АННОТАЦИЯ

Был спроектирован и реализован веб-сервис, позволяющий пользователю сравнивать два текста или один текст с базой документов и получать процентное содержание заимствований в тексте. Была разработана структура базы данных MySQL для хранения текстов. Была разработана многопоточная версия программы с применением технологий параллельных вычислений на CPU Intel(R) Xeon(R) E5 v3 и Intel(R) Xeon Phi(TM).

Количество страниц - 79, рисунков - 20, таблиц - 11, библиографических источников – 31.

ANNOTATION

The web service was designed and implemented, this allows the user to compare two texts or a text from the base documents and get the percentage of loans in the text. The structure of the MySQL database for storing texts was developed. Multi-threaded version of the program with the use of technologies of parallel computing on CPU Intel(R) Xeon(R) E5 v3 and Intel(R) Xeon Phi(TM) was developed.

Number of pages - 79, pictures - 20, tables - 11, references - 31.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 6](#_Toc516789638)

[1 АНАЛИЗ ОБЛАСТИ СРАВНЕНИЯ ТЕКСТОВЫХ ДОКУМЕНТОВ 8](#_Toc516789639)

[1.1 Понятие о плагиате 8](#_Toc516789640)

[1.2 Алгоритм шинглов для сравнения текстов 9](#_Toc516789641)

[1.3 Анализ актуальных систем проверки на плагиат 11](#_Toc516789642)

[1.3.1 Антиплагиат.ру и Антиплагиат ВУЗ 11](#_Toc516789643)

[1.3.2 Advego plagiatus 14](#_Toc516789644)

[1.3.3 Etxt антиплагиат 16](#_Toc516789645)

[1.3.4 Content watch 17](#_Toc516789646)

[1.3.6 Системы, разработанные студентами кафедры ЭВМиС ФЭВТ 19](#_Toc516789647)

[1.3.5 Результаты анализа систем проверки на плагиат 20](#_Toc516789648)

[2 ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ СРАВНЕНИЯ ТЕКСТОВЫХ ДОКУМЕНТОВ 23](#_Toc516789649)

[2.1 Функциональные особенности приложения 23](#_Toc516789650)

[2.2 Проектирование структуры приложения 24](#_Toc516789651)

[2.3 Выбор средств разработки 25](#_Toc516789652)

[2.3.1 Фреймворк ASP.NET Core 25](#_Toc516789653)

[2.3.2 Веб-API приложение 28](#_Toc516789654)

[2.3.3 Веб-приложение MVC 30](#_Toc516789655)

[2.3.4 СУБД MySQL 32](#_Toc516789656)

[2.4 Проектирование структуры веб-приложения с учетом возможностей, предоставляемых вычислительным кластером ВолгГТУ 34](#_Toc516789657)

[3 РЕАЛИЗАЦИЯ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ СРАВНЕНИЯ ТЕКСТОВЫХ ДОКУМЕНТОВ 36](#_Toc516789658)

[3.1 Реализация веб-API приложения 36](#_Toc516789659)

[3.1.1 Реализация алгоритма сравнения текстов 36](#_Toc516789660)

[3.1.2 API для взаимодействия с клиентами 43](#_Toc516789661)

[3.1.3 Реализация взаимодействия приложения с базой данных 51](#_Toc516789662)

[3.2 Реализация клиентского веб-приложения MVC 56](#_Toc516789663)

[3.2.1 Внешний вид веб-приложения 56](#_Toc516789664)

[3.2.2 Система авторизации 64](#_Toc516789665)

[3.2.3 Взаимодействие с API сравнения текстов 67](#_Toc516789666)

[4 ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ 68](#_Toc516789667)

[ЗАКЛЮЧЕНИЕ 74](#_Toc516789668)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 78](#_Toc516789669)

ВВЕДЕНИЕ

В настоящее время наблюдается тенденция к усилению контроля качества студенческих работ и, как следствие, самого учебного процесса. Это связано с тем, что с развитием интернет технологий в глобальной сети можно найти исчерпывающую информацию практически по любой теме, и студенты высших учебных заведений, разумеется, пользуются этим. К сожалению, зачастую написание рефератов, семестровых, курсовых или дипломных работ сводится к заимствованию полностью или частично уже готовых работ. Для борьбы с этим некоторые ВУЗы используют различные системы проверки на плагиат, но у такого подхода есть ряд минусов, самый главный из которых - затратность покупки и поддержки лицензии программного обеспечения.

С 1 января 2016 года ВУЗы на законодательном уровне обязали проводить проверки выпускных квалификационных работ на наличие заимствований с последующим хранением документов в системе ВУЗа [1]. В связи с этим актуальна тема разработки собственной системы проверки на плагиат. Такая система имела бы свои недостатки, такие как более низкая функциональность и надежность по сравнению с популярными системами, но при этом обладала бы рядом достоинств: гибкость, адаптивность, расширяемость, а главное, она не требовала бы существенных вложений. Также необходимо учитывать, что на кафедрах уже хранится большой объем документов, которых нет в Интернет-базах, и которые можно использовать для проверки текстов в системе проверки на плагиат.

Работа направлена на создание веб-приложения с использованием актуальных технологий разработки, клиентская часть которого доступна из браузера, а серверная часть расположена на вычислительном узле кластера Факультета электроники и вычислительной техники Волгоградского государственного технического университета (ВолгГТУ). Таким образом достигаются следующие преимущества: быстродействие, благодаря вычислениям на высокопроизводительных устройствах и применению методов сокращения времени вычислений, гибкость и портативность, благодаря возможности удаленного доступа с любого устройства, имеющего доступ к сети. Будет реализован режим сравнения двух текстовых документов и режим сравнения текста с базой документов, а также дополнительных возможностей работы с базой документов: удаление, скачивание файла с базы, добавление нового файла в базу, просмотр информации о документе из базы.

Таким образом целью работыявляется разработка веб-сервиса для сравнения текстовых документов и выявления процента заимствований в них. На основе цели можно выделить следующие задачи работы:

* анализ существующих систем проверки на плагиат;
* анализ инструментов и технологий разработки и проектирование структуры веб-сервиса;
* реализация веб-сервиса на основе проведенного анализа;
* анализ и применение способов повышения быстродействия системы;
* перенос сервиса на кластер ВолгГТУ;
* тестирование реализованной системы.

Структура пояснительной записки включает в себя четыре главы. В первой главе произведен анализ области сравнения текстовых документов, во второй главе описано проектирование веб-приложения, в третьей главе описана реализация приложения, а в четвертой – тестирование.

1 АНАЛИЗ ОБЛАСТИ СРАВНЕНИЯ ТЕКСТОВЫХ ДОКУМЕНТОВ

1.1 Понятие о плагиате

Плагиат – умышленное присвоение авторства чужого произведения искусства или достижения науки, технических решений или изобретений [2].

Понятие плагиата может присутствовать во многих областях человеческой деятельности, таких как: художественная литература, поэзия, изобразительное искусство, кинематограф, наука и др. Не всегда можно однозначно определить присутствие плагиата, так как это понятие порой перекликается с сопредельными понятиями: «заимствование», «соавторство», «подражание» и т.п. Так как большинство создаваемых работ основаны на уже известных идеях, не принадлежащих автору, такое заимствование не может считаться плагиатом.

