         {
198             TestResult response = null;
199             try
200             {
201                 response = iPoliticsTesterApi.GetTestResult(UserAnswers).Result;
202             }
203             catch (Exception e)
204             {
205                 Console.WriteLine(e.Message);
206             }
207             return response;
208         }
            1 usage
209         public IEnumerable<string> GetQuestions(TextModel textModel)
210         {
211             IEnumerable<string> response = null;
212             try
213             {
214                 response = iPoliticsTesterApi.GetQuestions(textModel).Result;
215             }
216             catch (Exception e)
217             {
218                 Console.WriteLine(e.Message);
219             }
220             return response;
221         }
```

Рисунок 26 – Реализация методов GetQuestions и GetTestResult

Данные методы вызываются при нажатии соответствующих кнопок пользовательского интерфейса клиентской части приложения: GetQuestions при нажатии кнопки «Получить вопросы», а GetTestResult при нажатии кнопки «Завершить» (рисунок 27).

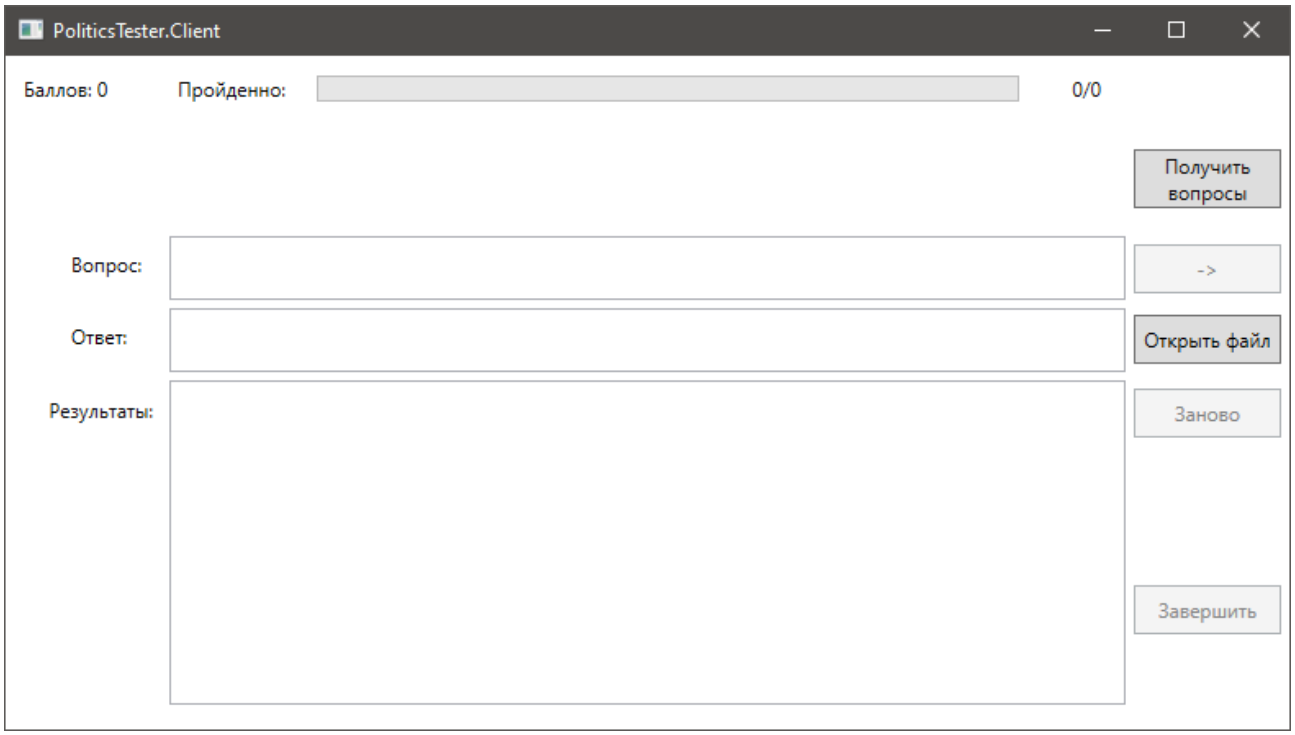

Рисунок 27 – Окно клиента при первичном запуске

5.6 Адаптивный пользовательский интерфейс

Помимо генерации вопросов был разработан адаптивный пользовательский интерфейс на базе WPF (Windows Presentation Foundation) для прохождения теста (рисунок 28).

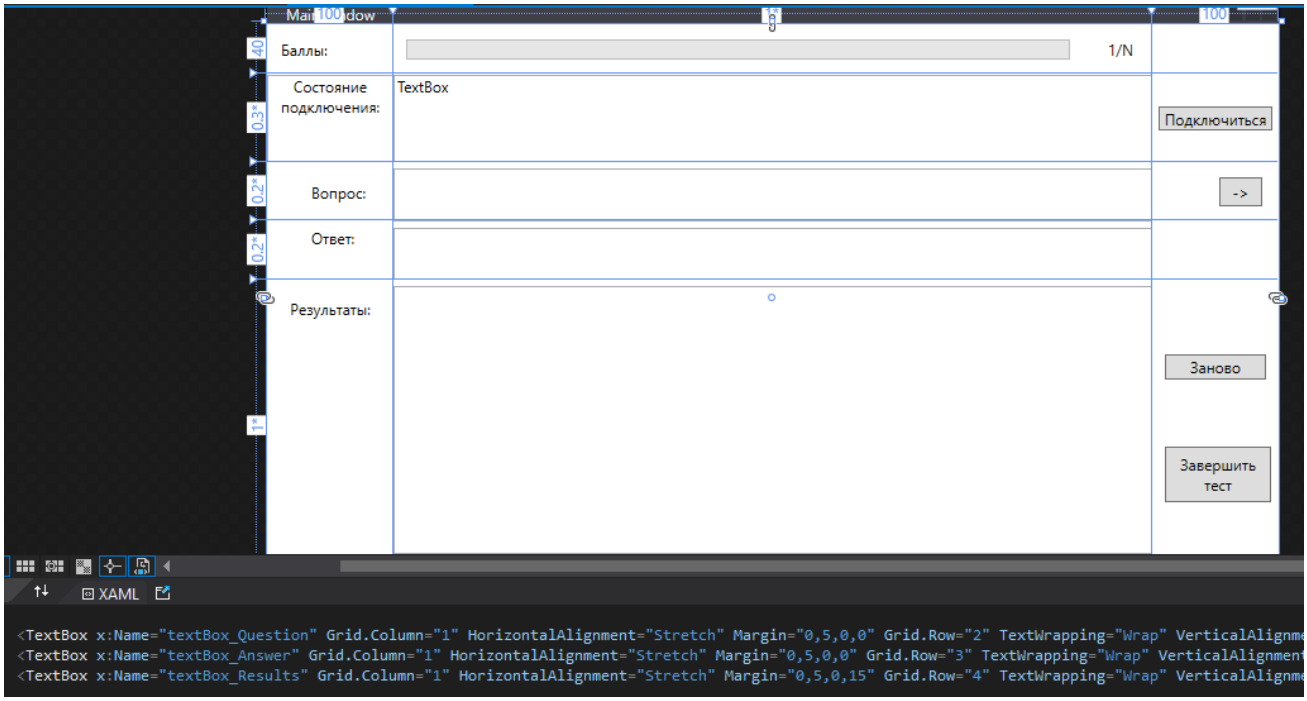

Рисунок 28 – Окно конструктора интерфейса приложения

Благодаря WPF окно приложения можно растягивать и сжимать без потери читабельности и функциональности интерфейса. Таким образом при большинстве разрешений мониторов, приложение не потребует индивидуальной настройки графических элементов под новые условия, все произойдет автоматически. Пример деформации окна представлены на рисунках 29 и 30.

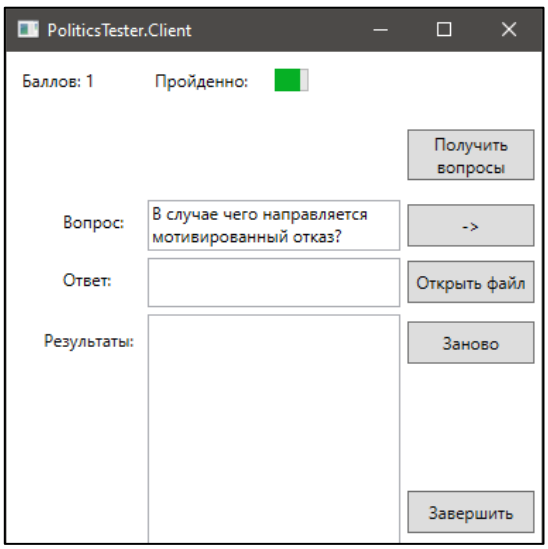

Рисунок 29 – Пример деформации окна приложения №1

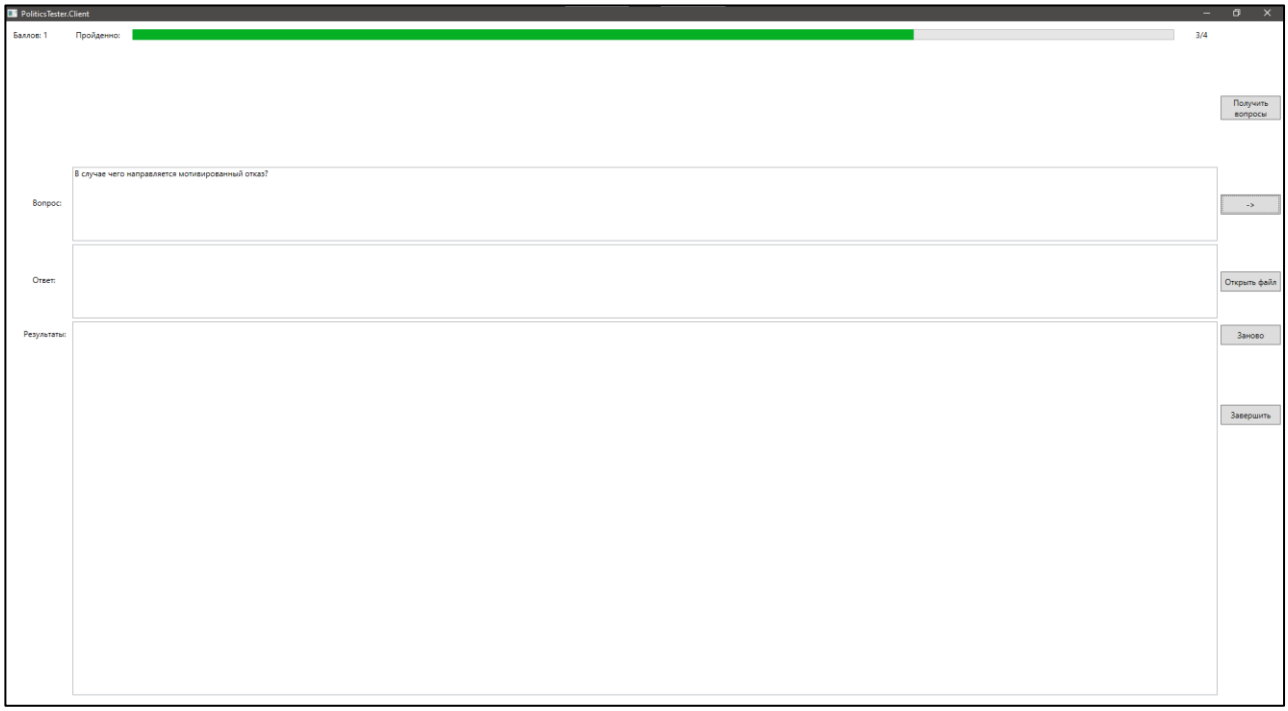

Рисунок 30 – Пример деформации окна приложения №2

5.7 Интеграция в процесс прохождения тестов методов игрофикации

В качестве реализации методов игрофикации была реализована балльная система – за прохождения теста, пользователь получает баллы. За каждый верный ответ 1 балл, за каждый не верный 0 баллов. Демонстрация прохождения теста видна на рисунке 31-33.

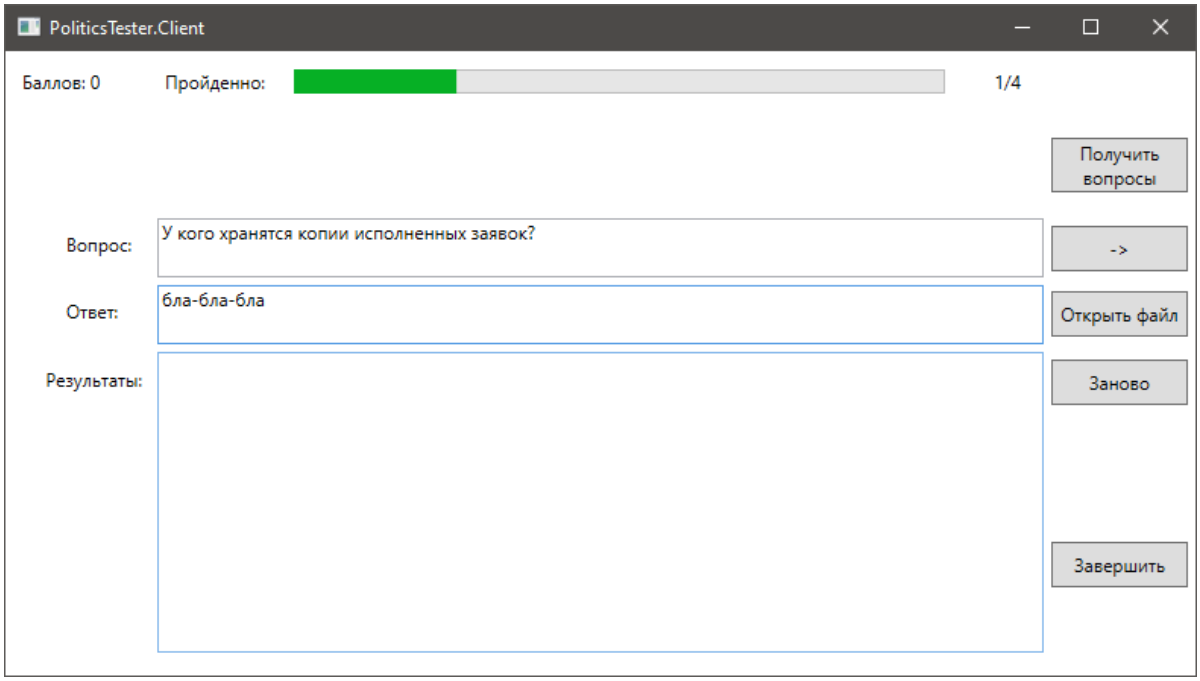

Рисунок 31 – Окно клиента при прохождении теста

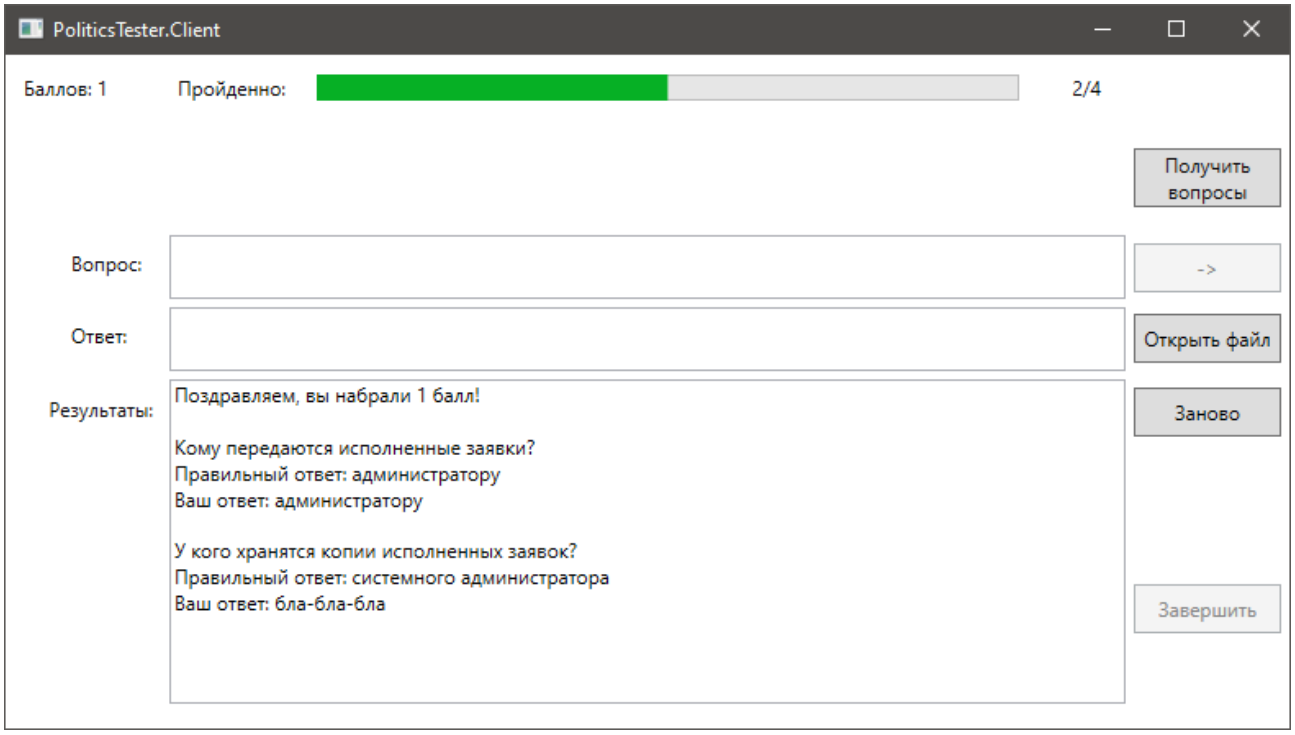

Рисунок 32 – Окно клиента при завершении теста с не нулевым результатом

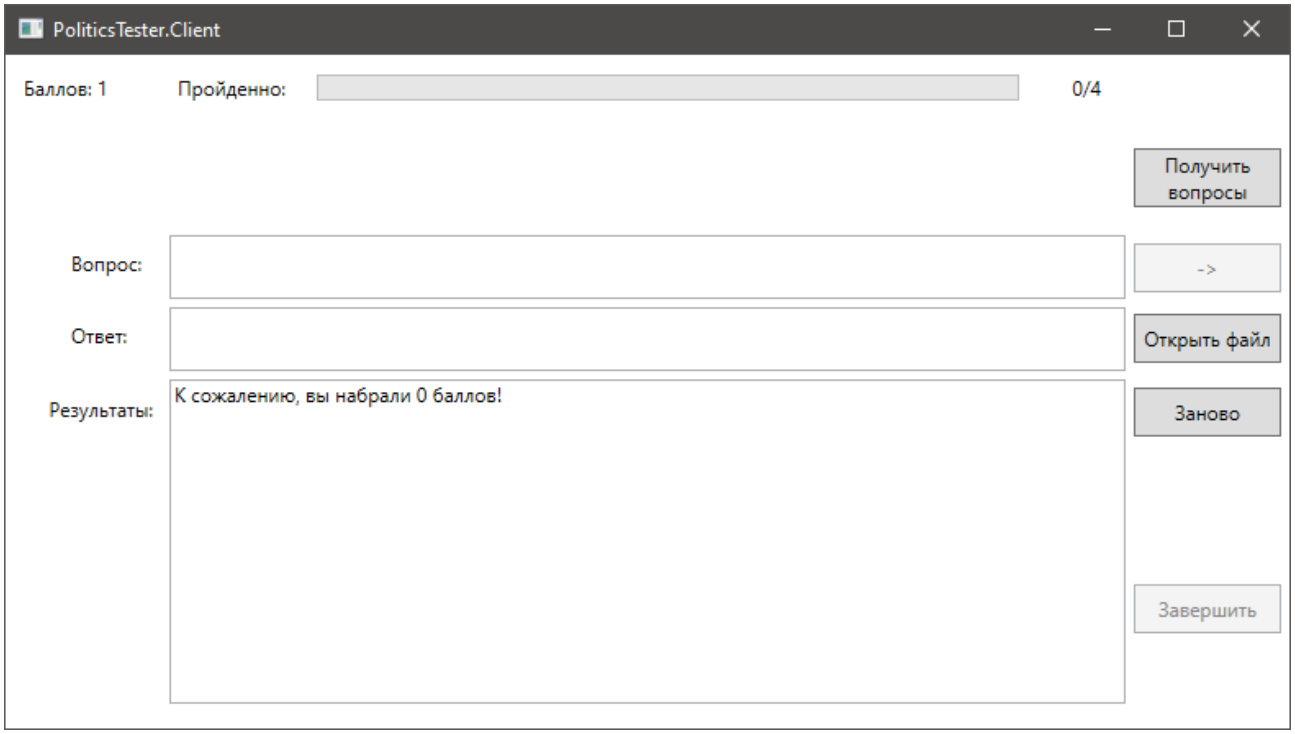

Рисунок 33 – Окно клиента при завершении теста с нулевым результатом

Данные о баллах и результатах прохождения хранятся у самих клиентов в виде класса TestResult. Он содержит свойство Points и массив QuestionResults, каждый элемент которого

представлен отдельным классом QuestionInformation, который в свою очередь состоит из свойств UserAnswer, Question, CorrectAnswer.

CorrectAnswer получает значение при генерации вопросов. В качестве ответа записывается существительное, к которому был задан вопрос.

Таким образом после завершения теста клиент получает целый набор информации, предварительно отправив на сервер ответы, которые пользователь дал во время теста. Сервер сравнил каждый ответ с корректным посчитал количество заработанных баллов и вернул клиенту TestResult.

5.8 Контрольный тест

В программу была загружена полная версия документа политики безопасности, взятой из открытого доступа в Интернете, со следующими характеристиками файла: 307 КБ, 7900 слов, 29 страниц. Время обработки всего текста для получения вопросов составило около 1 часа. Результаты генерации: 7,7% корректных вопросов (8 из 104). Отсев корректных вопросов проводился вручную.

Список корректных вопросов:

1. Заявка согласуется с администратором информационной безопасности и передается системному администратору (программисту) на исполнение. Вопрос: с кем согласуется заявка?
2. Копии исполненных заявок хранятся у системного администратора. Вопрос: у кого хранятся копии исполненных заявок?
3. В случае обнаружения подсистемой защиты информации факта нарушения информационной безопасности или осуществления НСД к защищаемым информационным ресурсам ИС рекомендуется уведомить администратора информационной безопасности и/или начальника информационного отдела. Вопрос: в случае чего уведомить администратора информационной безопасности информационного отдела?
4. Типовое положение об органе по аттестации объектов информатизации по требованиям безопасности информации (утверждено приказом председателя Государственной технической комиссии при Президенте Российской Федерации т 5 января 1996 г. Вопрос: о чем утверждено типовое положение?

