# ВВЕДЕНИЕ

В современном мире, где информационные технологии занимают все более важное место, проблема плагиата и заимствования чужих текстов становится все более актуальной. Т.к. информация легко доступна и распространяется с высокой скоростью, вопросы оригинальности и авторского выходят на первый план. Плагиат не только нарушает этические принципы и стандарты академической честности, но и подрывает основы научного и интеллектуального развития.

Системы проверки уникальности текстов играют важную роль в борьбе с этим явлением. Они позволяют авторам, студентам, ученым и другим пользователям убедиться в оригинальности своих текстов, а также выявить случаи копирования и несанкционированного использования чужих материалов.

Такие системы помогают поддерживать академическую честность, защищать интеллектуальную собственность, обеспечивать качество научных и учебных работ. Они также способствуют повышению уровня образования, науки и культуры, поскольку стимулируют творческий подход к написанию текстов, исключая возможность простого копирования и плагиата. Данные системы являются неотъемлемой частью современного образовательного и научного процесса.

Активную роль в разработке таких систем играют нейронные сети.  Благодаря своей способности анализировать большие объемы информации и распознавать сложные паттерны, нейросети могут эффективно выявлять сходства между текстами, даже если они были изменены или переформулированы.

Целью преддипломной практики является разработка веб-приложения «Plagiarism Detection», определяющего источники заимствований и сравнения двух документов.

Из всего вышеперечисленного можно сказать, что понимание проблемы плагиата и заимствования поможет ученым, студентам, писателям и другим творческим личностям в создании оригинальных и значимых работ, способствуя развитию интеллектуальной среды, науки и культуры в целом.

# 1 ОБЩИЕ СВЕДЕНИЯ ОБ ОРГАНИЗАЦИИ ГУО «ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИИ П.О. СУХОГО»

**1.1 История организации**

Гомельский государственный технический университет имени Петра Осиповича Сухого (ГГТУ им. П.О. Сухого) – это одно из ведущих технических учебных заведений в Республике Беларусь, имеющее богатую историю и заслуженное имя своего основателя. История университета берет свое начало в далеком 1967 году, когда было принято решение о создании высшего технического учебного заведения в городе Гомель. Университет был призван стать катализатором интеллектуального и технического прогресса в регионе

Первоначально университет назывался Гомельский инженерно-строительный институт, который был ориентирован на подготовку специалистов в области строительства, архитектуры, инженерии и других технических специальностей. С течением времени институт развивался и расширял свою деятельность, стремясь к повышению качества образования и научных исследований. Внешний вид здания университета представлен на рисунке 1.1.



Рисунок 1.1 – Внешний вид главного корпуса университета

В 1993 году институт был переименован в Гомельский государственный технический университет имени П.О. Сухого в честь выдающегося белорусского ученого и инженера Петра Осиповича Сухого, чье имя стало символом качества и престижа учебного заведения. Под руководством нового ректора университет продолжил свое развитие, становясь центром научных исследований и технологических инноваций.

Сегодня ГГТУ имени П.О. Сухого является крупным образовательным центром, предлагающим широкий спектр образовательных программ в области технических наук, информационных технологий, строительства, экономики и других областей. Университет активно сотрудничает с предприятиями и организациями, проводит научные исследования, участвует в международных проектах, способствуя развитию инновационной деятельности и подготовке высококвалифицированных специалистов.

История ГГТУ им. П.О. Сухого свидетельствует о его значимой роли в образовании и науке, о стремлении к совершенствованию и инновациям, о важности его вклада в развитие технической и научной сферы Республики Беларусь.

**1.2 Структурные подразделения организации**

Гомельский государственный технический университет имени П.О. Сухого имеет разветвленную структуру, включающую в себя несколько ключевых подразделений, каждое из которых выполняет определенные функции и задачи.

Факультеты. Университет состоит из нескольких факультетов, таких как Факультет автоматизированных и информационных систем, Энергетический факультет, Машиностроительный факультет, Механико-технологический факультет и Гуманитарно-экономический факультет. Каждый факультет специализируется на определенных областях знаний и предлагает свои программы обучения [1].

Исследовательские центры. Университет активно развивает научно-исследовательскую деятельность и имеет специализированные исследовательские центры, где ведется работа над различными проектами в области технических наук, информационных технологий, энергетики и других направлений.

Лаборатории и учебные мастерские. Университет обладает современными лабораториями и учебными мастерскими, где студенты могут практически применять полученные знания, проводить эксперименты и разрабатывать проекты под руководством опытных специалистов.

Административные подразделения. В университете также функционируют административные подразделения, ответственные за организацию учебного процесса, финансовое планирование, кадровые вопросы, международное сотрудничество и другие аспекты управления университетом.

Центры поддержки студентов. Университет предоставляет разнообразные услуги и программы поддержки студентов, включая карьерное консультирование, психологическую помощь, спортивные мероприятия, культурные мероприятия и другие инициативы, способствующие разностороннему развитию студентов.

Данные подразделения университета обеспечивают комплексную поддержку образовательного процесса, научно-исследовательской работы и развития студенческой жизни в университете.

**1.3 Престижность получаемой специальности**

Факультет автоматизированных и информационных систем (ФАИС) является одним из ключевых подразделений ГГТУ им. П.О. Сухого, где студенты получают современные знания в области автоматизации, информационных технологий, программирования и других важных аспектов цифровой трансформации. Преподаватели и исследователи на факультете ФАИС активно внедряют инновационные методики обучения, обеспечивая студентам актуальные навыки и знания, необходимые для успешной карьеры в сфере информационных технологий.

Получаемые на факультете ФАИС специальности отличаются высокой престижностью на рынке труда. Студенты, завершившие обучение по направлениям автоматизированных и информационных технологий, пользуются высоким спросом со стороны работодателей, как на территории Республики Беларусь. Это свидетельствует о качественной подготовке специалистов на факультете ФАИС и о высокой оценке их компетенций в сфере IT.

Гомельский государственный технический университет имени П.О. Сухого обеспечивает своим выпускникам гарантированные перспективы трудоустройства. Благодаря престижности университета и качеству обучения, выпускники ГГТУ им. П.О. Сухого имеют возможность выбирать из многочисленных предложений о работе как на родине, так и за рубежом. Университет поддерживает тесные связи с ведущими компаниями и предприятиями, что обеспечивает выпускникам оптимальные условия для старта своей карьеры и профессионального роста.