В любой научной сфере, в частности, в академических кругах плагиат студентов, преподавателей и научных работников не приветствуется. Студенты, виновные в выдаче чужих идей за свои, подлежат наказаниям, таким как плохая оценка за работу, учебный семестр, или даже отчисление. Но это зависит от серьезности отношения учебного заведения или преподавателя к проверке работ на плагиат. Многие учебные заведения не ведут контроль за тем, чтобы работы их студентов и сотрудников обладали высокой уникальностью. Также высокий уровень плагиата в студенческих работах может быть из-за того, что студенты не понимают в полной мере, что такое плагиат, не знают основных требований оформления источников и цитат или снисходительно относятся к наказаниям за использование плагиата в работах.

С развитием и повышением доступности интернет технологий проблема заимствования чужих идей принимает глобальные масштабы. Сложно без специальных средств в большом открытом информационном пространстве следить за соблюдением авторских прав и проверять степень уникальности новых работ. Для этого разрабатываются новые технологии, в частности системы антиплагиата.

1.2 Алгоритм шинглов для сравнения текстов

Шингл в переводе с английского – это ячейка, чешуйка, частичка, кирпичик. В сфере анализа текстов шингл означает те составные части, из которых строится текст. Впервые по отношению к методам анализа текстов это понятие ввел в употребление в 1997 году сотрудник компании «Yahoo!» Андрей Бродер [3].

Алгоритм шинглов – это алгоритм для поиска копий текста и выявления плагиата в тексте. На рисунке 1 представлена диаграмма алгоритма шинглов, который включает в себя следующие этапы [4]:

а) канонизация текста;

б) разбиение текста на наборы шинглов;

в) расчет контрольных сумм для шинглов;

г) сравнение массивов контрольных сумм текстов;

д) подсчет совпадений и определение результата.

Канонизация приводит текст в единую форму, все предлоги, союзы, знаки препинания и прочее удаляются, так как они не несут смысловой нагрузки. Также в ряде модификаций алгоритма шинглов могут удаляться прилагательные, а существительные приводиться к именительному падежу и единственному числу, либо вообще оставляются лишь корни слов [4].

Следующий этап – это разбиение текста на наборы шинглов. Шингл в данном случае – это выделенные из теста последовательности слов определенной длины, которые формируются внахлест [4]. К примеру, если имеется уже канонизированный текст «пришел домой сел компьютер делать уроки завтра» и длина шинглов равна трем, то для него будут следующие наборы шинглов: «пришел домой сел», «сел компьютер делать» и «делать уроки завтра».

Увеличение количества шинлов приводит к экспоненциальному росту количества операций, что сильно отражается на производительности, поэтому в алгоритме сравниваются не сами шинглы в виде текста, а наборы контрольных сумм (хэшей), рассчитанные для каждого шингла при помощи хэш-функций, таких как SHA1, CRC32, MD5 и т.п.

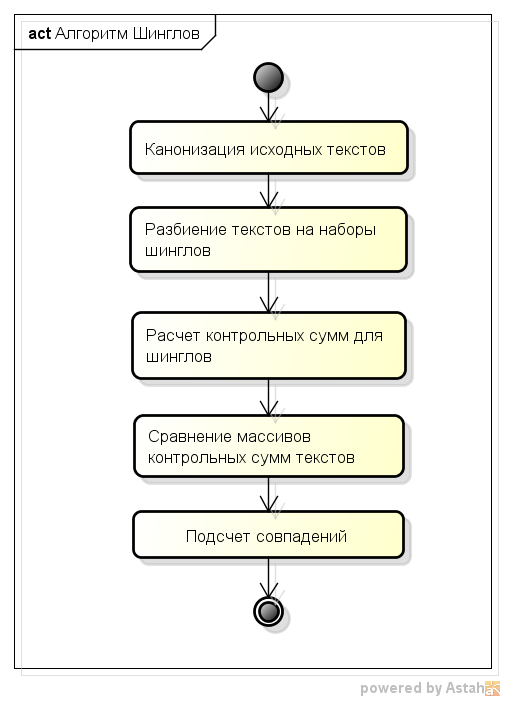


Рисунок 1 – Схема алгоритма шинглов

На следующем этапе выполняется сравнение набора рассчитанных контрольных сумм шинглов одного текста с набором другого. Для увеличения скорости проверки могут сравниваться не каждый набор с каждым, а только выбранные по определенному критерию, например, только четные или нечетные значения хэшей, или минимальные значения строки. После подсчета количества совпадений становится известен результат сравнения.

1.3 Анализ актуальных систем проверки на плагиат

В наше время разработано и активно используется большое количество различных систем проверки текстов на плагиат. Они бывают как платные, так и бесплатные, как имеющие свою собственную базу, размеры которой могут доходить до десятков миллионов документов, так и выполняющих поиск и сравнение текстов в сети Интернет. Для анализа были выбраны следующие популярные системы проверки на плагиат:

* Antiplagiat.ru;
* Антиплагиат ВУЗ;
* Advego plagiatus;
* Etxt антиплагиат;
* Content watch.

Далее будет подробно рассмотрена каждая из вышеперечисленных систем, её функционал, достоинства и недостатки, будет произведена оценка систем по выбранным критериям.

Следует также отметить, студентами кафедры ЭВМ и систем факультета электроники и вычислительной техники ВолгГТУ было разработано две системы: C.С. Серовым в 2012 году в рамках выпускной работы бакалавра; Полежаевым А.В. 2016 году, также в рамках выпускной работы бакалавра, и с тех пор достаточно успешно применяются на кафедре в качестве вспомогательного средства оценки схожести студенческих работ [9].

1.3.1 Антиплагиат.ру и Антиплагиат ВУЗ

«Антиплагиат ВУЗ» – это пакетное расширение опорной программы «Антиплагиат.ру», ее закрытая платная корпоративная версия, доступная только преподавательскому составу высших учебных заведений, заключивших с разработчиками контракт на пользование системой. Оба варианта работают по одному алгоритму обнаружения совпадений, но у «Антиплагиата ВУЗ» расширенная база поиска, которая формируется для каждой образовательной организации с учетом ее профиля. В нее обязательно включаются [5]:

* работы выпускников прошлых лет; печатные источники, чаще всего используемые в учебном процессе;
* коллекции электронных библиотек Book.ru, «Университетская система online», «Юрайт», «Айбукс», «БиблиоРоссика», «Лань»;
* документы модуля «Кольцо вузов».

«Антиплагиата ВУЗ» также дополнительно включает в себя интерфейс взаимодействия преподавателей и студентов, систему присвоения тэгов документам, дополнительные библиотеки с огромным количеством документов и др. Стоимость системы «Антиплагиат ВУЗ» составляет несколько сотен тысяч рублей.

Самостоятельно по «Антиплагиат ВУЗ» проверить текст перед сдачей преподавателю не получится, так как варианта программы для скачивания или свободной онлайн-проверки нет.

«Антиплагиат.ру» предоставляет частным клиентам четыре типа подписки: бесплатную, базовую, продвинутую и высоконагруженную, каждая из которых отличается функционалом, количеством отчетов и длительностью подписки (рисунок 2) [5].