5. Для решения задач контроля защищенности ИС используются инструментальные средства для тестирования реализованных в составе СЗИ ИС Управления средств и функций защиты. Вопрос: для чего используются инструментальные средства?
6. Запрещается использование указанных АРМ другими пользователями без согласования с администратором информационной безопасности Управления. Вопрос: без чего запрещается использование указанных АРМ?
7. В случае невозможности исполнения инициатору заявки направляется мотивированный отказ с приложением заявки. Вопрос: в случае чего направляется мотивированный отказ?
8. Исполненные заявки передаются администратору информационной безопасности. Вопрос: кому передаются исполненные заявки?

При этом стоит отметить, что многие вопросы были не учтены как корректные именно из-за отсутствия возможности определить контекст по самому вопросу. То есть, не зная исходного предложения на подобные вопросы просто невозможно ответить.

Выборка некорректных вопросов:

1. Программно-технических и организационных решений, позволяющих свести к минимуму возможные потери от технических аварий и ошибочных действий персонала. Вопросы:
    - К чему свести персонала?
    - От чего свести персонала?
    - Чего свести персонала?
2. Для этого в Управлении выполняются следующие мероприятия: определяется порядок работы с документами, образцами изделиями. Вопрос: в чем выполняются этого?
3. Они могут впоследствии использоваться: для восстановления полномочий пользователей после аварий в ИС Управления; для контроля правомерности наличия у конкретного пользователя прав доступа к информационному ресурсу; тем или иным ресурсам системы при разборе конфликтных ситуаций; для проверки системным администратором правильности настройки средств разграничения доступа к ресурсам системы. Вопрос: для чего использоваться впоследствии в последствии они впоследствии они ис управления контроля правомерности наличия конкретного пользователя прав доступа информационному ресурсу тем или иным ресурсам системы разборе конфликтных ситуаций?

4. В заявке указывается: должность, фамилия. Вопрос: в чем указывается?
5. По окончании внесения изменений в заявке делается отметка о выполнении задания за подписями исполнителей. Вопрос: по чем делается отметка выполнении задания подписями исполнителей?

6. Руководство пользователя

6.1 Введение

Программа позволяет проводить контроль знаний пользователя об изученном тексте политики безопасности.

Программа предоставляет пользователю следующие возможности:

1. выбрать любой текстовый файл формата doc, docx, pdf, txt;
2. получить на его основе список вопросов;
3. пройти тест, состоящий из полученных вопросов, отслеживая свой прогресс (увеличение полосы прогресса по мере приближения пользователя к последнему вопросу);
4. получить результаты прохождения теста;
5. начать тест заново.

6.2 Назначение и условия применения

Приложение написано на языке C# и требует установленного на компьютере .NET Core 5.0 (Runtime версия). Дисковой памяти для запуска программы потребуется не менее 18,2 Мб памяти. Приложение разработано с учетом возможности функционирования как по локальной, так и по Интернет сети.

6.3 Подготовка к работе

Запустите файл серверной части приложения PoliticsTester.Server.exe, откроется отладочное окно в браузере. Взаимодействовать с ним пользователю необходимости нет. Затем запустите PoliticsTester.Client.exe.

6.4 Описание операций

В открывшемся окне программы, вид которого представлен на рисунке 34, необходимо нажать кнопку «открыть файл». Откроется диалоговое окно (рисунок 35), в котором можно выбрать текстовый документ.

После выбора файла необходимо нажать кнопку «Получить вопросы», после чего кнопки «–>», «Заново» и «Завершить» станут доступны для нажатия, а в поле «Вопрос:» появится первый вопрос из списка полученных. Общее число вопросов видно в верхнем правом углу окна приложения (рисунок 36).

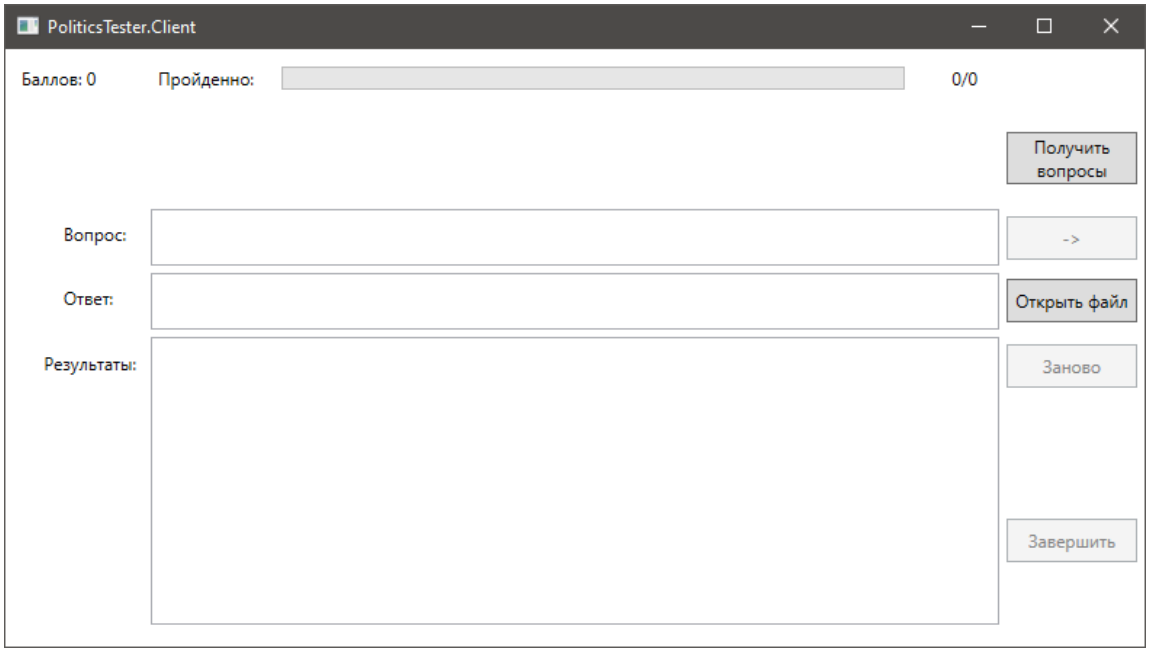

Рисунок 34 – Окно клиента при первичном запуске

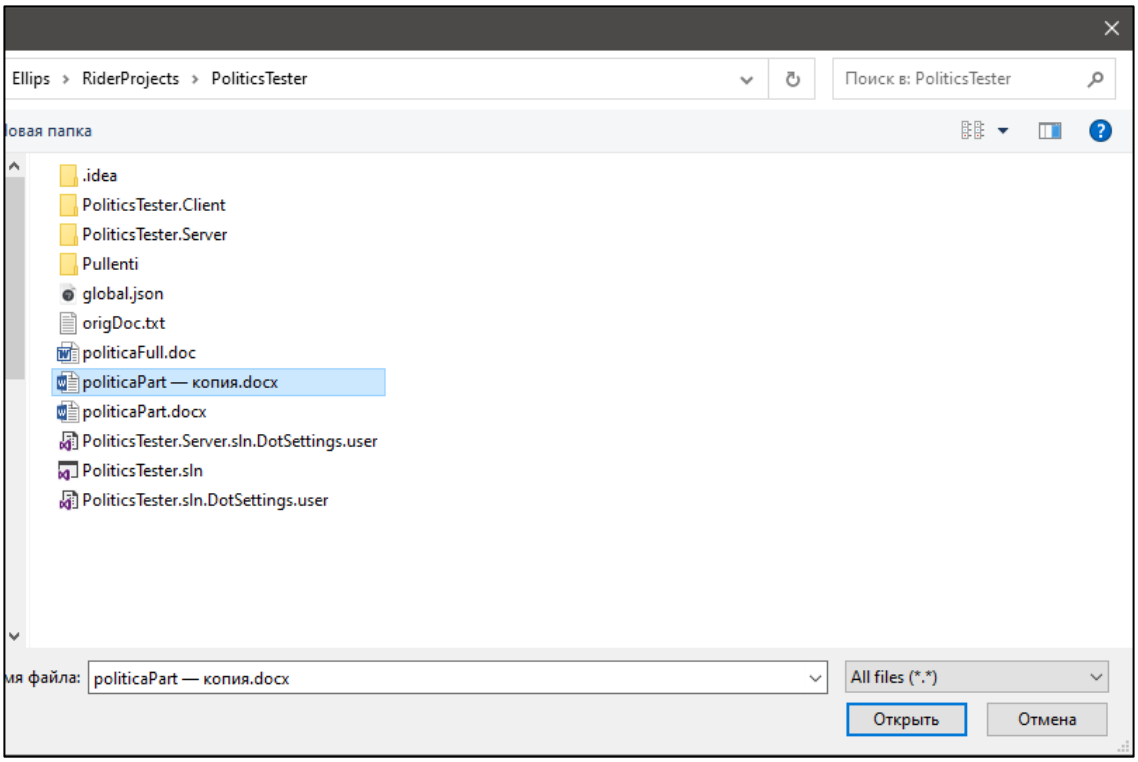

Рисунок 35 – Выбор текстового файла

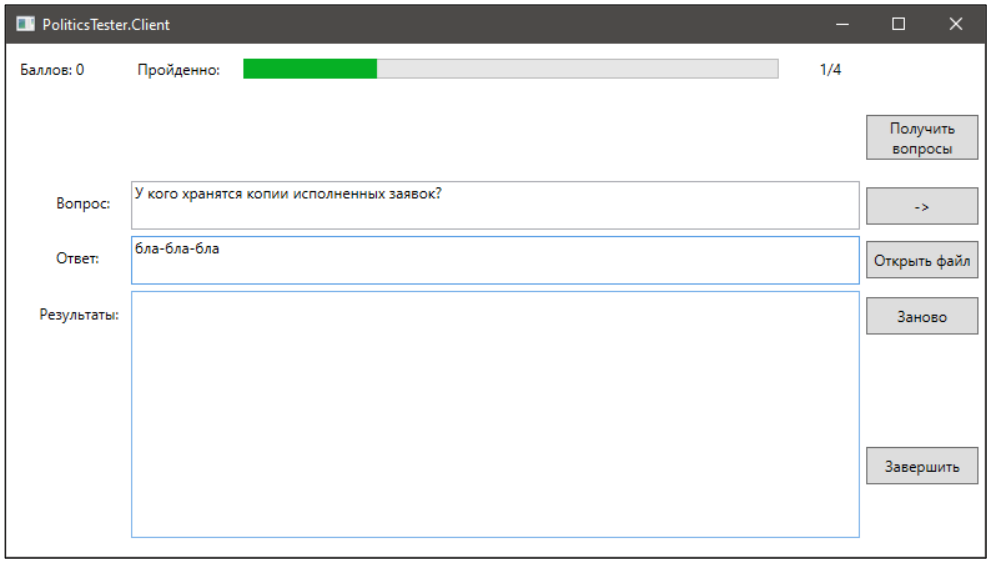

Рисунок 36 – Вид клиентского окна после получения вопросов

Заполнив поле «Ответ:», для перехода к следующему вопросу нажмите кнопку «–>». Закончить тест можно в любой момент. После окончания всех вопросов или нажатия кнопки «Завершить» в поле «Результаты:» будет выведено сообщение с количеством баллов и списком вопросов, корректных ответов и ответов пользователя (рисунок 37).

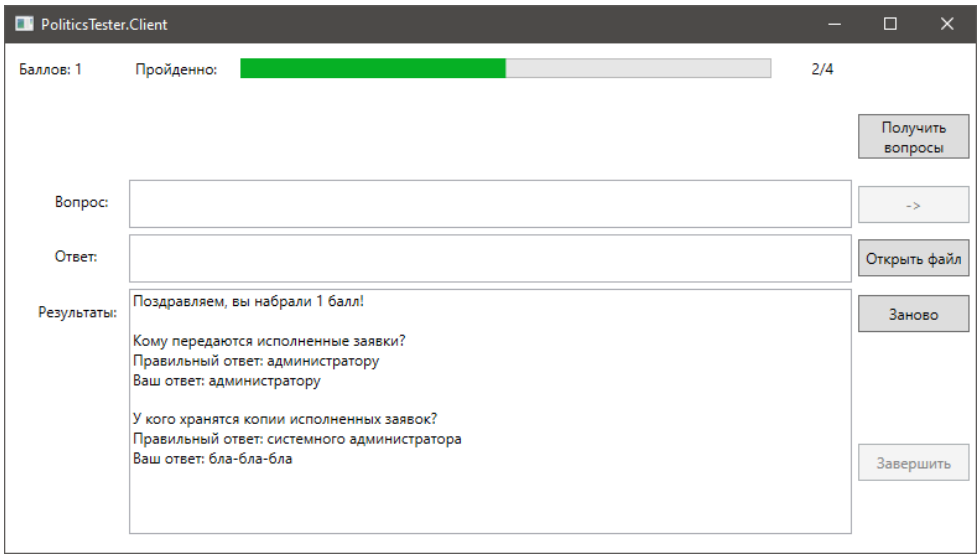