**1.4 Должностные обязанности инженера-программиста в организации**

Инженер-программист в организации «ГГТУ им. П.О. Сухого» играет важную роль в разработке и поддержке программного обеспечения, необходимого для эффективного функционирования университета. Должностные обязанности этого специалиста включают в себя широкий спектр задач и ответственностей, направленных на обеспечение работы информационных систем и технологий университета.

Один из ключевых аспектов работы инженера-программиста в «ГГТУ им. П.О. Сухого» – это разработка и сопровождение специализированного программного обеспечения, необходимого для управления учебным процессом, научно-исследовательской деятельностью, административными процессами и другими аспектами работы университета. Инженер-программист отвечает за создание программных продуктов, учитывая специфические потребности и требования университетской среды.

Кроме того, инженер-программист в организации занимается анализом потребностей пользователей, определением функциональных требований к программному обеспечению и разработкой соответствующих технических спецификаций. Он также участвует в тестировании и отладке программ, обеспечивая их надежную работу и соответствие заявленным характеристикам.

Важной частью работы программиста является поддержка и обновление существующих программных продуктов, адаптация их к изменяющимся потребностям и технологическим требованиям. Он также отвечает за обеспечение безопасности информационных систем университета, включая защиту данных, обеспечение конфиденциальности и целостности информации.

Наряду с этой, инженер-программист взаимодействует с другими специалистами и отделами университета, участвует в проектных группах, проводит консультации и обучение пользователей, обеспечивая эффективное внедрение и использование программного обеспечения.

Таким образом, инженер-программист играет ключевую роль в поддержании информационной инфраструктуры университета, обеспечивая эффективное функционирование учебного процесса и научно-исследовательской работы.

**2 ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ И ПОДХОДОВ К ПРОЕКТИРУЕМОЙ СИСТЕМЕ**

## 

**2.1 Основные этапы разработки системы проверки текстов**

Для разработки программного обеспечения необходимо определить используемую методологию проектирования. Основные задачи методологии проектирования – построение логической схемы последовательности действий по организации процесса проектирования, разработку методов и алгоритмов выполнения проектных процедур и операций. Это позволит грамотно организовать процесс разработки программного обеспечения. Для приложения применена методология проектирования *Waterfall*. Схема методологии представлена на рисунке 2.1.

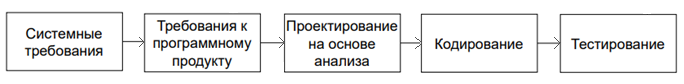


Рисунок 2.1 – Схема каскадной методологии проектирования

*Waterfall*, или каскадная модель – методология, которая подразумевает последовательное прохождение стадий, каждая из которых должна завершиться полностью до начала следующей. При использовании такой модели легко управлять проектом. Модель дает хорошие результаты для относительно небольших проектов с четко определенными требованиями.

Исходя из схемы, процесс реализации приложения разделен на следующие этапы:

1. определение примерных характеристик, которым должен соответствовать компьютер для того, чтобы на нём могло использоваться разрабатываемое программное обеспечение;
2. описание требований к программному продукту, которым должно отвечать создаваемое программное обеспечение (основные функции приложения, входные данные, форма представления выходных данных и т.п.) с учетом системных требований;
3. проектирование на основе анализа сформированных системных и функциональных требований приложения, в процессе которого определяются:
   1. внутренние свойства программы;
   2. схема данных;
   3. взаимодействие компонентов приложения;
   4. интерфейс приложения;
4. реализация спроектированных программных модулей и их взаимодействия;
5. проверка работоспособности программы для определения, соответствует ли разработанное программное обеспечение поставленным требованиям к реализации.

Важным этапом является поддержка и обновление приложения после его запуска. Регулярное обновление приложения помогает улучшать его функциональность, исправлять ошибки и следить за его производительностью. Все эти этапы в совокупности обеспечивают успешную разработку и поддержку веб-приложения, отвечающего потребностям пользователей и бизнеса.

**2.2 Требования, предъявляемые к проектируемой системе**

Для определения основных функций программы применен метод декомпозиции, который подразумевает разделение задач проекта на более мелкие подзадачи. Это позволяет точно составить содержание каждой подзадачи проекта и сформировать проект в виде иерархической структуры. После применения метода декомпозиции приложение было разделено на блоки (рисунок 2.2).

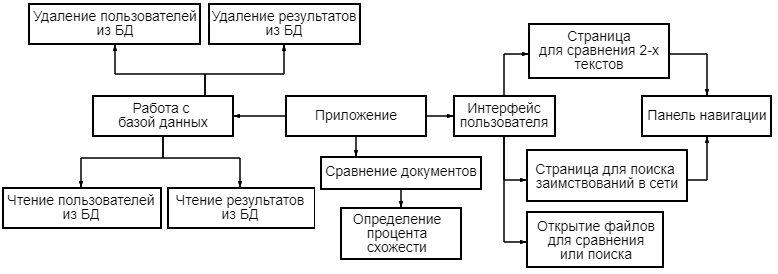


Рисунок 2.2 – Функциональное назначение частей приложения

Из построенной схемы следует, что разработке подлежат следующие системы:

* система проверки уникальности текстов, реализующая проверку двух заданных пользователем документов либо сравнение заданного документа с имеющимися источниками из сети;
* база данных, которая должна содержать зарегистрированных пользователей и результаты сравнений документов;
* интерфейс пользователя, который будет удобен для пользователя и будет предоставлять возможность работы с основными функциями программы посредством его взаимодействия с элементами меню.

Система проверки уникальности текстов должна использовать различные алгоритмы сравнения текстов, позволяющих определять лексическое и семантическое сходство между текстами, в том числе с помощью нейронных сетей.

База данных должна быть создана средствами свободной реляционной системы управления базами данных *PostgreSQL* с использованием языка запросов *SQL* для добавления результатов и чтения их из базы для дальнейшего использования приложением.

Интерфейс пользователя должен состоять из следующих элементов:

* главное меню, содержащее пункты меню для выбора варианта сравнения текстов;
* страница для сравнения заданных пользователем текстов между собой;
* страница для поиска заимствований источников из сети;
* панель навигации между меню приложения.

Тексты для сравнения могут быть выбраны следующими способом: использовать элемент меню и в появившемся диалоговом окне выбрать файл, который подлежит сравнению.

Исходя из описанных функций приложения, можно составить схему функциональной структуры программной системы (рисунок 2.3).