Пользование бесплатной версией доступно сразу после регистрации на сайте проекта. В личном кабинете можно один раз в 6 минут проверять на уникальность как документ, там и скопированный в специальное поле текст. При проверке документа ограничения на количество символов нет. При использовании бесплатной подписки по результатам проверки можно получить краткий отчет, который содержит показатели заимствования и цитирования текста документа в процентных во всем документе и по источникам отдельно, а также относительные показатели черного и белого заимствования (рисунок 3).



Рисунок 2 – Тарифы системы «Антиплагиат.ру» для частных лиц

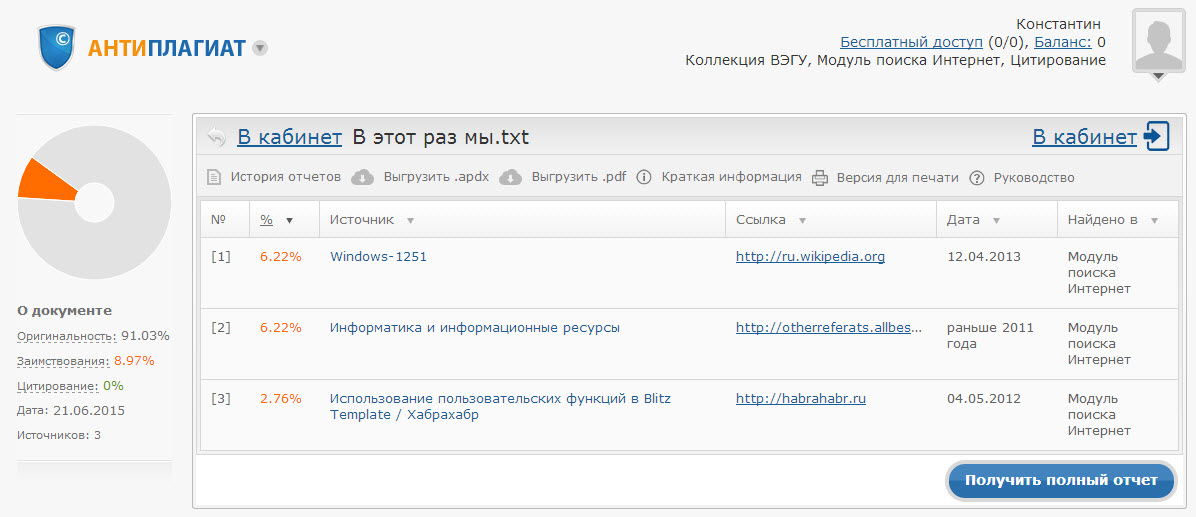


Рисунок 3 – Внешний вид сервиса «Антиплагиат»

Также в данном тарифе имеется ограничение на формат загружаемого файла – доступны только pdf или txt. Техническая поддержка при использовании бесплатного тарифа доступна только на форуме проекта.

Однако, при использовании платного тарифа все вышеперечисленные ограничения снимаются, остается лишь ограничение на количество проверок за день, неделю, месяц или полгода.

При использовании платных тарифов: базового стандартного или высоконагруженного можно осуществлять от 10 до 1000 проверок за выбранный срок, соответственно, и стоимость тарифа будет различаться – от 150 рублей за 10 проверок в день для тарифа «Базовый», до 11500 рублей за 1000 проверок в месяц. Таким образом, цена одной проверки составляет 15 рублей. После анализа будет доступен полный отчет, включающий в себя, как и процент заимствований и цитирований, так и текст документа с размеченными фрагментами. Также в данном типе подписке техническая поддержка может осуществляться на сайте проекта, по e-mail и по телефону, доступны функции редактирования и скачивания отчета.

Стандартно поиск совпадений осуществляется среди ресурсов сети Интернет, но за отдельную плату можно подключить дополнительные модули поиска: «Коллекция LEXPRO», «Коллекция Гарант», «Коллекция РГБ», «Коллекция eLIBRARY.RU», и др. При подключении каждого модуля стоимость проверки одной работы может увеличиться на сумму 8-75 рублей в зависимости от выбранного модуля [5].

Проверка работы осуществляется по очереди поступивших работ. У пользователей базовой и стандартной подписок приоритет в очереди выше, чем при использовании обычной подписки.

Компания «Антиплагиат» занимается разработкой системы «Антиплагиат ВУЗ» специально для высших учебных заведений.

1.3.2 Advego plagiatus

«Advego Plagiatus» - программа поиска в интернете частичных или полных копий текстового документа, разработанная создателями биржи контента «Advego». Данный программный продукт является бесплатным.

После установки на компьютер пользователю доступны два типа проверки: глубокая и быстрая. Глубокая проверка осуществляется по заданным параметрам по всем доступным ресурсам сети Интернет, быстрая проверка также осуществляется по заданным настройкам, но не по всем сайтам и при выборке меньшего количества частей текста, в результате чего она занимает меньше времени.

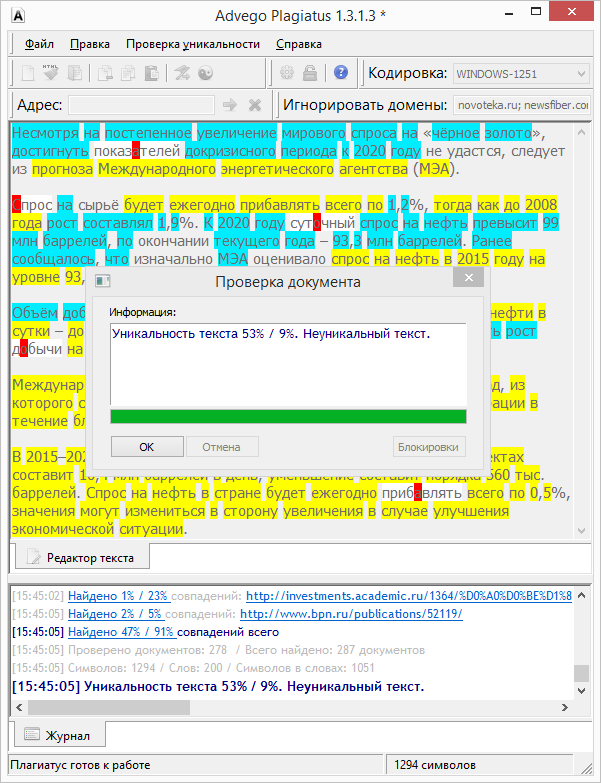


Рисунок 4 - Внешний вид программы «Advego Plagiatus»

«Advego Plagiatus» дает возможность пользователям настраивать проверку по следующим параметрам: прокси сервер, ограничение на размер загружаемых страниц, таймаут (время ожидания отклика от сайта), размер шингла и поисковой фразы, поисковые системы и другим. Также можно указать игнорируемые адреса и кодировку текста [6].

По результатам проверки составляется отчет, включающий в себя процент уникальности, процент уникальности с учетом рерайта (по мнению «Advego»), список ресурсов, в которых были найдены совпадения и процент этого совпадения. В тексте желтым цветом выделяются неуникальные фразы, голубым цветом выделяется рерайт (рисунок 4).

1.3.3 Etxt антиплагиат

«eTXT Антиплагиат» предоставляет возможность как и онлайн проверки текста на уникальность, так и при помощи программы от тех же разработчиков, которая устанавливается на компьютер (рисунок 5).