Рисунок 37 – Окно клиента после завершения теста

Если необходимо создать тест по другому документу, то выберите его через «Открыть файл», затем нажмите «Получить вопросы» и данные полей будут обновлены.

7. Место работы в комплексной защите объекта информатизации

Использование разработанной программы актуально для контроля знаний сотрудников политики ИБ, чтобы автоматизировать процесс тестирования и создания вопросов по имеющемуся текстовому материалу. А также чтобы повысить мотивацию сотрудников к изучению политик, посредством игрофицированного подхода. Данное средство необходимо в виду наличия такой проблемы как, внутренний нарушитель, который возникает во многом из-за незнания или некачественного освоения политики безопасности.

Относительно нормативно правовых актов, разработанная программа и регламент её использования для сотрудников компаний относятся к техническим и организационным категориям мер защиты информации, которые определены в ФЗ-149 (ст. 16 п. 1) [45].

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы была достигнута основная цель: разработка программы, генерирующей тесты с вопросами, созданными из текста политики безопасности.

Для достижения этой цели были реализованы следующие задачи:

1. изучены аспекты создания графа знаний, лингвистического анализа, генерации вопросов, а также игрофикации;
2. проведен анализ и выбор средств, необходимых для разработки программы;
3. создан функционал, позволяющий извлекать текст из основных текстовых форматов при помощи библиотеки Sautinsoft.Document;
4. написан алгоритм построения семантического графа с использование библиотеки Pullenti;
5. написан алгоритм генерации вопросов;
6. разработан пользовательский адаптивный интерфейс;
7. реализована клиент-серверная архитектура;
8. интегрирована бальная система.

Получившаяся программа хоть и способна выдать для теста несколько корректных вопросов (около 7,7% от общего числа), но она далека от идеала и есть ещё не мало аспектов, которые следуют улучшить или даже вовсе переделать. В частотности необходимо: попробовать использовать уже существующие алгоритмы генерации вопросов, такие как DQG, RefNet; интегрировать более богатую систему игрофикации; реализовать обработку текста на естественном языке именно через граф знаний (Neo4J, Graph Aware и др.), а не только через семантический граф. При этом стоит отменить, что готовых аналогов задуманному приложению практически нет. Таким образом данная работа является одним из первых шагов к автоматизированному контролю знаний сотрудников о тех или иных документах (не только о политиках безопасности) с применением методов игрофикации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Утечки информации ограниченного доступа: отчет за 9 месяцев 2020. [Электронный ресурс]. – Экспертно-аналитический центр InfoWatch, 2021 г. – URL: https://d-russia.ru/wp-content/uploads/2020/12/infowatch_2020_9_monts_data_leak.pdf (дата обращения: 09.03.2022).

2. Information security awareness, a tool to mitigate information security risk: a literature review / O.S. Dada, E.A. Irunokhai, C.J. Shawulu, A.M. Nuhu, E.E. Daniel // Innovative Journal of Science. – 2021. – Vol. 3, N 3. – P. 29–54. – URL: https://www.researchgate.net/publication/353849520_Information_Security_Awareness_a_Tool_to_Mitigate_Information_Security_Risk_a_Literature_Review (accessed: 09.03.2022).

3. Галкин М. Курс по графам знаний (Knowledge Graphs) и как их готовить в 2021 году [Электронный ресурс]: курс видео лекций / М. Галкин., В. Сафронов, С. Иванов, Д. Муромцев – ИТМО, 2021. – URL: https://migalkin.github.io/kgcourse2021/ (дата обращения: 13.02.2022).

4. RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax [Электронный ресурс]. – The Internet Society, 2005. – URL: https://datatracker.ietf.org/doc/html/rfc3986#section-1.1.3 (accessed: 20.05.2022).

5. Schrader Bess. What’s the Difference Between an Ontology and a Knowledge Graph? [Электронный ресурс]. – Enterprise Knowledge, 2020. – URL: https://enterprise-knowledge.com/whats-the-difference-between-an-ontology-and-a-knowledge-graph/ (accessed: 14.02.2022).

6. RDF Vocabulary Description Language 1.0: RDF Schema [Электронный ресурс] // W3C. – 2003. – URL: https://www.w3.org/2001/sw/RDFCore/Schema/200212bwm/#ch_type (accessed: 02.03.2022).

7. Subrata Paul. A Review on Graph Database and its representation [Электронный ресурс] / Subrata Paul, Chandan Koner and Anirban Mitra // International Conference on Recent Advances in Energy-efficient Computing and Communication (ICRAECC). – 2019. – URL: https://www.researchgate.net/publication/339266779_A_Review_on_Graph_Database_and_its_representation (accessed: 03.01.2022). – DOI: 10.1109/ICRAECC43874.2019.8995006.