Рисунок 2.3 – Схема функциональной структуры программы

*Python* является одним из наиболее популярных языков программирования для разработки сегодня. Его преимущества включают в себя простоту и интуитивность синтаксиса, обширную поддержку библиотек для обработки текстов и машинного обучения, а также широкое сообщество разработчиков, что обеспечивает доступ к множеству ресурсов и инструментов для работы с текстовыми данными. *Python* также хорошо подходит для алгоритмов в области обработки естественного языка и нейронных сетей.

Архитектуры трансформеров, такие как *SBERT*, *GPT* и другие, представляют собой мощные инструменты для работы с текстовыми данными и обработки естественного языка. Они обладают способностью эффективно анализировать и генерировать тексты. Применение данной архитектуры позволяет улучшить точность и эффективность алгоритмов, обеспечивая более надежное определение уникальности. Для задачи семантического сходства подходит фреймворк *sentence-transformers* с набором предобученных моделей, в том числе и для русского языка.

Из существующих средств отправки запросов поисковой системе был выбран *Google Search API*, так как он имеет подробную документацию, бесплатен, не требует привязки карты и использует алгоритмы выдачи самой популярной поисковой системы *Google*.

Приложение должно работать на операционной системе *Windows*. В качестве фреймворка для создания серверной веб-приложения используется *Flask*. Он является одним из популярных инструментов, а также гибким и легким в использовании.

**2.3 Существующие аналоги систем проверки уникальности текстов**

Сейчас в сети Интернет имеются различные ресурсы, которые предоставляют возможность проверки уникальности текстов. Каждый из них обладает преимуществами, недостатками и различиями в виде набора опций, пользовательского интерфейса и алгоритмов сравнения. Ниже приведен перечень таких ресурсов и их описание.

Антиплагиат.ру – российский интернет-проект, программно-аппаратный комплекс для проверки текстовых документов на наличие заимствований из открытых источников в сети Интернет и других источников. Проект доступен как для рядовых пользователей, так и для высших учебных заведений [2].

Система предлагает набор услуг, в совокупности реализующих технологию определения заимствований из общедоступных сетевых источников, специализированных коллекций документов, электронных библиотек и т.п. Сервис представляет собой специализированную поисковую систему.

Система собирает информацию из различных источников: загружает из сети Интернет и обрабатывает сайты, находящиеся в открытом доступе, базы научных статей и рефератов. Загруженные документы проходят процедуру фильтрации, в результате которой отбрасывается бесполезная (с точки зрения потенциального цитирования) информация. На следующем этапе каждый из полученных таким образом текстов определённым образом форматируется и заносится в системную базу данных. Кроме того, в общую базу текстов поступают документы, загруженные на проверку пользователем, если такая возможность была разрешена им во время процедуры загрузки.

Все пользовательские документы, загружаемые для проверки, ставятся в очередь на обработку. Проверка документа, такого как, например, реферат среднего размера, занимает несколько десятков секунд. После проверки документа пользователь получает доступ к отчёту, в котором представляются результаты. Пример отчета отображен на рисунке 2.4.

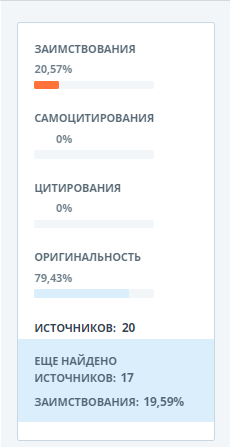


Рисунок 2.4 – Отчет после проверки документа

Структура полного анализа позволяет выделять в проверяемом тексте заимствованные части как по всем источникам, так и по их любому подмножеству.

*ETXT* является одной из известнейших программ сравнения текстов. *EТХТ* не редко используется для проверки оригинальности студенческих проектов, так как позволяет проверять бесплатно большие по объему тексты с возможностью просмотра полного отчета. Однако ее алгоритмы направлены на определение уникальности контента для сайтов, но никак не учебных и, тем более, научных трудов. Интерфейс *ETXT* представлен на рисунке 2.5.

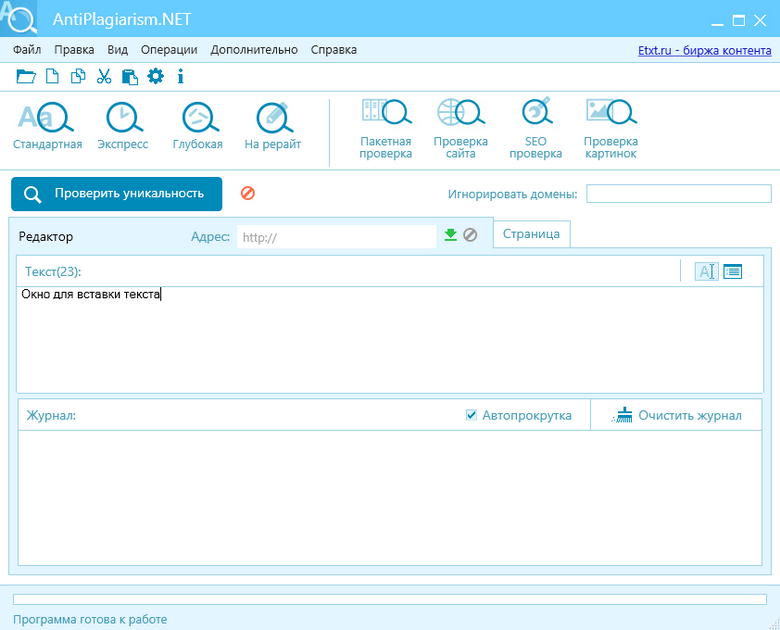


Рисунок 2.5 – Интерфейс *ETXT*

Поиск заимствований *ETXT* осуществляет путем сравнения проверяемого контента с находящимися в свободном доступе интернет-источниками, используя различные поисковые системы. Также программа способна выявить рерайтинг – обработанный текст исходного документа. Разнообразие алгоритмов проверок на рерайтинг часто показывает результаты, существенно отличающиеся от результатов других программ проверки на оригинальность.

Онлайн-версия программы обеспечивает довольно быструю проверку, независимо от внешних факторов с функцией сохранения результатов проверки на сервере. Однако бесплатно проверить на плагиат онлайн можно лишь текст, объемом до 3000 символов без регистрации и до 5000 символов после прохождения процедуры регистрации.