Пользователь может сделать несколько бесплатных онлайн проверок теста с ограничением в 5000 символов, после чего стоимость проверки 1000 символов будет составлять 1.5 рубля. При платной проверке результаты можно хранить на сервере [7].

Программа «eTXT Антиплагиат» является бесплатной. Она предоставляет три режима проверки на уникальность, которые отличаются друг от друга скоростью и глубиной проверки. Режим пакетной проверки позволит поставить на проверку несколько текстовых файлов.

Настройки программы включают:

* размер выборки;
* число выборок;
* число ссылок на выборку;
* порог уникальности;
* поисковые системы;
* использование прокси;
* таймаут загрузки страницы и другие.

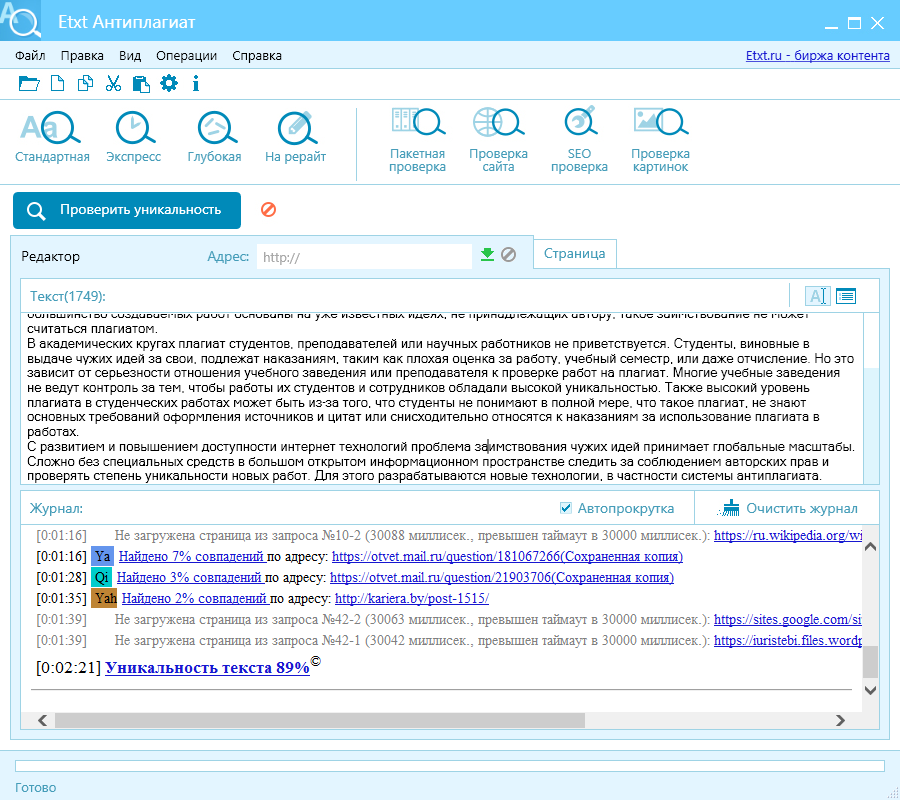


Рисунок 5 – Внешний вид программы «eTXT Антиплагиат»

1.3.4 Content watch

«Content Watch» – сервис для проверки уникальности теста или контента сайта. Поиск осуществляется по ресурсам сети Интернет. Внешний вид ресурса представлен на рисунке 5. Разработчики сообщают, что в их сервисе используется алгоритм собственной разработки, который проверяет текст по смысловому содержанию, выполняет более качественные и быстрые проверки, нежели альтернативные ресурсы, большая часть которых работает на основе алгоритма шинглов [8].

Бесплатно «Content Watch» можно пользоваться не больше 7 раз в день, проверяя на уникальность тексты длиной до 10 тысяч символов, но используя платные подписки «Копирайтер», «Команда» и «Агентство» можно поднять планку ограничений, а также получить доступ к таким функциям, как история проверок за месяц, проверка больших текстов, доступ с любого компьютера. Первая подписка обойдется в 140 рублей в месяц и позволит выполнять 150 проверок в день текста; вторая подписка стоит 590 рублей в месяц и дает доступ к 1000 проверкам текста; стоимость третьей подписки составляет 1490 рублей в месяц, и при помощи нее можно будет проверять на уникальность до 3000 документов в день. Во всех типах подписки действует ограничение на длину текста равное 20 тысячам символов. Следует отметить, что Content Watch предоставляет API разработчикам веб-приложений. Стоимость одной проверки с использованием API составляет 25 копеек, но клиенту может быть предоставлена скидка от 20 до 60% при активном использовании сервиса [8].

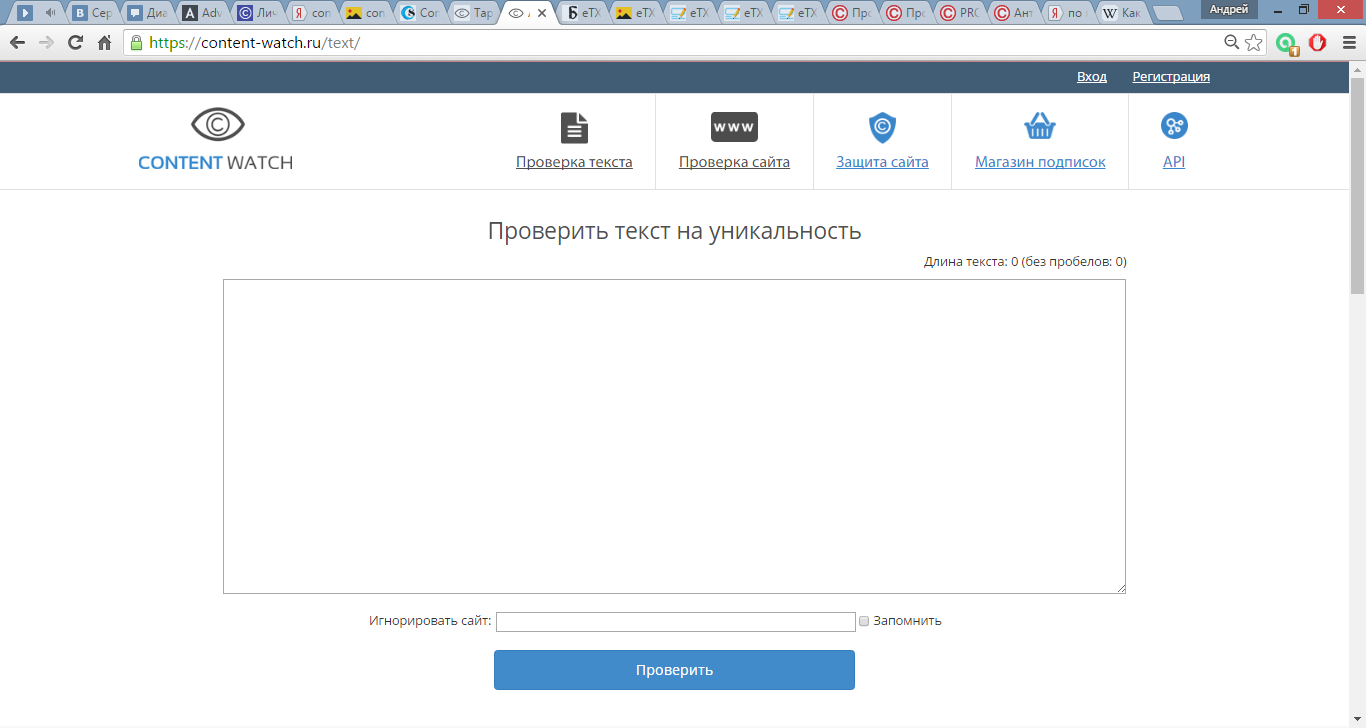


Рисунок 6 – Внешний вид сервиса «Content Watch»

1.3.6 Системы, разработанные студентами кафедры ЭВМиС ФЭВТ

Одна из систем проверки схожести текстов была разработана C.С. Серовым в 2012 году [9].