8. Sasaki B.M. Graph Databases for Beginners: Other Graph Technologies [Электронный ресурс] - Neo4j, 2018. - URL: https://neo4j.com/blog/other-graph-database-technologies/ (accessed 15.02.2022).

9. List of SPARQL implementations [Электронный ресурс] // Wikipedia, the free encyclopedia. – URL: https://en.wikipedia.org/wiki/List_of_SPARQL_implementations (accessed: 26.03.2022).

10. Graph Query Language Comparison Series - Gremlin vs Cypher vs nGQL [Электронный ресурс]. – 2020. – URL: https://nebula-graph.io/posts/graph-query-language-comparison-cypher-gremlin-ngql (accessed: 15.03.2022).

11. Gurin V. S. Knowledge Graph Essentials and Key Technologies [Электронный ресурс] / E. V. Kostrov, Yu. Yu. Gavrilenko, D. F. Saada, E. A. Ilyushin, I. V. Chizhov // Sovremennye informacionnye tehnologii i ITobrazovanie = Modern Information Technologies and IT-Education. – 2019. С. 932-944, т.15(4). – URL: https://cyberleninka.ru/article/n/predstavlenie-znaniy-v-vide-grafa-osnovnye-tehnologii-i-podhody (accessed: 10.03.2022) – DOI: 10.25559/SITITO.15.201904.932-944.

12. Pan S. J. Transfer Joint Embedding for Cross-Domain named entity recognition [Электронный ресурс] / Pan S. J., Toh Z., Su, J. // Institute for Infocomm Research. – 2013. – URL: https://personal.ntu.edu.sg/sinnopan/publications/[TOIS13]Transfer%20Joint%20Embedding%20for%20Cross-Domain%20Named%20Entity%20Recognition.pdf (accessed: 12.03.2022). DOI: http://dx.doi.org/10.1145/2457465.2457467.

13. Daniel Jurafsky. Coreference Resolution [Электронный ресурс] / Daniel Jurafsky, James H. Martin // Speech and Language Processing. – 2021. – chapter 21. – URL: https://web.stanford.edu/~jurafsky/slp3/ (accessed: 08.03.2022).

14. Каримов Р.И. NLP - обработка естественного языка. Лингвистический метод обработки текста [Электронный ресурс] // Материалы XV Всероссийской молодежной научной конференции: в 7 томах. – Уфимский государственный авиационный технический университет, 2021 г. – С. 250-254. – URL: https://www.elibrary.ru/item.asp?id=47305095 (дата обращения: 15.02.2022).

15. Белов С.Д. Обзор методов автоматической обработки текстов на естественном языке [Электронный ресурс] / Белов С.Д.; Зрелова Д.П.; Зрелов П.В.; Кореньков В.В. // Сетевое научное издание «Системный анализ в науке и образовании». – 2020 г. – С. 8-22. – URL: https://www.elibrary.ru/item.asp?id=44288349 (дата обращения: 05.03.2022). DOI: 10.37005/2071-9612-2020-3-8-22.

16. Alexander Lepe. DeepMorphy. Морфологический анализатор для русского языка на C# для .NET [Электронный ресурс]. – 2019. – URL: https://github.com/lepeap/DeepMorphy (дата обращения: 22.04.2022).

17. Кузнецов К.И. SDK Pullenti. Библиотека лингвистического анализа [Электронный ресурс]. – 2013. – URL: https://www.pullenti.ru/Document (дата обращения: 20.03.2022).

18. Overview: Extracting and serving feature embeddings for machine learning [Электронный ресурс] // Center, Cloud Architecture. – 2019. – URL:

https://cloud.google.com/architecture/overview-extracting-and-serving-feature-embeddings-for-machine-learning (accessed: 27.03.2022).

19. Бурина Т.А. Обработка естественного языка, поиск и извлечение / Бурина Т.А., Евдокимов М.С // Научно-практические исследования. – 2020. – 5-2 (28). – С. 57-59. – URL: https://www.elibrary.ru/item.asp?id=42831177 (дата обращения: 25.03.2022).

20. Клыч А.А. Разработка подхода для генерирования вопросов / Клыч А.А., Коршунов А.Г., Стативко Р.У. // Актуальные проблемы теории и практики обучения физико-математическим и техническим дисциплинам в современном образовательном пространстве: IV Всероссийская (с международным участием) научно-практическая конференция. – Курск, 2020. – С. 367–373. – URL: https://elibrary.ru/item.asp?id=44845075.

21. Белянова М.А. Автоматическая генерация вопросов на основе текстов и графов знаний [Электронный ресурс] / М.А. Белянова, Г.И. Ревунков, Г.И. Афанасьев, Ю.Е. Гапанюк // Динамика сложных систем – XXI век. – 2020. – Т. 14, № 4. – С. 55–64. – DOI: 10.18127/j19997493-202004-06.

22. Вэньцзу Л. Онтологический подход к автоматической генерации вопросов в интеллектуальных обучающих системах [Электронный ресурс] // Доклады БГУИР. – 2020. – Т. 18, № 5. – С. 44–52. – DOI: 10.35596/1729-7648-2020-18-5-44-52.

23. Переходько И.В., Мячин Д.А. Оценка качества компьютерного перевода [Электронный ресурс] // Вестник Оренбургского государственного университета. – 2017. – № 2 (202). – С. 92–96. – URL: https://elibrary.ru/item.asp?id=28904673 (дата обращения: 09.03.2022).