Система *Advego* не предназначена для проверки дипломных, курсовых и прочих студенческих проектов, но имеет ряд преимуществ, позволяя проверять онлайн на плагиат большие тексты до 95000 знаков на 8 языках, включая русский. Помимо этого, *Advego* позволяет обнаружить рерайтинг и распознать контент, обработанный синонимайзерами, обнаружить наличие сторонних символов или программного кода, направленных на искусственное повышение уникальности текста. Зарегистрированным пользователям доступна опция хранения результатов проверок. В целом, система проверки контента на плагиат *Advego* оценивает как техническую, так и смысловую оригинальность, а также наличие речевых штампов и вводных выражений.

*TextDiff* – бесплатная программа для сравнения файлов. Кроме сравнения содержимого файлов, программа *TextDiff* имеет возможность сравнить два каталога, выделяя одинаковые и различающиеся папки и файлы сравниваемых директорий. Внешний вид программы представлен на рисунке 2.6.

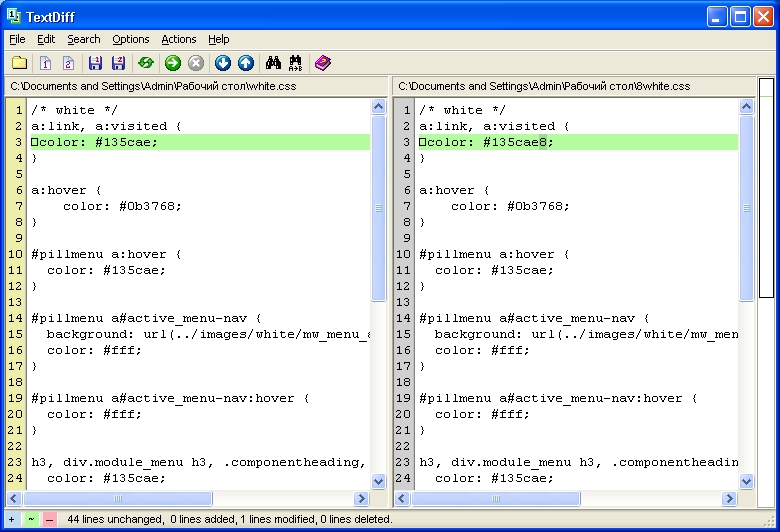


Рисунок 2.6 – Интерфейс *TextDiff*

Главное окно программы *TextDiff* разделено по вертикали на два смежных окна. В них отображается содержимое сравниваемых файлов. Скроллинг строк в области просмотра одного файла одновременно ведёт к автоматическому скроллингу строчек другого файла, и пользователь сможет оценить, в какой строке между файлами есть различия. Различающиеся, удалённые и добавленные строки сравниваемых файлов утилита выделяет другим цветом. *TextDiff* не требует установки и полностью поддерживает *Unicode* [3].

Все рассмотренные программы могут работать как с подключением к сети Интернет, так и без него. В результате проведенного анализа существующих решений предлагается разработать систему искусственного интеллекта для проверки уникальности текстов. При проверке необходимо выводить результат сравнения. Целью разработки является уменьшение количества заимствований чужих работ студентами и (или) иными научными сотрудниками. Преимуществами приложения будут являться удобный интерфейс и возможность использования нейронных сетей современной архитектуры трансформеров, обеспечивая более надежное определение плагиата и заимствований, выявляя сложные паттерны и зависимости в текстовых данных.

**3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ «PLAGIARISM DETECTION»**

**3.1 Структура и архитектура веб-приложения «Plagiarism Detection»**

Для определения структуры приложения использована модель «черного ящика», которая подразумевает собой некую систему, в котором полностью отсутствуют сведения о содержании этого фрагмента (структура и внутренние процессы не известны), а задаются только входные и выходные связи системы со средой. В такой модели границы между системой и средой не описываются, а только подразумеваются, признаются существующими. Такая модель хороша ввиду своей внешней простоты.

Графическая модель «черного ящика» отображает только связи системы со средой, в виде перечня «входов» и «выходов». Вид модели представлен на рисунке 3.1.

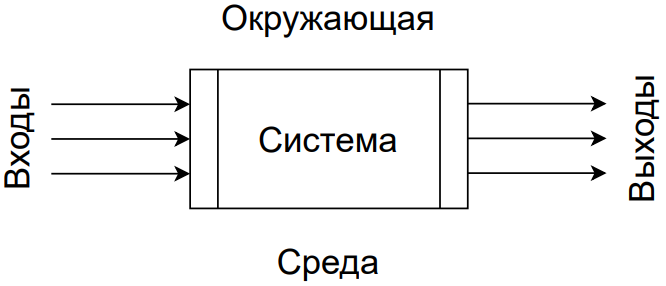


Рисунок 3.1 – Графическое представление модели «черного ящика»

Построение модели «черного ящика» не является тривиальной задачей, так как ответ на вопрос, сколько входов-выходов следует включать в модель и какие именно, не всегда однозначен. От правильного построения модели зависит корректное выполнение ее функций и достижение поставленных целей. Перед построением модели необходимо правильно обозначить цель системы – состояние среды, удовлетворяющая заданным требованиям и реализующая поставленную задачу.

В данном случае целью системы является сравнение текстов или определение заимствований из сети. Исходя из этого, необходимо определить входы и выходы системы:

1. входы:
   1. данные о документах;
   2. тексты для сравнения;
2. выходы:
   1. таблица в базе с данными о результатах документов;
   2. результат сравнения в процентном соотношении;
   3. таблица результатов с найденными источниками из сети.

Исходя из описанного выше, целесообразно провести разбиение системы на более мелкие части по следующему принципу:

* построенная система разбивается на набор подсистем;
* каждая подсистема разбивается на отдельные неделимые части (элементы).

В целом, такая модель помогает абстрагироваться от внутренней реализации системы и сосредоточиться на её взаимодействии с внешней средой. Это позволяет лучше понять, как система взаимодействует с внешними элементами и какие входные данные она принимает, а также какие выходные данные генерирует. В результате разбиения получена модель, отражающая состав системы и описание каждой находящейся в ней подсистемы (рисунок 3.2).