Данная программа использует алгоритм шинглов в упрощенной формулировке и написана на языке C#, представляя собой настольное Windows-приложение с графическим интерфейсом пользователя (рисунок 7) [9].

Можно сравнивать как два документа, так и документ с выбранным каталогом документов. По окончанию проверки система выдает процент заимствований каждой проверяемой пары документов. На рисунке 6 представлен внешний вид сервиса.

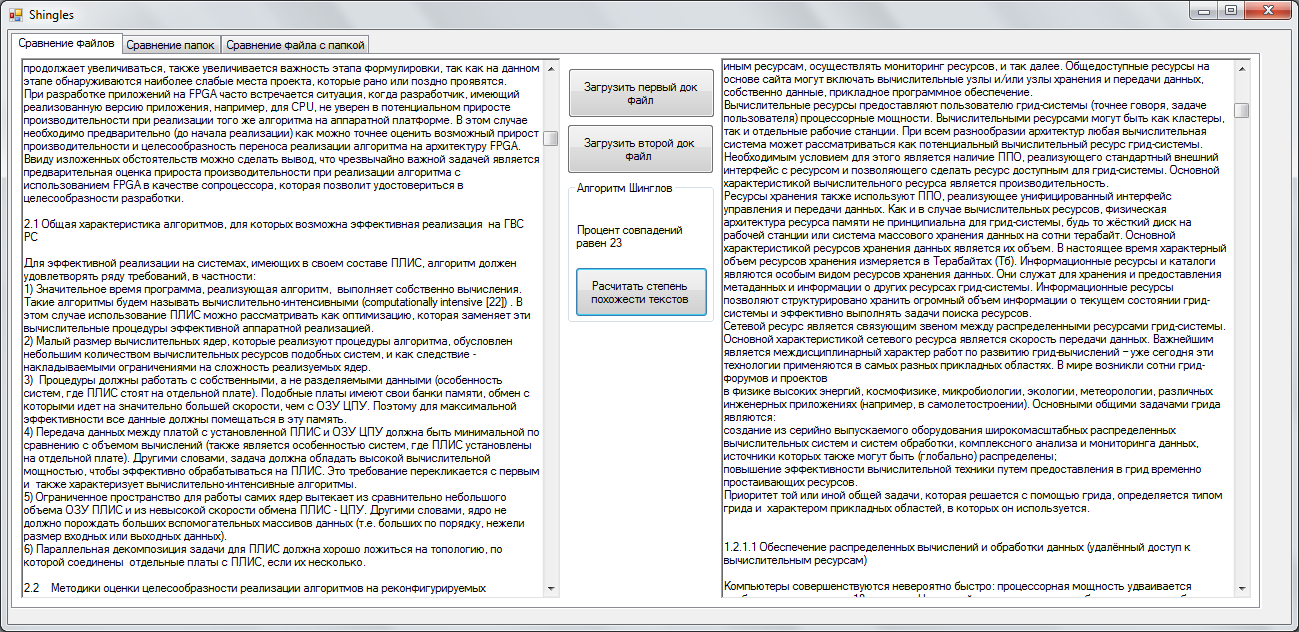


Рисунок 7 – Внешний вид системы Серова С.С.

Система, разработанная студентом Полежаевым А.В. в 2016 году, представляет собой веб-приложение, состоящее из клиентской и серверной части. Внешний вид представлен на рисунке 8. Клиентская часть представляет собой веб-приложение, серверное – вычислительный сервис. Также данная система уже имеет свою базу текстовых документов, в которую можно загружать текстовые файлы формата txt.

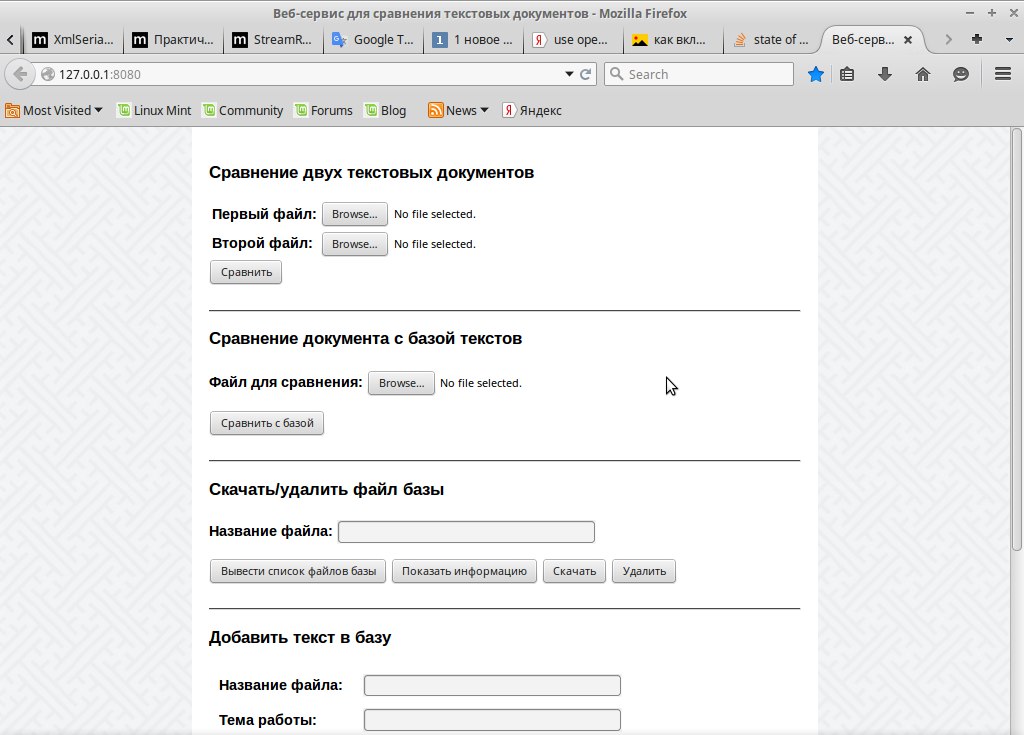


Рисунок 8 – Внешний вид системы Полежаева А.В.

1.3.5 Результаты анализа систем проверки на плагиат

Для примера возьмем ситуацию, когда кафедре учебного заведения необходимо проверять студенческие работы на уникальность. Так как на уникальность обычно проверяются курсовые или дипломные работы, возьмем средний размер текста – 5000 слов. Предположим, что каждый день потребуется проверять 20-30 работ, то есть около 600-900 проверок в месяц.