24. Легостина М.С. Метрики оценки качества машинного перевода [Электронный ресурс] // Инноватика-2019: сборник материалов XV Международной школы-конференции студентов, аспирантов и молодых ученых. – Томск, 2019. – URL: https://elibrary.ru/item.asp?id=39222633 (дата обращения: 09.03.2022).

25. Андреичев М.Д., Ференец А.А. Разработка программного комплекса генерации вопросов по заданным субъектам при помощи семантической сети [Электронный ресурс] // Электронные библиотеки. – 2019. – Т. 22, № 2. – С. 68–71. – DOI: 10.26907/1562-5419-2019-22-2-68-94.

26. Chen Y. Toward subgraph guided knowledge graph question generation with graph neural networks [Электронный ресурс] / Chen Y., Wu L., Zaki M.J. // Journal of Latex Class Files. – 2015. – Vol. 14, N 8. – URL: https://arxiv.org/abs/2004.06015 (accessed: 09.03.2022).

27. Chai Z., Wan X. Learning to ask more: semi-autoregressive sequential question generation under dual-graph interaction [Электронный ресурс] // Proceedings of the 58th Annual Meeting of

the Association for Computational Linguistics. – Association for Computational Linguistics, 2020. – P. 225–237. – DOI: 10.18653/v1/2020.acl-main.21.

28. Let’s ask again: refine network for automatic question generation [Электронный ресурс] / P. Nema, A.K. Mohankumar, M.M. Khapra, B. Ravindran, B.V. Srinivasan // Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). – Hong Kong, China, 2019. – DOI: 10.18653/V1/D19-1326.

29. Semantic graphs for generating deep questions [Электронный ресурс] / L. Pan, Y. Xie, Y. Feng, T.-S. Chua, M.-Y. Kan // Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. – Association for Computational Linguistics, 2020. – P. 1463–1475. – DOI: 10.18653/v1/2020.acl-main.135.

30. Ю-Кай Чоу. Геймифицируй это. Как стимулировать клиентов к покупке, а сотрудников – к работе. – Москва: Эксмо, 2022. — 400 с.: ил. — (Экономика эмоций).

31. Татаринов К.А. Геймфикация в обучении студентов. // 1(26), 2019 г., Т. 8. – URL: https://cyberleninka.ru/article/n/geymifikatsiya-v-obuchenii-studentov (дата обращения: 15.04.2022). – DOI: 10.26140/bgz3-2019-0801-0074.

32. Кэвин Вербах. Вовлекай и властвуй. / Кэвин Вербах, Дэн Хантер — М.: Манн, Иванов и Фербер, 2015. — 224 с.

33. Áron Tóth. The introduction of gamification: A review paper about the applied. [Электронный ресурс] / Áron Tóth, Sarolta Tóvölgyi // Conference: 2016 7th IEEE International Conference on Cognitive Infocommunications (CogInfoCom). – URL: https://www.researchgate.net/publication/312110882_The_introduction_of_gamification_A_review_paper_about_the_applied_gamification_in_the_smartphone_applications (accessed: 14.04.2022) – DOI: 10.1109/CogInfoCom.2016.7804551.

34. Томита-парсер. [Электронный ресурс]. – URL: https://yandex.ru/dev/tomita/ (дата обращения: 23.02.2022).

35. Natasha. [Электронный ресурс]. – URL: https://github.com/natasha/natasha (accessed: 23.02.2022).

36. Catalyst NLP. [Электронный ресурс]. – URL: https://github.com/curiosity-ai/catalyst (accessed: 23.02.2022).

37. Stanza. [Электронный ресурс]. – URL: https://stanfordnlp.github.io/stanza/ (accessed: 23.02.2022).

38. Nebula Graph. [Электронный ресурс]. – URL: https://nebula-graph.io/ (accessed: 20.02.2022).

39. Hume Features for Mission Critical Graph Analytics. graphaware.com. [Электронный ресурс]. – URL: https://graphaware.com/features/ (дата обращения: 20.02.2022).

40. Neo4j Graph Database. neo4j.com. [Электронный ресурс]. – URL: https://neo4j.com/product/neo4j-graph-database/ (дата обращения: 20.02.2022).

41. Лесных И. Сравнение инструментов для GRAPH MINING [Электронный ресурс] веб-сайт «newtechaudit.ru», сообщество data аналитиков и инженеров. – URL: https://newtechaudit.ru/sravnenie-instrumentov-dlya-graph-mining/ (дата обращения: 21.02.2022).

42. Networkx [Электронный ресурс]. – URL: https://networkx.org/documentation/stable/reference/readwrite/index.html (дата обращения: 21.02.2022).

43. Gephi [Электронный ресурс]. – URL: https://gephi.org/ (дата обращения: 21.02.2022).

44. Graphviz [Электронный ресурс]. – URL: https://graphviz.org/docs/outputs/ (дата обращения: 21.02.2022).

45. Федеральный закон от 27.07.2006 № 149 «Об информации, информационных технологиях и о защите информации». – URL: http://www.consultant.ru/document/cons_doc_LAW_61798/ (дата обращения: 15.05.2022).

ПРИЛОЖЕНИЕ А

Листинг серверной программы

Program.cs

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace PoliticsTester.Server
{
   public class Program
   {
      public static void Main(string[] args)
      {
         CreateHostBuilder(args).Build().Run();
      }

      public static IHostBuilder CreateHostBuilder(string[] args) =>
         Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder => {
webBuilder.UseStartup<Startup>(); });
   }
}
```