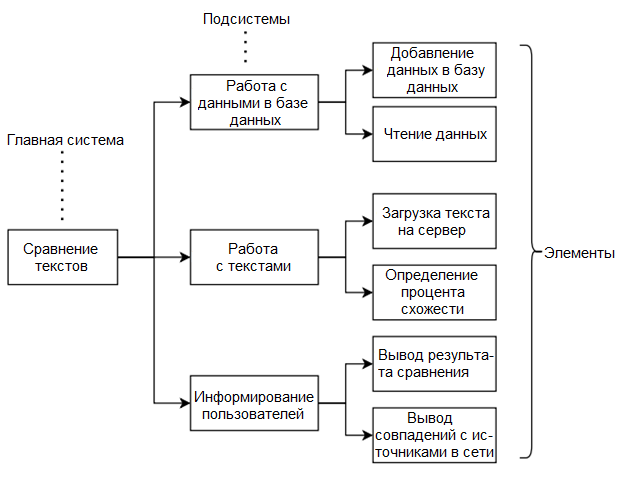


Рисунок 3.2 – Модель системы проверки уникальности текстов

При разработке архитектуры использован паттерн *Model-View-Controller* (далее – *MVC*), который подразумевает собой разделение данных приложения, пользовательского интерфейса и управляющей логики на отдельные компоненты. Это позволяет изменять каждый из компонентов независимо. Схема паттерна *MVC* изображена на рисунке 3.3.

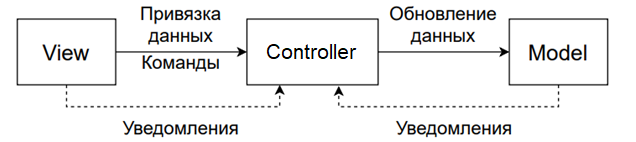


Рисунок 3.3 – Схема паттерна *MVC*

Согласно схеме, логика приложения разделена на следующие части:

* модель (*Model*);
* представление (*View*), графический интерфейс для работы с данными или их отображения;
* контроллер (*Controller*), связи между моделью и представлением.

Модель описывает используемые в приложении данные и содержит логику, непосредственно связанную с этими данными. При этом модель не должна содержать никакой логики, связанной с отображением данных и взаимодействием с визуальными элементами управления.

Модель содержит работу с базой данных и сущность. Данный уровень состоит из классов, представляющих сущности таблиц. Основным элементами являются классы, описывающие сущность пользователя *Customer* и результатов проверки документов *DocumentsResults*. В качестве свойств класса пользователя необходимо указать:

* первичный ключ числового типа, представляющий собой номер записи в базе данных;
* имя пользователя в формате строки;
* логин пользователя в формате строки;
* пароль пользователя в формате строки;
* администратор ли он в виде булевого значения.

Также необходимо обеспечить полноценный доступ к данным при помощи методов для извлечения и присвоения данных. Модель является лишь контейнером для хранения данных. Этот класс является основой, при помощи которой происходит сопоставление таблицы с классом.

Работа с базой данных является ресурсоемким процессом, поэтому методы должны быть максимально простыми. Стандартной классификации функций по манипуляции данными в *SQL* соответствуют операторы *Insert* (вставка), *Select* (чтение записей) и *Delete* (удаление записей). Исходя из этого, следует, что для каждого вида запроса необходимо реализовать свой метод. Схема данных отображена на рисунке 3.4.

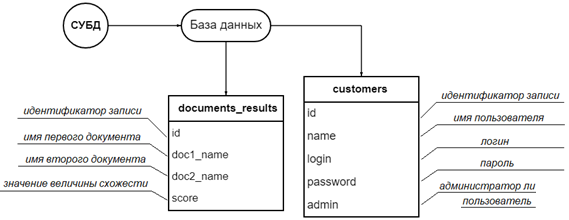


Рисунок 3.4 – Схема базы данных

Представление определяет визуальный интерфейс, при помощи которого пользователь взаимодействует с приложением. В случае работы с веб-приложением представление – это *html*-разметка, которая определяет интерфейс в виде кнопок, текстовых полей, таблиц и других визуальных элементов.

Представление реализовано в виде отдельного программного модуля в виде *html*-страниц. Для создания элементов меню и графики интерфейса будут применены различные файлы с разметкой, скрипты и файлы стилей соответственно. Данный программный модуль предполагает реализацию следующих компонентов:

* главная страница, с формой авторизации;
* страница, отображаемая после авторизации и содержащая основные элементы меню;
* страница, на которой происходит сравнение двух документов;
* страница для сравнения текста с имеющимися источниками в сети;
* страница администратора с возможностью добавления результата в базу данных и получения таблицы результатов сравнения документов.

Стоит отметить, что каждой роли (пользователь или администратор) соответствует отдельная *html*-страница и *js*-скрипт, при помощи которого происходит реализация всей логики по обработке поведения на этой странице.

Для страницы сравнения текстов между собой, что и с предыдущей: кнопки для выбора файла, список для метода сравнения и отправки на сервер. Главное различие состоит в том, что второе текстовое поле недоступно для редактирования пользователем и предназначено для вывода совпадений текста с имеющимися документами базы данных. Для каждого документа должна отдельно рассчитываться степень уникальности.

Пользователь в роли администраторе имеет возможность добавить результат сравнения документов в базу данных. При желании пользователя добавить данные в базу, проверяется выбраны ли были файлы и получен ли результат сравнения. Также перед проверкой текстов происходит анализ содержимого документов на предмет превышения максимально допустимого количества символов.

Навигация и взаимодействия элементов пользовательского интерфейса представлена на рисунке 3.5.

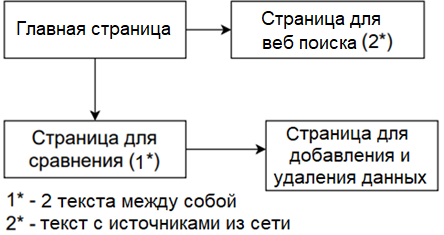


Рисунок 3.5 – Взаимодействие элементов интерфейса

Модель представления реализует свойства и команды, к которым привязано представление данных, и может уведомлять представление о любых изменениях состояния с помощью событий уведомления об изменениях. Свойства и команды, предоставляемые моделью представления, определяют функциональные возможности, предоставляемые пользовательским интерфейсом, но представление определяет, как эти функции должны отображаться.

Контроллеры представляют собой часть приложения, которая отвечает за обработку запросов от пользователей, взаимодействие с моделью данных и подготовку данных для отображения в представлении. В *Flask* контроллеры обычно представлены в виде функций или методов классов, которые связывают *url*-адреса с определенными действиями или операциями.

Контроллеры в *Flask* обрабатывают запросы от клиента, извлекают данные из модели (например, из базы данных), обрабатывают эти данные и передают их в представление для отображения пользователю. Они выполняют различные операции, такие как создание, чтение, обновление и удаление данных, а также реализовывать бизнес-логику приложения. Хорошо спроектированные контроллеры помогают разделить логику приложения на отдельные компоненты, что упрощает поддержку, тестирование и масштабирование приложения.