Были выделены наиболее важные свойства системы проверки на плагиат:

* Параметры проверки – это оценка широты предоставляемых настроек проверки.
* Отчетность – оценка объема представляемой по итогам анализа отчетности. Предоставляет ли система помимо итогового количества найденных совпадений отчет с источниками, выделенными фрагментами совпадающих участков текста и т.п.
* Скорость проверки.
* Качество проверки. Более качественной считается та система, которая находит больше совпадений среди своих аналогов при одинаковом исходном тексте.
* Стоимость. В качестве стоимости бралась средняя цена лицензии, необходимой для нормальной работы в течении дня, сумма указана за месяц пользования услугами.

Для выявления качества проверки был произведен анализ на уникальность одного и того же текста, состоящего из 5000 слов. В результате «Антиплагиат» показал 100% уникальность, «Advego Plagiatus» – 89%, «eTXT Антиплагиат» – 89%, «Content watch» – 75%. Следует отметить, что низкие проценты уникальности, показанные системой в результате анализа, не обязательно означают то, что производилась более глубокая проверка, или что работа некачественна. Сервисы могут видеть плагиат в типовых, общих фразах или в коротких словосочетаниях если неправильно выставить параметры анализа. Баллы по всем критериям, кроме стоимости, выставлялись по пятибалльной шкале на основании субъективного мнения автора работы. Стоимость указана в рублях за месяц пользования системой для указанной выше ситуации. Результаты анализа представлены в таблице 1.

Таблица 1- результаты анализа систем проверки на плагиат

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Параметры проверки | Антиплагиат | Advego Plagiatus | eTXT Антиплагиат | Content watch |
| Параметры проверки | 5 | 5 | 5 | 1 |
| Отчетность | 5 | 4 | 4 | 1 |
| Скорость проверки | 3 | 2 | 2 | 3 |
| Качество проверки | 3 | 4 | 4 | 5 |
| Стоимость в месяц, руб. | 10 000 | 0 | 0 | 590 |

Таким образом, на сегодняшний момент существует множество готовых систем, при помощи которых можно проверять студенческие работы. Но использование таких систем для проверки большого числа работ в рамках кафедры или всего учебного заведения не всегда удобно и оптимально. Большинство ресурсов не обладает достаточным качеством проверки или быстродействием и многие из них затратные в плане вложений.

Достоинствами приложения Серова С.С. по сравнению с популярными аналогичными системами являются: независимость от подключения к сети Интернет, отсутствие ограничения на объем сравниваемых текстов, не требует платной лицензии. Недостатками являются: невысокая скорость работы, отсутствие режима сравнения с материалами из Интернета, отсутствие собственной базы текстовых документов.

Первым главными отличиям данной системы Полежаева А.В. от приложения Серова С.С. является сама структура, то есть разделение по функционалу, что позволяет иметь доступ из браузера с любого устройства, но при этом можно подключиться к сервису с другого разработанного клиентского приложения, например, приложения на ОС Android. Вторым из основных отличий является быстродействие системы за счет использования технологий параллельных вычислений и возможности размещения вычислительного сервиса на высокопроизводительной системе.

Однако данная система нуждается в доработке:

* Внешний вида и функционал клиентского веб-приложения.
* Структура базы данных, а также технология ее реализации, что может повлиять на быстродействие системы.
* Работа базы данных только с текстовыми документами формата «txt».
* Технологии разработки – с момента реализации системы вышли новые технологии реализации веб-приложений и сервисов, которые превосходят по производительности те, что использовались при разработке данной системы.
* Система авторизации – для полноценного удаленного пользования системой необходимо реализовать систему защиты доступа к приложению.

2 ПРОЕКТИРОВАНИЕ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ СРАВНЕНИЯ ТЕКСТОВЫХ ДОКУМЕНТОВ

2.1 Функциональные особенности приложения

Основная функция разрабатываемого приложения – предоставление пользователю возможности сравнения текстов и подсчет количества совпадений в них.

Можно выделить следующие прецеденты веб-приложения (рисунок 9):

* cравнение двух текстов;
* сравнение текста с базой текстов;
* поиск текстового документа в базе;
* добавление текстового документа в базу;
* удаление текстового документа из базы;
* изменение информации о текстовом документе базы;
* скачивание документа из базы.



Рисунок 9 – Диаграмма прецедентов веб-приложения

Доступ к приложению должен осуществляться с мобильного или стационарного устройства, подключенного к сети сервера, через браузер.

2.2 Проектирование структуры приложения

Структура приложения включает в себя две части: клиентскую и серверную. Разделение было сделано для того, чтобы отделить вычислительную часть, требующую затрат вычислительных ресурсов от клиентской, которую при данной структуре можно реализовать как веб-приложение, десктоп-приложение или приложение для ОС Android. Таким образом, к серверу могут подключаться несколько клиентов с различных устройств. Структурная схема приложения показана на рисунке 10.