Startup.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.OpenApi.Models;
using PoliticsTester.Server.Data;
using PoliticsTester.Server.Services;

namespace PoliticsTester.Server
{
   public class Startup
   {
      public Startup(IConfiguration configuration)
      {
         Configuration = configuration;
      }

      public IConfiguration Configuration { get; }

      // This method gets called by the runtime. Use this method to add
services to the container.
      public void ConfigureServices(IServiceCollection services)
      {
         services.AddSingleton<QuestionSets>();
         services.AddScoped<ExtractorSentences>();
         services.AddScoped<SemanticAnalyser>();
         services.AddScoped<QuestionsGenerator>();
         services.AddControllers();
         services.AddSwaggerGen(c =>
         {
            c.SwaggerDoc("v1", new OpenApiInfo {Title =
"PoliticsTester.Server", Version = "v1"});
         });
      }

      // This method gets called by the runtime. Use this method to
configure the HTTP request pipeline.
      public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)
      {
         if (env.IsDevelopment())
         {
            app.UseDeveloperExceptionPage();
            app.UseSwagger();
            app.UseSwaggerUI(c =>
c.SwaggerEndpoint("/swagger/v1/swagger.json", "PoliticsTester.Server
v1"));
         }

         app.UseHttpsRedirection();

         app.UseRouting();

         app.UseAuthorization();

         app.UseEndpoints(endpoints => { endpoints.MapControllers(); });
      }
   }
}
```

PoliticsController.cs

```
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using PoliticsTester.Server.Data;
using PoliticsTester.Server.Models;
using PoliticsTester.Server.Services;

namespace PoliticsTester.Server.Controllers
{
   [ApiController]
   [Route("[controller]")]
   public class PoliticsController : ControllerBase
   {
      private readonly QuestionSets _questionSets;
      private readonly ExtractorSentences _extractorSentences;
      private readonly SemanticAnalyser _semanticAnalyser;
      private readonly QuestionsGenerator _questionsGenerator;

      public PoliticsController(QuestionSets questionSets,
ExtractorSentences extractorSentences,
         SemanticAnalyser semanticAnalyser, QuestionsGenerator
questionsGenerator)
      {
         _questionSets = questionSets;
         _extractorSentences = extractorSentences;
         _semanticAnalyser = semanticAnalyser;
         _questionsGenerator = questionsGenerator;
      }

      [HttpPost]
      [Route("questions")]
      public IEnumerable<string> GetQuestions(TextModel textModel)
      {
         var sentences =
_extractorSentences.GetSentences(textModel.Text);
         var questionsAndAnswers = new List<QA>();

         foreach (var sentence in sentences)
         {
            var basicResult = _semanticAnalyser.ExtractEntities(sentence);
```

```
            var semDoc =
_semanticAnalyser.ExtractRelationships(basicResult);
            var currentQuestionsAndAnswers =
_questionsGenerator.GenerateQuestions(semDoc);

questionsAndAnswers.AddRange(currentQuestionsAndAnswers);
        }

        _questionSets.SetsValues = questionsAndAnswers;

        return questionsAndAnswers.Select(qa => qa.Question);
    }

    [HttpGet]
    [Route("testresult")]
    public TestResult GetTestResult([FromQuery]UserAnswers
userAnswers)
    {
        var answers = userAnswers.Answers[0].Split(',').ToList();
        int points = 0;
        List<QuestionInformation> questionResults = new();

        for (int i = 0; i < answers.Count; i++)
        {
            questionResults.Add(new QuestionInformation(){UserAnswer
= answers[i], Question =  _questionSets.SetsValues[i].Question,
CorrectAnswer = _questionSets.SetsValues[i].Answer});

            if (answers[i] == _questionSets.SetsValues[i].Answer)
            {
                points++;
            }
        }

        return new TestResult()
        {
            Points = points,
            QuestionResults = questionResults.ToArray()
        };
    }
  }
}
```

UserAnswers.cs

```
using System.Collections.Generic;

namespace PoliticsTester.Server.Models
{
  public class UserAnswers
  {
    public List<string> Answers { get; set; }
  }
}
```

TextModel.cs

```
namespace PoliticsTester.Server.Models
{
    public class TextModel
    {
        public string Text { get; set; }
    }
}
```

TestResult.cs

```
namespace PoliticsTester.Server.Models
{
    public class TestResult
    {
        public int Points { get; set; }

        public QuestionInformation[] QuestionResults
{ get; set; }
    }
}
```

QuestionInformation.cs

```
namespace PoliticsTester.Server.Models
{
    public class QuestionInformation
    {
        public string UserAnswer { get; set; }

        public string Question { get; set; }

        public string CorrectAnswer { get; set; }
    }
}
```

QA.cs

```
namespace PoliticsTester.Server.Models
{
    public class QA
    {
        public string Question { get; set; }

        public string Answer { get; set; }
    }
}
```

QuestionSets.cs

```
using System.Collections.Generic;
using PoliticsTester.Server.Models;

namespace PoliticsTester.Server.Data
{
  public class QuestionSets
  {
    public List<QA> SetsValues { get; set; } = new List<QA>();
  }
}
```

ExtractorSentences.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace PoliticsTester.Server.Services
{
  public class ExtractorSentences
  {
    public string[] GetSentences(string text)
    {
      text = DeleteSpecialSymbols(text);
      text = ReplaceDotsBetweenNumbers(text);
```

```
        char[] separators = {'|'};

        List<string> splitedText = new List<string>();

        text = text.Replace(".", ".|").Replace("?", "?|").Replace("!",
"!|").Replace("?|!|", "?!|")
            .Replace(".|.|.|", "...|");

        splitedText.AddRange(text.Split(separators,
StringSplitOptions.RemoveEmptyEntries));

        splitedText = splitedText.Select(sent => sent.Trim()).ToList();

        DeleteExcessSentences(splitedText);

        return splitedText.ToArray();
    }

    private string ReplaceDotsBetweenNumbers(string text)
    {
      //Пропускуем первый и последний символы,
      //т.к. они определенно не могут быть точками между цифрами
      for (int i = 1; i < text.Length - 1; i++)
      {
        if (text[i] == '.')
        {
          char leftChar = text[i - 1];
          char RightChar = text[i + 1];
          if (Char.IsDigit(leftChar) && Char.IsDigit(RightChar))
          {
            StringBuilder sb = new StringBuilder(text);
            sb.Replace(".", ",", i, 1);
            text = sb.ToString();
          }
        }
      }

      return text;
    }

    private void DeleteExcessSentences(List<string> sentences)
    {
      for (int i = 0; i < sentences.Count; i++)
      {
        int n = 0;
        foreach (char c in sentences[i])
          if (c == ' ' || c == '.')
            n++;
        if (n < 4)
        {
          sentences.RemoveAt(i);
          i--; //Чтобы не пропустить предложение ставшее на
место удаленного
        }
      }
    }

    private string DeleteSpecialSymbols(string text)
    {
      return text.Replace("\r\n", " ");
    }
  }
}
```

SemanticAnalyser.cs

```
using System.Collections.Generic;
using Pullenti.Ner;
using Pullenti.Ner.Core;
using Pullenti.Semantic;

namespace PoliticsTester.Server.Services
{
  public class SemanticAnalyser
  {
    public Dictionary<string, int> nameGroups;
    public List<Referent> namedEntities;
    public SemDocument semDoc;
    private Processor processor;

    public AnalysisResult ExtractEntities(string sentence)
    {
      Pullenti.Sdk.InitializeAll();
      processor = ProcessorService.CreateProcessor(); //Создание
экземпляра процессора со стандартными анализаторами

      var basicResult = processor.Process(new
SourceOfAnalysis(sentence));  //Базовый анализ, включающий в себя
NER

      ExtractNounPhraseTokens(basicResult);  //Получение токенов
"Именные группы"

      return basicResult;
    }

    private void ExtractNounPhraseTokens(AnalysisResult basicResult)
    {
      nameGroups = new Dictionary<string, int>();
      for (Token t = basicResult.FirstToken; t != null; t = t.Next)
      {
        NounPhraseToken npt = NounPhraseHelper.TryParse(t,
NounPhraseParseAttr.ParseVerbs);

        if (npt == null) continue;

        string normal = npt.GetSourceText();
        //string normal = npt.GetNormalCaseText(null,
MorphNumber.Undefined, MorphGender.Undefined, true); //
нормализуем к единственному числу

        if (!nameGroups.ContainsKey(normal))
          nameGroups.Add(normal, 1);
        else
          nameGroups[normal]++;

        t.Kit.EmbedToken(npt);  // встраиваем
        // теперь вместо npt.BeginToken (=t) ... npt.EndToken в
последовательности один npt

        t = npt.EndToken; // перемещаемся в конец метатокена
      }
    }

    public SemDocument ExtractRelationships(AnalysisResult
basicResult)
    {
      return SemanticService.Process(basicResult);
    }
  }
}
```