Для контроллеров необходимо определить следующие методы:

* метод, который осуществляет поиск заимствований из интернета и возвращает таблицу с найденными ссылками по соответствующим предложениям;
* метод, который будет производить сравнение заданных пользователем документов и алгоритма сравнения и предоставлять результат схожести;
* методы для добавления и удаления записей из базы данных о результатах сравнения двух документов.

**3.2 Интерфейс пользователя системы проверки текстов**

Верификация приложения происходит путем проверки работоспособности и взаимодействия классов, корректности выполнения запросов пользователей, отображений представлений на страницах.

На рисунке 3.6 приведена диаграмма вариантов использования, которая описывает функциональное назначение разработанного приложения. Диаграмма вариантов использования демонстрирует полный функционал разработанного приложения, с которым может взаимодействовать пользователь.

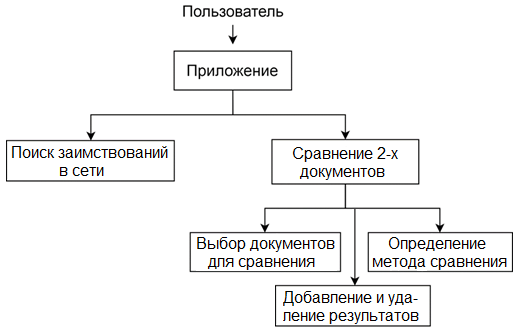


Рисунок 3.6 – Диаграмма вариантов использования приложения

Работа с созданной системой проверки тестов состоит из следующих этапов:

* запуск созданного приложения и базы данных *PostgreSQL*, с которым оно будет взаимодействовать;
* конфигурация параметров, используемых при сравнении (определение файлов документа и алгоритма проводимого сравнения);
* добавление результатов проверки документов в базу данных при условии, что пользователь обладает правами администратора.

На начальной странице пользователь должен пройти аутентификацию. Если пользователь не имеет аккаунта, имеется возможность пройти регистрацию по кнопке «*Don’t have an account? Sign up*».

При авторизации пользователя он попадает на главную страницу приложения, на которой ему предлагается выбор обработки текстов. Меню приложения представлено на рисунке 3.7.

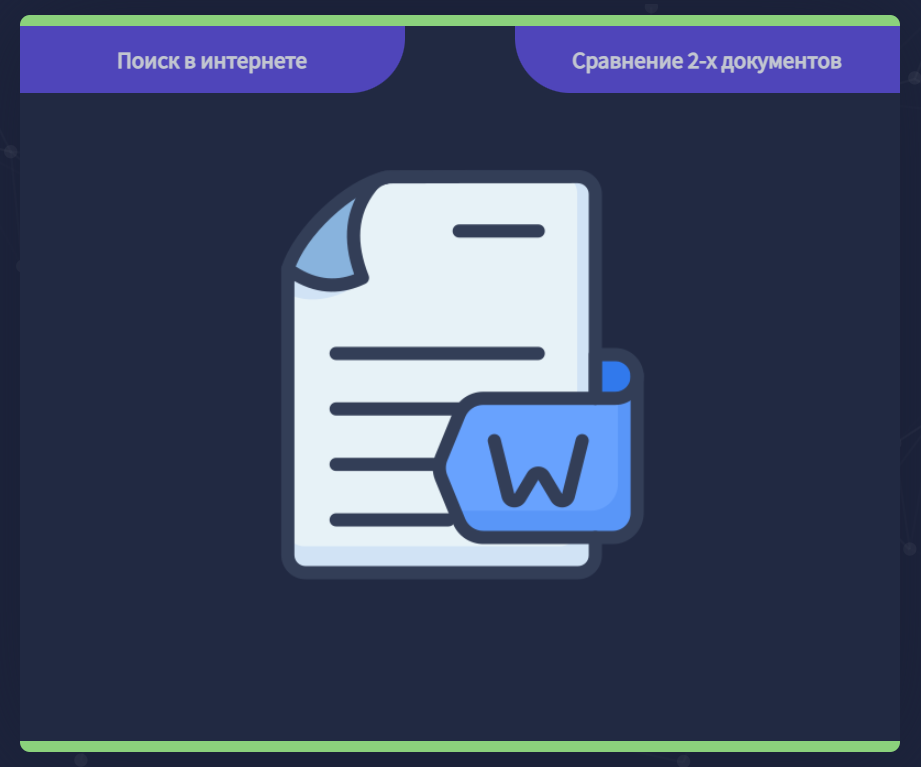


Рисунок 3.7 – Начальная страница пользователя

При нажатии кнопки «Поиск в интернете» пользователь имеет возможность загрузить файл документа для отправки его на сервер и дальнейшей проверки его содержимого на предмет заимствований из сети. Процесс проверки происходит по предложениям текста. В процессе поиска источников в интернете, отображается значок загрузки в виде вращающейся по кругу полоски. По окончанию проверки выводится таблица с найденными источниками, номером предложения и показателе схожести с найденным предложением в процентах. Результат поиска заимствований из сети представлен а рисунке 3.8.



Рисунок 3.8 – Таблица найденных источников в интернете

При выборе кнопки «Сравнение 2-х документов» пользователю предлагается загрузить оба документа и выбрать метод проверки (алгоритм шинглов, алгоритм *tf-idf*, сравнение текстов целиком или по частям с помощью предобученной нейронной сети). После отправки двух документов на проверку сервером, пользователю возвращается ответ в виде степени схожести двух текстов. Чем ближе число к единице, тем более похожи между собой документы. Результат проверки двух документов представлен на рисунке 3.9.