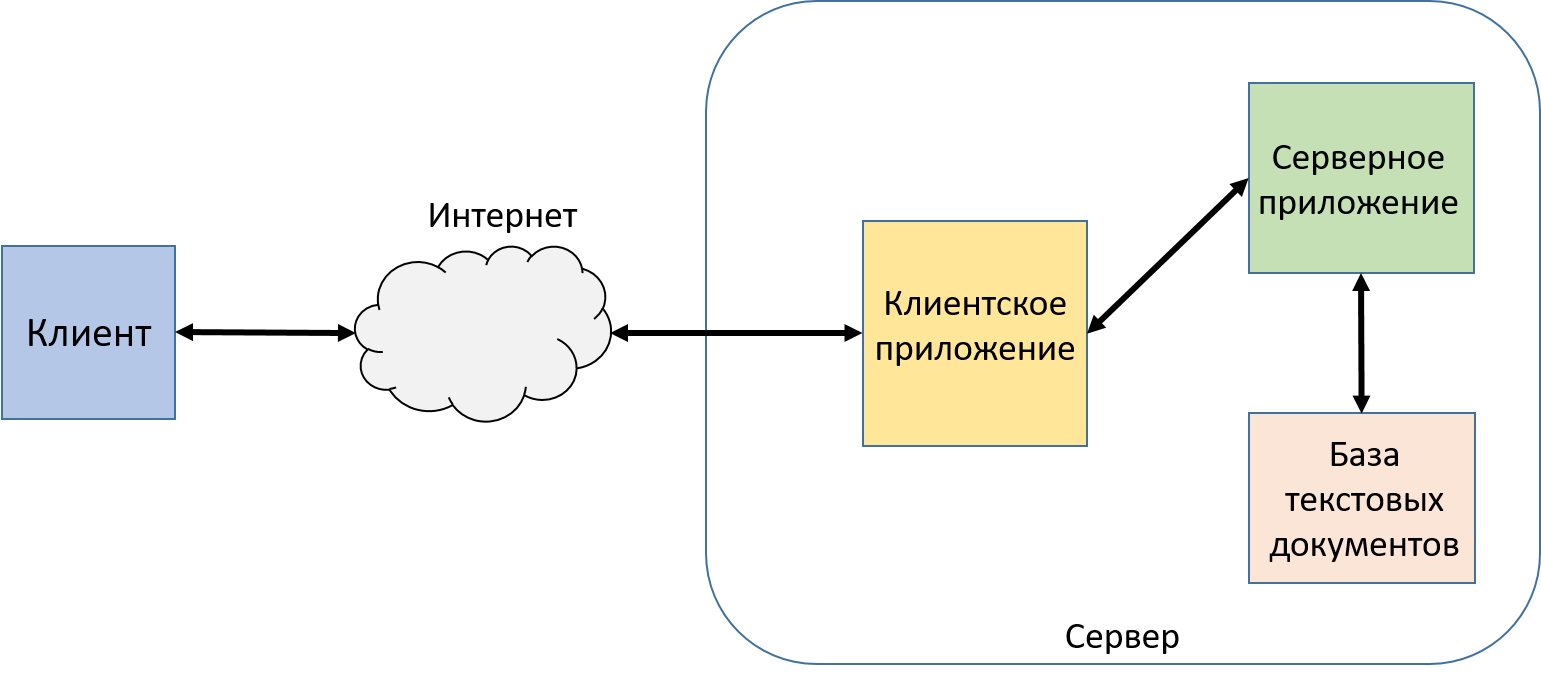


Рисунок 10 – Структура разрабатываемого веб-приложения

На приведенной на рисунке 8 схеме клиентское приложение развертывается на сервере вместе с серверным приложением, однако на практике это не должно обязательно быть так. Так, клиентское приложение может находиться на другом сервере, либо, в случае выступающего в качестве клиентской части десктоп-приложения, на компьютере пользователя. Та же ситуация и с базой данных. В этом состоит еще одно преимущество данной структуры.

2.3 Выбор средств разработки

В данной работе в качестве клиентского приложения будет разрабатываться веб-приложение, доступ к которому пользователь будет получать через браузер на своем устройстве. На рисунке 11 представлена структурная схема приложения для сравнения текстов с учетом выбранных технологий, которые будут описаны далее.

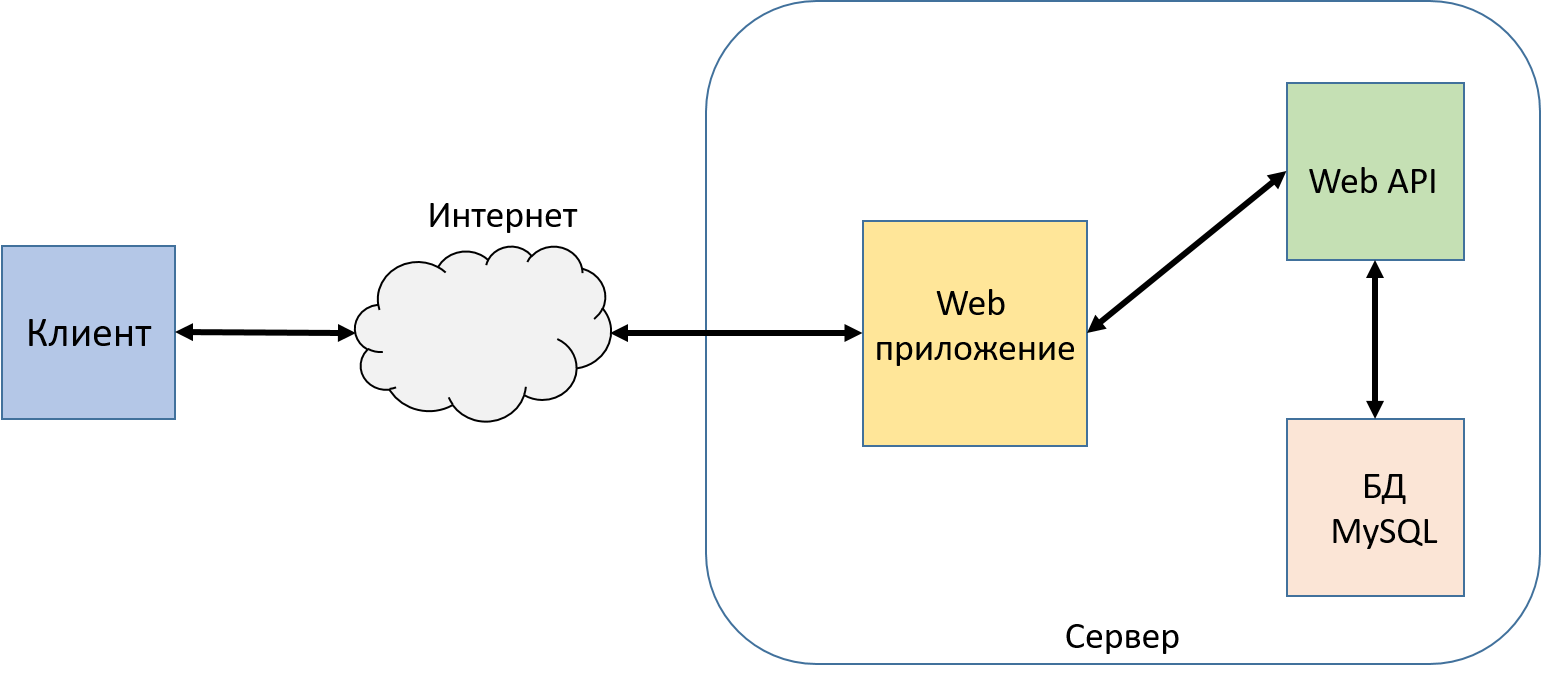


Рисунок 11 – Структура разрабатываемого веб-приложения

2.3.1 Фреймворк ASP.NET Core

ASP.NET Core является кроссплатформенной, высокопроизводительной средой с открытым исходным кодом для создания современных облачных приложений, подключенных к Интернету [11]. ASP.NET Core позволяет выполнять следующие задачи [12]:

* Создавать веб-приложения и службы, приложения IoT и серверные части для мобильных приложений.
* Использовать избранные средства разработки в Windows, macOS и Linux.
* Выполнять развертывания в облаке или локальной среде.
* Работать в .NET Core или .NET Framework.

ASP.NET Core – это модификация ASP.NET 4.x с архитектурными изменениями, формирующими более рациональную и более модульную платформу. ASP.NET Core имеет следующие преимущества:

1. Единое решение для создания пользовательского веб-интерфейса и веб-API.
2. Интеграция современных клиентских платформ и рабочих процессов разработки.
3. Облачная система конфигурации на основе среды.
4. Встроенное введение зависимостей.
5. Упрощенный высокопроизводительный модульный конвейер HTTP-запросов.
6. Возможность размещения в IIS, Nginx, Apache, Docker и др.
7. Параллельное управление версиями приложения, ориентированное на .NET Core.
8. Инструментарий, упрощающий процесс современной веб-разработки.
9. Возможность сборки и запуска в ОС Windows, macOS и Linux.
10. Открытый исходный код и ориентация на сообщество.

ASP.NET Core MVC предоставляет функции, которые позволяют создавать веб-интерфейсы API и веб-приложения. Шаблон Model-View-Controller (MVC) помогает сделать веб-API и веб-приложения тестируемыми.

Страницы Razor (новый компонент в ASP.NET Core 2.0) — это основанная на страницах модель программирования, которая упрощает создание пользовательского веб-интерфейса и повышает его эффективность. Разметка Razor предоставляет эффективный синтаксис для страниц представлений MVC. Вспомогательные функции тегов позволяют серверному коду участвовать в создании и отображении HTML-элементов в файлах Razor.