QuestionsGenerator.cs

```
using System.Collections.Generic;
using System.Linq;
using PoliticsTester.Server.Models;
using Pullenti.Morph;
using Pullenti.Ner;
using Pullenti.Semantic;
using Pullenti.Semantic.Utils;

namespace PoliticsTester.Server.Services
{
    public class QuestionsGenerator
    {
        // private float weight;
        // private List<MorphBaseInfo> nouns;
        private string questionSentence;
        private List<string> parts;
        private List<CaseQuestionForms> caseQuestionForms;

        class CaseQuestionForms
        {
            public MorphCase typeCase;
            public string animatedQuestion;
            public string inanimateQuestion;
        }

        private void InitCaseQuestionForms()
        {
            caseQuestionForms = new List<CaseQuestionForms>();
            caseQuestionForms.Add(new CaseQuestionForms()
                {typeCase = MorphCase.Nominative, animatedQuestion =
"кто", inanimateQuestion = "что"});
            caseQuestionForms.Add(new CaseQuestionForms()
                {typeCase = MorphCase.Genitive, animatedQuestion = "кого",
inanimateQuestion = "чего"});
            caseQuestionForms.Add(new CaseQuestionForms()
                {typeCase = MorphCase.Accusative, animatedQuestion =
"кого", inanimateQuestion = "что"});
            caseQuestionForms.Add(new CaseQuestionForms()
                {typeCase = MorphCase.Dative, animatedQuestion = "кому",
inanimateQuestion = "чему"});
            caseQuestionForms.Add(new CaseQuestionForms()
                {typeCase = MorphCase.Instrumental, animatedQuestion =
"кем", inanimateQuestion = "чем"});
            caseQuestionForms.Add(new CaseQuestionForms()
                {typeCase = MorphCase.Prepositional, animatedQuestion =
"ком", inanimateQuestion = "чем"});
        }

        private List<SemLink> CollectAllTargetNouns(SemDocument
semDoc)
        {
            var targetNounLinks = new List<SemLink>();

            if (semDoc.Blocks.Count > 0)
            {
                foreach (var block in semDoc.Blocks)
                foreach (var fragment in block.Fragments)
                foreach (var link in fragment.Graph.Links)
                    if (link.Source.Tokens.Count > 0 && link.Question != null
&& link.Target.Tokens.Count > 0)
                    {
                        foreach (MetaToken t in link.Source.Tokens)
                            if (!t.Morph.Class.IsVerb)
                                return targetNounLinks;
                        foreach (MetaToken t in link.Target.Tokens)
                            if (t.Morph.Class.IsNoun)
                                targetNounLinks.Add(link);
                    }
            }

            return targetNounLinks;
        }

        private string CreateQuestionSentence(SemLink nounLink)
        {
            InitCaseQuestionForms();
            questionSentence = "";
            parts = new List<string>();

            //Добавление предлога (если он есть) перед вопросительным
местоимением
            if (nounLink.Preposition != null && nounLink.Preposition != "")
                questionSentence += nounLink.Preposition.ToLower() + " ";

            //Выделение из MetaToken'а состоящего из нескольких слов
слова, являющегося существительным
            string str = nounLink.Target.Tokens[0].GetNormalCaseText();
            List<string> strs = str.Split(" ").ToList();

            Dictionary<string, MorphWordForm> mwfs = new
Dictionary<string, MorphWordForm>();
            foreach (string s in strs)
                mwfs.Add(s, MorphologyService.GetWordBaseInfo(s));

            string noun = "";
            foreach (var m in mwfs)
                if (m.Value.Class.ToString() == MorphClass.Noun.ToString())
                    noun = m.Key;

            CaseQuestionForms curQuestionForms = new
CaseQuestionForms();
            //Подбор одушевленного или неодушевленного вопроса
            if (!DerivateService.IsAnimated(noun.ToUpper(),
MorphLang.RU))
            {
                curQuestionForms = caseQuestionForms.Find(item =>
                    item.typeCase.ToString() ==
nounLink.Target.Tokens[0].Morph.Case.ToString());
                if (curQuestionForms != null)
                    questionSentence += curQuestionForms.inanimateQuestion;
            }
            else
            {
                curQuestionForms = caseQuestionForms.Find(item =>
                    item.typeCase.ToString() ==
nounLink.Target.Tokens[0].Morph.Case.ToString());
                if (curQuestionForms != null)
                    questionSentence += curQuestionForms.animatedQuestion;
            }

            //Добавление слова, являющегося исчтоником вопроса
            questionSentence += " " +
nounLink.Source.Tokens[0].GetSourceText().ToLower();

            //Добавление всех слов входящих в цепочку семантически
связанных элементов
            foreach (SemLink linkFrom in nounLink.Source.LinksFrom)
            {
                if (linkFrom.Target.ToString() != "?")
                {
                    parts.Add(" " +
linkFrom.Target.Tokens[0].GetSourceText().ToLower());
                    questionSentence += GetNextWords(linkFrom);
                }
```

```
            else if (linkFrom.Typ == SemLinkType.Pacient)
            {
                parts.Add(" " +
linkFrom.Target.Tokens[0].GetSourceText().ToLower());
                questionSentence += GetNextWords(linkFrom);
            }
        }

        questionSentence = questionSentence.TrimEnd() + "?";
        RaiseRegisterOfFirstChar();
        return questionSentence;
    }

    private void RaiseRegisterOfFirstChar()
    {
        string firstChar = questionSentence[0].ToString().ToUpper();
        questionSentence = questionSentence.Remove(0, 1);
        questionSentence = questionSentence.Insert(0, firstChar);
    }

    private string GetNextWords(SemLink newLink)
    {
        if (newLink.Target.LinksFrom.Count > 0)
        {
            foreach (SemLink linkFrom2 in newLink.Target.LinksFrom)
            {
                if (linkFrom2.Target.Tokens.Count > 0)
                {
                    if (linkFrom2.Typ == SemLinkType.Detail ||
linkFrom2.Typ == SemLinkType.Pacient)
                        parts.Add(" " +
linkFrom2.Target.Tokens[0].GetSourceText().ToLower());
                    if (linkFrom2.Typ == SemLinkType.Participle)
                        if (parts.Count > 0)
                            parts.Insert(parts.Count - 1,
                                " " +
linkFrom2.Target.Tokens[0].GetSourceText().ToLower());
                        else
                            parts.Add(" " +
linkFrom2.Target.Tokens[0].GetSourceText().ToLower());

                    if (linkFrom2.Target.Tokens[0].GetSourceText() !=
                        newLink.Target.Tokens[0]
                            .GetSourceText()) //Иногда в семантической
цепочки два триплета ссылаются друг на друга и получается
закольцованность
                        return GetNextWords(linkFrom2);
                }
            }
        }

        string joinedParts = "";
        for (int i = 0; i < parts.Count; i++)
        {
            joinedParts += parts[i];
        }

        return joinedParts;
    }


    public List<QA> GenerateQuestions(SemDocument semDoc)
    {
        var result = new List<QA>();
        var targetNounLinks = CollectAllTargetNouns(semDoc);

        if (targetNounLinks.Count > 0)
        {
            for (int i = 0; i < targetNounLinks.Count; i++)
            {
                result.Add(new QA()
                {
                    Question = CreateQuestionSentence(targetNounLinks[i]),
                    Answer =
targetNounLinks[i].Target.Tokens[0].GetSourceText().ToLower()
                });
            }
        }

        return result;
    }
  }
}
```