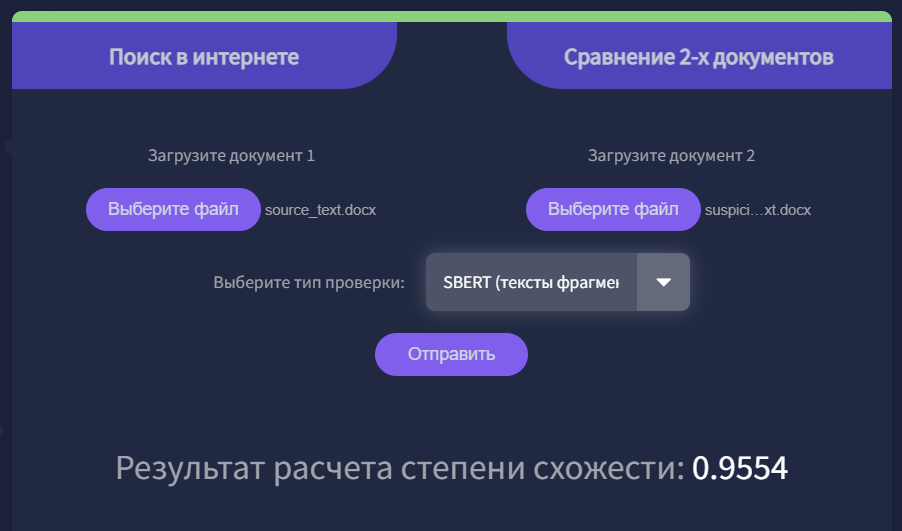


Рисунок 3.9 – Результат проверки двух документов

При обладании пользователем правами администратора, имеется возможность сохранять в базу данных результат проверки двух документов выбранным методом. После проверки появляется кнопка «Добавить», которая сохраняет запись в таблице *documents\_results*. Если данные были успешно добавлены, в браузер выводится сообщение об успешном выполнении операции.

Также администратор имеет возможность вывести все данные из таблицы результатов с помощью кнопки «Результаты». Данные представляют собой имена файлов первого и второго документов и величину их схожести между собой. Имеется также возможность удалить запись из таблицы, после чего происходит удаление строки как из таблицы html-разметки, так и записи базы данных.

Таблица результатов различных документов представлена на рисунке 3.10.



Рисунок 3.10 – Таблица результатов сравнения различных документов

**3.3 Используемые методы сравнения текстов и поиска плагиата в сети**

Внешние методы обнаружения заимствований сравнивают подозрительные документы с проверочным набором документов с целью нахождения потенциальных источников заимствований. В качестве проверочной базы могут служить как заранее подготовленная коллекция документов, так и ресурсы из Интернета. Глобальные методы анализа сходства используются для сравнения текстовых документов и применяются в задаче нахождения дубликатов документов.

В качестве алгоритмов сравнения текстов были применены алгоритм шинглов, *TF-IDF* и модели нейросетей *sentence-transformers*. Далее рассматривается каждый из них.

Алгоритм шинглов основан на разбиении текста на небольшие фрагменты (*shingles* – чешуйки) и сравнении их между собой для выявления сходств. Он позволяет находить нечеткие дубликаты, когда содержание одного документа почти идентично содержанию другого. Данный метод позволит найти нечеткие дубликаты, где разница документов составляет лишь небольшой участок текста (например, когда заменены только дата и имя автора документа, добавлены/удалены небольшие фрагменты текста), или когда тексты почти совпадают за исключением маленького количества слов в предложениях (например, замена всех вхождений слова его синонимом) [4, c. 47].

Недостатком алгоритма является его малая способность определения идентичности фраз. Многие сервисы проверки уникальности, основанные на алгоритме шинглов, покажут, что фразы уникальны, хотя для поисковых систем они идентичны. Зачастую проблема не в самом алгоритме, а в методах канонизации текста. При использовании морфологии в канонизации алгоритм легко распознает фразы как одинаковые, независимо от их окончаний. Однако он все еще не способен выявить более сложные конструкции и провести смысловую оценку текста [5].

*TF-IDF* (*Term Frequency*-*Inverse Document Frequency*) – это один из наиболее распространенных алгоритмов, используемых для определения схожести текстов. Этот алгоритм помогает оценить важность каждого слова в документе относительно других документов в коллекции. TF-IDF широко применяется в информационном поиске, анализе текстов, рекомендательных системах и других областях, где необходимо сравнивать и классифицировать текстовую информацию.

Основная идея алгоритма *TF-IDF* заключается в том, что важность слова в документе возрастает пропорционально частоте его употребления в документе (*TF*) и обратно пропорциональна частоте его употребления во всех остальных документах коллекции (*IDF*). Таким образом, слова, которые часто встречаются в конкретном документе, но редко в других документах, получают более высокий вес.

Для вычисления *TF-IDF* для каждого слова в документе необходимо выполнить следующие шаги:

1. Вычислить *TF* (частоту термина) – отношение числа вхождений слова к общему числу слов в документе;
2. Вычислить *IDF* (обратную частоту документа) – логарифм отношения общего числа документов к числу документов, содержащих данное слово;
3. Умножить *TF* на *IDF* для получения значения *TF-IDF* для каждого слова в документе.

После вычисления *TF-IDF* для всех слов в документах можно сравнивать их схожесть путем вычисления косинусного расстояния между векторами *TF-IDF* каждого документа. Чем ближе значения косинусного расстояния к 1, тем более схожи тексты [5].

В качестве инструмента вычисления схожести двух текстов используется класс *TfidfVectorizer* библиотеки *Scikit-learn*. Он выполняет токенизацию и стемминг поступающего набора текстовых данных и преобразует его в матрицу *TF-IDF* весов терминов.

Для сравнения текста также часто используются искусственные нейронные сети, представляющие собой математические модели, состоящие из взаимосвязанных и взаимодействующих простых процессоров (искусственных нейронов). Эти сети, объединенные в достаточно большую сеть с контролируемыми взаимодействиями, позволяют коллективно выполнять довольно сложные задачи отдельными простыми процессорами.

Нейронные сети архитектуры трансформеров используют механизмы самовнимания (*self-attention*), позволяющие модели взвешивать различные части входного текста при создании векторных представлений (эмбеддингов). Это помогает фиксировать долгосрочные зависимости и контекст.

Одним из современных и эффективных способов определения схожести текстов является использование библиотеки *sentence-transformers* от компании *Hugging Face*, которая предоставляет доступ к широкому спектру предобученных моделей для работы с текстом. *Sentence-transformers* – это библиотека *Python*, основанная на *PyTorch* и библиотеке *transformers* от *Hugging Face*. Она состоит из моделей глубокого обучения, способных преобразовывать предложения и текст в векторные представления, учитывающие семантическую близость и контекст. Вектора текста, созданные этими моделями, фиксируют семантическое содержание текста, что позволяет проводить точное сравнение и оценку сходства. Этот подход более точный и эффективный по сравнению с классическими методами, такими как *TF-IDF* или *Word2Vec*. Он широко используется в задачах сравнения текстов, поиска дубликатов, кластеризации текстов и других приложениях.

Для определения схожести текстов с помощью *sentence-transformers* необходимо выполнить следующие шаги:

1. Преобразовать каждое предложение (или текст) в векторное представление с помощью предварительно обученной модели нейронной сети;
2. Вычислить косинусное расстояние между векторами предложений. Чем ближе значение косинусного расстояния к 1, тем более схожи предложения семантически.