ПРИЛОЖЕНИЕ В

Листинг клиентской программы

MainWindow.cs

```
using System;
using System.Windows;
using System.Collections.Generic;
using System.Linq;
using PoliticsTester.Client.Models;
using PoliticsTester.Client.Integrations;
using SautinSoft.Document;
using System.IO;
using Microsoft.Win32;
using Refit;

namespace PoliticsTester.Client
{
    public partial class MainWindow
    {
        private readonly IPoliticsTesterApi iPoliticsTesterApi;
        private UserAnswers UserAnswers { get; set; }
        private TestResult TestResult { get; set; }
        private List<string> questions;
        private int SummPoints { get; set; }
        private int CurQuestionID { get; set; }
        private TextModel textFromFile = new TextModel();
        private string filePath;

        public MainWindow()
        {
            UserAnswers = new UserAnswers();
            UserAnswers.Answers = new List<string>();

            InitializeComponent();

            iPoliticsTesterApi =
RestService.For<IPoliticsTesterApi>("https://localhost:5001");

            questions = new List<string>();
            SummPoints = 0;
            RefreshUI(0);
        }

        private void butt_OpenFile_Click(object sender, RoutedEventArgs
e)
        {
            OpenFileDialog openFileDialog = new OpenFileDialog();
            openFileDialog.Multiselect = false;
            openFileDialog.Filter = "All files|*.*|Word|*.docx|Word 97-
2003|*.doc|PDF files|*.pdf|Text files|*.txt";
            Nullable<bool> dialogOk = openFileDialog.ShowDialog();

            if (dialogOk == true)
            {
                filePath = openFileDialog.FileName;
            }
        }
        private void butt_Connect_Click(object sender, RoutedEventArgs e)
        {
            textFromFile.Text = GetTextFromFile();

            if (textFromFile.Text != "")
            {
                UserAnswers.Answers.Clear();
                questions.Clear();
                questions = GetQuestions(textFromFile).ToList();

                if (questions.Count > 0)
                {
                    RefreshUI(1);
                }
            }
        }

        private void butt_Next_Click(object sender, RoutedEventArgs e)
        {
            if (progressBar.Value < progressBar.Maximum)
            {
                UserAnswers.Answers.Add(textBox_Answer.Text);
                CurQuestionID++;
                RefreshUI(2);
            }

            if (Math.Abs(progressBar.Value - progressBar.Maximum) == 0)
            {
                FinishTest();
            }
        }

        private void butt_Finish_Click(object sender, RoutedEventArgs e)
        {
            FinishTest();
        }

        private void butt_Restart_Click(object sender, RoutedEventArgs e)
        {
            RefreshUI(1);
        }

        private string GetTextFromFile()
        {
            if (File.Exists(filePath))
            {
                DocumentCore dc = DocumentCore.Load(filePath);
                return dc.Content.ToString();
            }
            else
                return "";
        }

        private void RefreshUI(int idCase)
        {
            switch (idCase)
            {
                case 0: //Когда вопросов ещё нет
                    butt_Next.IsEnabled = false;
                    butt_Finish.IsEnabled = false;
                    butt_Restart.IsEnabled = false;
                    progressBar.Value = 0;
                    textBox_Question.Text = "";
                    break;
                case 1: //Когда вопросы появились или тест запущен заново
                    butt_Next.IsEnabled = true;
                    butt_Finish.IsEnabled = true;
                    butt_Restart.IsEnabled = true;
                    CurQuestionID = 0;
                    progressBar.Value = 0;
                    textBox_Question.Text = questions[0];
```

```
                progressBar.Maximum = questions.Count;
                break;
            case 2: //Когда вопросы листают
                progressBar.Value = CurQuestionID;
                if (CurQuestionID <= questions.Count - 1)
                {
                    textBox_Question.Text = questions[CurQuestionID];
                }
                break;
            case 3: //Когда тест завершен
                butt_Next.IsEnabled = false;
                butt_Finish.IsEnabled = false;
                butt_Restart.IsEnabled = true;
                textBox_Question.Text = "";
                break;
        }

        textBox_Answer.Text = "";
        textBox_Results.Text = "";
        label_Scores.Content = "Баллов: " + SummPoints;
        label_QuestionNumber.Content = progressBar.Value + "/" +
questions.Count;
    }

    private void FinishTest()
    {
        if (textBox_Answer.Text != "")
        {
            UserAnswers.Answers.Add(textBox_Answer.Text);
            CurQuestionID++;
            RefreshUI(2);
        }

        if (UserAnswers.Answers.Count > 0)
        {
            TestResult = GetTestResult();
            UserAnswers.Answers.Clear();
            SummPoints += TestResult.Points;
            RefreshUI(3);
            textBox_Results.Text =
GetStartTextOfResult(TestResult.Points) + "\n\n";

            foreach (QuestionInformation questionInformation in
TestResult.QuestionResults)
            {
                textBox_Results.Text += questionInformation.Question +
"\n";
                textBox_Results.Text += "Правильный ответ: " +
questionInformation.CorrectAnswer + "\n";
                textBox_Results.Text += "Ваш ответ: " +
questionInformation.UserAnswer + "\n\n";
            }
        }
        else
        {
            RefreshUI(3);
            textBox_Results.Text = GetStartTextOfResult(0) + "\n\n";
        }
    }

    private string GetStartTextOfResult(int curPoints)
    {
        string startText = "";
        if (curPoints > 0)
            startText += "Поздравляем, ";
        else
            startText += "К сожалению, ";
            startText += "вы набрали " + curPoints;

            if (curPoints == 1)
                startText += " балл!";
            if (curPoints > 1 && curPoints < 5)
                startText += " балла!";
            else if (curPoints >= 5 || curPoints == 0)
                startText += " баллов!";

            return startText;
        }

        public TestResult GetTestResult()
        {
            TestResult response = null;
            try
            {
                response =
iPoliticsTesterApi.GetTestResult(UserAnswers).Result;
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            return response;
        }
        public IEnumerable<string> GetQuestions(TextModel textModel)
        {
            IEnumerable<string> response = null;
            try
            {
                response = iPoliticsTesterApi.GetQuestions(textModel).Result;
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            return response;
        }
    }
}
```

IPoliticsTesterApi.cs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using PoliticsTester.Client.Models;
using Refit;

namespace PoliticsTester.Client.Integrations
{
    public interface IPoliticsTesterApi
    {
        [Post("/politics/questions")]
        public Task<IEnumerable<string>>
GetQuestions(TextModel textModel);

        [Get("/politics/testresult")]
        public Task<TestResult>
GetTestResult(UserAnswers userAnswers);
    }
}
```

QuestionInformation.cs

```
namespace PoliticsTester.Client.Models
{
   public class QuestionInformation
   {
      public string UserAnswer { get; set; }

      public string Question { get; set; }

      public string CorrectAnswer { get; set; }
   }
}
```

UserAnswers.cs

```
using System.Collections.Generic;

namespace PoliticsTester.Client.Models
{
   public class UserAnswers
   {
      public List<string> Answers { get; set; }
   }
}
```

TestResult.cs

```
namespace PoliticsTester.Client.Models
{
   public class TestResult
   {
      public int Points { get; set; }

      public QuestionInformation[] QuestionResults { get; set; }
   }
}
```

TextModel.cs

```
namespace PoliticsTester.Client.Models
{
   public class TextModel
   {
      public string Text { get; set; }
   }
}
```

MainWindow.xaml

```
<Window x:Class="PoliticsTester.Client.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
      xmlns:local="clr-namespace:PoliticsTester.Client"
      mc:Ignorable="d"
      Title="PoliticsTester.Client" Height="450" Width="800">
   <Grid>
      <Grid.RowDefinitions>
         <RowDefinition Height="40"></RowDefinition>
         <RowDefinition Height="0.3*"></RowDefinition>
         <RowDefinition Height="0.2*"></RowDefinition>
         <RowDefinition Height="0.2*"></RowDefinition>
         <RowDefinition Height="*"></RowDefinition>
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
         <ColumnDefinition Width="100"></ColumnDefinition>
         <ColumnDefinition></ColumnDefinition>
         <ColumnDefinition Width="100"></ColumnDefinition>
      </Grid.ColumnDefinitions>

      <TextBox x:Name="textBox_Question" Grid.Column="1"
HorizontalAlignment="Stretch" Margin="0,5,0,0" Grid.Row="2"
TextWrapping="Wrap" VerticalAlignment="Stretch"
IsReadOnly="True"/>
      <TextBox x:Name="textBox_Answer" Grid.Column="1"
HorizontalAlignment="Stretch" Margin="0,5,0,0" Grid.Row="3"
TextWrapping="Wrap" VerticalAlignment="Stretch"/>
      <TextBox x:Name="textBox_Results" Grid.Column="1"
HorizontalAlignment="Stretch" Margin="0,5,0,15" Grid.Row="4"
TextWrapping="Wrap" VerticalAlignment="Stretch"
IsReadOnly="True" VerticalScrollBarVisibility="Auto"/>

      <Label Grid.Column="0" Grid.Row="2" Content="Вопрос:"
HorizontalAlignment="Right" Margin="0,0,5,0"
VerticalAlignment="Center" Width="60"/>
      <Label Grid.Column="0" Grid.Row="3" Content="Ответ:"
HorizontalAlignment="Right" Margin="0,0,5,0"
VerticalAlignment="Center" Width="60"/>
      <Label Grid.Column="0" Grid.Row="4" Content="Результаты:"
HorizontalAlignment="Right" Margin="0,10,5,0"
VerticalAlignment="Top"/>
      <ProgressBar x:Name="progressBar" Grid.Column="1"
Height="16" Margin="90,0,65,0" VerticalAlignment="Center"/>
      <Label x:Name="label_QuestionNumber" Content="0/N"
Grid.Column="1" HorizontalAlignment="Right"
VerticalAlignment="Center" Margin="465,0,10,0"/>
      <Label x:Name="label_Progressbarr" Content="Пройденно:"
Grid.Column="1" HorizontalAlignment="Left"
VerticalAlignment="Center" Margin="0,0,0,0"/>
      <Label x:Name="label_Scores" Content="Баллы: 0"
HorizontalAlignment="Left" Margin="6,0,0,0"
VerticalAlignment="Center"/>

      <Button x:Name="butt_Next" Grid.Column="2" Grid.Row="2"
Content="-&amp;gt;" Margin="0,5,0,0" Width="90" Height="30"
Click="butt_Next_Click"/>
      <!--<Button x:Name="butt_Back" Grid.Column="2" Grid.Row="2"
Content="&amp;lt;-" Margin="-40,0,0,0" Width="34" Height="23"
Click="butt_Back_Click" Visibility="Hidden"/>-->
      <Button x:Name="butt_Finish" Content="Завершить"
Grid.Row="4" Grid.Column="2" HorizontalAlignment="Center"
VerticalAlignment="Top" Margin="0,130,0,0" Height="30" Width="90"
Click="butt_Finish_Click"/>
      <Button x:Name="butt_OpenFile" Content="Открыть файл"
Grid.Column="2" HorizontalAlignment="Center" Margin="0,5,0,0"
Grid.Row="3" VerticalAlignment="Center"
Click="butt_OpenFile_Click" Width="90" Height="30"/>
      <Button x:Name="butt_Restart" Content="Заново"
Grid.Column="2" HorizontalAlignment="Center" Margin="0,10,0,0"
Grid.Row="4" VerticalAlignment="Top" Click="butt_Restart_Click"
Width="90" Height="30"/>
      <Button x:Name="butt_Connect" Grid.Column="2"
HorizontalAlignment="Center" Margin="0,5,0,0" Grid.Row="1"
VerticalAlignment="Center" Click="butt_Connect_Click" Width="90">
         <Button.Content>
            <TextBlock Text="Получить вопросы"
TextWrapping="Wrap" TextAlignment="Center"/>
         </Button.Content>
      </Button>
   </Grid>
</Window>
```