Если размер текстов небольшой, можно сравнивать их целиком. Но в случае с длинными текстами необходимо разбить его на фрагменты определенной длины. Это могут быть абзацы текста или куски по несколько сотен слов [6]. Различные модели предоставляют различную максимальную длину последовательности (текста) в токенах. Поэтому фрагмент текста должен быть не больше этой величины. Для русскоязычных текстов хорошо показала себя модель «*cointegrated/rubert-tiny2»*, чья максимальная длина последовательности составляет 2048 токенов. Т.к. фрагменты могут иметь разную длину, оставшиеся место заполняется специальными токенами заполнителями (*padding tokens*).

Для метрики схожести используется косинусное расстояние. После получения эмбеддингов фрагментов текста необходимо сравнить их между собой и определить среднее максимальных значений схожести [7, с 3145]. Далее можно установить некий порог сходства, ниже которого проверяемый документ можно считать оригинальным. Схема работы алгоритма представлена на рисунке 3.11.



Рисунок 3.11 – Схема сравнения текстов с помощью нейронной сети

В вопросе поиска заимствований из интернета можно использовать подход с поиском фрагментов текста, используя *Google Search API* для получения первых *k* результатов поиска, которые считаются самыми релевантными. В качестве запросов для апи используются предложения текста, полученные посредством токенизации с помощью библиотеки *NLTK*.

Далее потребуется библиотека *BeautifulSoup* для анализа *HTML*-кода и извлечения соответствующей информации. Получая определённый список результатов поиска, находятся те страницы, в которых есть присутствие данного предложения, процент заимствования вычисляется с помощью метода шинглов. Если заимствований не найдено, предложение не добавляется итоговую таблицу результатов. Общая схема поиска представлена на рисунке 3.12.

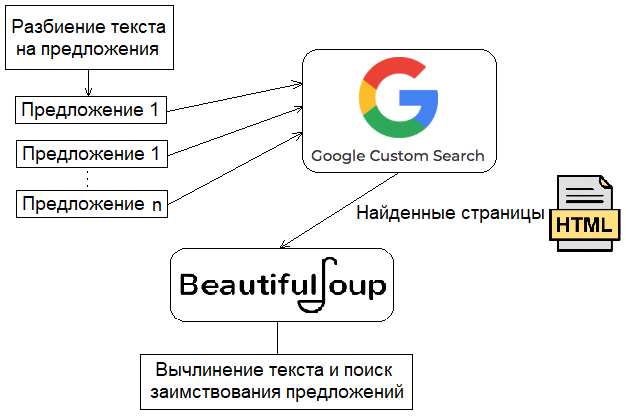


Рисунок 3.12 – Схема поиска заимствований текста из сети

Чем больше объем текста, тем длительнее осуществляется поиск источников. Для повышения скорости поиска применяется многопоточная обработка предложений.

Разработанное программное обеспечение реализует систему проверки уникальности текстов и имеет гибкую модульную структуру. Использование паттернов проектирования и объектно-ориентированного подхода позволяет достаточно просто расширять его возможности путём добавления нового функционала, не нарушая при этом работоспособность уже имеющегося. Приложение состоит из нескольких программных модулей, представляющих собой пользовательский интерфейс, алгоритмы сравнения текстов и поиска заимствований, а также логика работы с базой данных.

# ЗАКЛЮЧЕНИЕ

В процессе прохождения преддипломной практики было разработано веб-приложение для проверки уникальности текстовых документов. Данная система позволяет выявить плагиат студенческих работ, мотивируя их создавать более оригинальные работы. В ходе разработки приложения решены следующие задачи:

* проведён обзор имеющихся методов выявления плагиата в текстах;
* проведен аналитический обзор аналогов и прототипов имеющихся подобных систем с цель определения необходимого функционала и формирования требований к разработке;
* разработано ПО, реализующее систему проверки уникальности текстов с целью выявления заимствований;
* применены шаблоны проектирования и объектно-ориентированный подход для обеспечения гибкости и масштабируемости системы;
* проведена верификация и апробация ПП.

При реализации веб-приложения использованы средства языка программирования *Python*, фреймворков *Flask* и *sentence-transformers*, а также библиотек *Google* *Search API*, *BeautifulSoup*, *NLTK* и *Scikit-learn*. Сетевое взаимодействие реализовано при помощи протокола *HTTP.*

Практическое *Hidden text* задание выполнено самостоятельно и проверена в системе «Антиплагиат». Процент оригинальности составляет 92%. Цитирования обозначены ссылками на публикации, указанные в «Списке использованных источников».

Спроектированы основные сущности системы. Для системы сравнения текстов дополнительно разработан и проведен ряд тестов с целью определения степени и корректности реализации всех необходимых функций для его работы и соответствия поставленным требованиям. Результаты тестирования показывают, что система способна производит качественное сравнение выбранных документов, искать в интернете ссылки на источники и в некоторой степени выявлять рерайтинг. Кроме того, приложение просто масштабировать, что позволяет добавлять новый функционал без изменения уже имеющегося.

**Список использованных источников**

1. «Гомельский государственный технический университет». – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Гомельский\_государственный\_технический\_универуниве/. – Дата доступа: 14.04.2024.
2. «Антиплагиат (*Антиплагиат.ру*)». – Электронные данные. – Режим доступа: https://ru.wikipedia.org/wiki/Антиплагиат. – Дата доступа: 15.04.2024
3. Программы для проверки уникальности текста. – Электронные данные. – Режим доступа: http://text-stati.ru/rerajting/proverka-unikalnosti-teksta/. – Дата доступа 15.04.2024.
4. Гукасян, Ц.Г. Методы и программные средства для выявления заимствования в текстах на армянском языке / Ц.Г. Гукасян – Москва: Российско-армянский университет 2021. – c. 45-70
5. Ишанкулиев Д.О., Миткина М.А. Сравнение методов и практические рекомендации по сопоставлению текстовых документов. // Экономика и предпринимательство, № 11, 2021. – с. 1460-1463
6. Hagen M., Potthast M., Stein B. Source Retrieval for Plagiarism Detection from Large Web Corpora: Recent Approaches / CLEF. – 2015.
7. Ayoub Ali M. Saeed, Alaa Yaseen Taqa, Using Retrieved Sources for Semantic and Lexical Plagiarism Detection // Iraqi Journal of Science, 2023.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

Добавлю при печати