Благодаря встроенной поддержке нескольких форматов данных и согласованию содержимого веб-API становятся доступными для множества клиентов, включая браузеры и мобильные устройства.

Привязка модели автоматически сопоставляет данные из HTTP-запросов с параметрами методов действия. Проверка модели автоматически выполняется на стороне сервера и клиента.

ASP.NET Core легко интегрируется с распространенными клиентскими платформами и библиотеками, в том числе Angular, React и Bootstrap.

Приложения ASP.NET Core могут выполняться в .NET Core или .NET Framework. Приложения ASP.NET Core, предназначенные для .NET Framework, не являются кроссплатформенными – они выполняются только в Windows. Отказ от поддержки .NET Framework в ASP.NET Core не планируется. Как правило, ASP.NET Core состоит из библиотек .NET Standard. Приложения, написанные для .NET Standard 2.0 запускаются везде, где есть поддержка .NET Standard 2.0 [13].

При использовании .NET Core существуют некоторые преимущества, и их число увеличивается с каждым выпуском. Преимущества .NET Core по сравнению с .NET Framework:

* кроссплатформенность (выполняется на macOS, Linux и Windows);
* повышение производительности;
* управление параллельными версиями;
* новые интерфейсы API;
* открытый исходный код;

На основании этого, в качестве фреймворка для создания приложения был выбран .NET Core 2.0, который позволяет создавать приложения с поддержкой разных платформ (Windows, Linux и macOS) [11]. Высокопроизводительная серверная среда выполнения для Windows Server и Linux делает .NET самой эффективной веб-платформой в тестах TechEmpower [13].

2.3.2 Веб-API приложение

В качестве серверной части будет реализовано ASP .NET Core Web API приложение.

Web API представляет способ построения приложения ASP.NET, который специально заточен для работы в стиле REST (Representation State Transfer или «передача состояния представления»). REST-архитектура предполагает применение следующих методов или типов запросов HTTP для взаимодействия с сервером:

* GET;
* POST;
* PUT;
* DELETE.

Зачастую REST-стиль особенно удобен при создании всякого рода Single Page Application, которые нередко используют специальные javascript-фреймворки типа Angular, React или Knockout [14]. По сути Web API представляет собой веб-службу, к которой могут обращаться другие приложения. Причем эти приложения могут представлять любую технологию и платформу - это могут быть веб-приложения, мобильные или десктопные клиенты.

На рисунке 12 представлена структура веб API приложения. В качестве клиента может выступать любой компонент, использующий API (мобильное приложение, браузер и т. д.). Модель – это объект, представляющий данные в приложении. В данной схеме модель представлена результатом сравнения двух текстов, состоящего из полей: процент совпадений и промежуток времени сравнения. Модели представлены классами C#. Контроллер – это объект, который обрабатывает HTTP-запросы и создает HTTP-ответ.

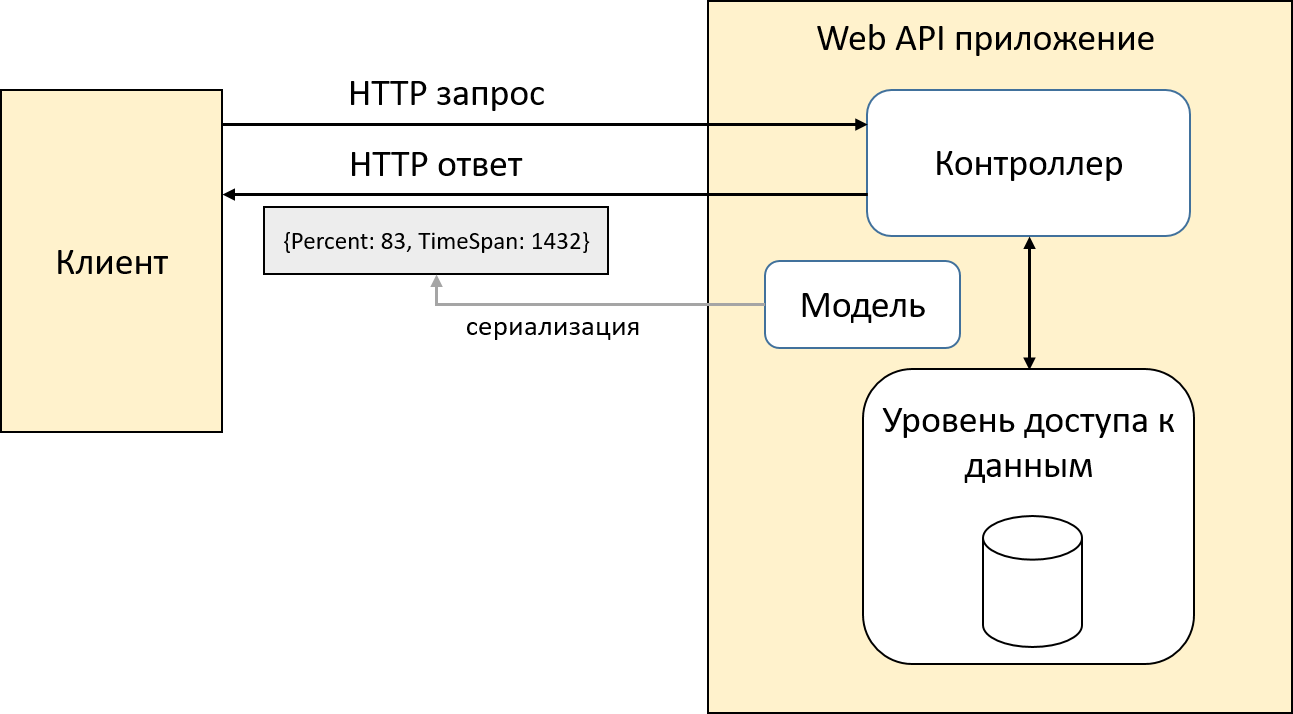


Рисунок 12 – Структура Web API приложения

Запросы к контроллеру работы с текстовыми документами будут выглядеть следующим образом, согласно архитектуре REST:

1. GET api/file/{id} – получение текстового файла из базы по идентификатору.
2. POST api/file – добавление файла, переданного в запросе в базу, данные в теле запроса.
3. PUT api/file – изменение информации о файле, данные в теле запроса.
4. DELETE api/file/{id} – удалить файл из базы.

Для последующего увеличения производительности работы приложения с использованием параллельных вычислений, вычислительную часть, а именно метод шинглов, необходимо реализовать на языке C++.

Механизм Platform Invoke, более известный как P/Invoke, позволяет вызывать из управляемого кода функции в стиле языка C, экспортируемые библиотеками DLL [15]. Чтобы воспользоваться механизмом P/Invoke, управляемый код должен объявить статический внешний (static extern) метод с сигнатурой (типы параметров и тип возвращаемого значения), эквивалентной функции на языке C/C++. Сам метод должен быть снабжен атрибутом DllImport, определяющим, как минимум, библиотеку DLL, экспортирующую требуемую функцию. Процесс создания моста между управляемым и неуправляемым кодом называется маршалингом [16].

Схема вызова функция неуправляемого кода динамической библиотеки представлена на рисунке 13.

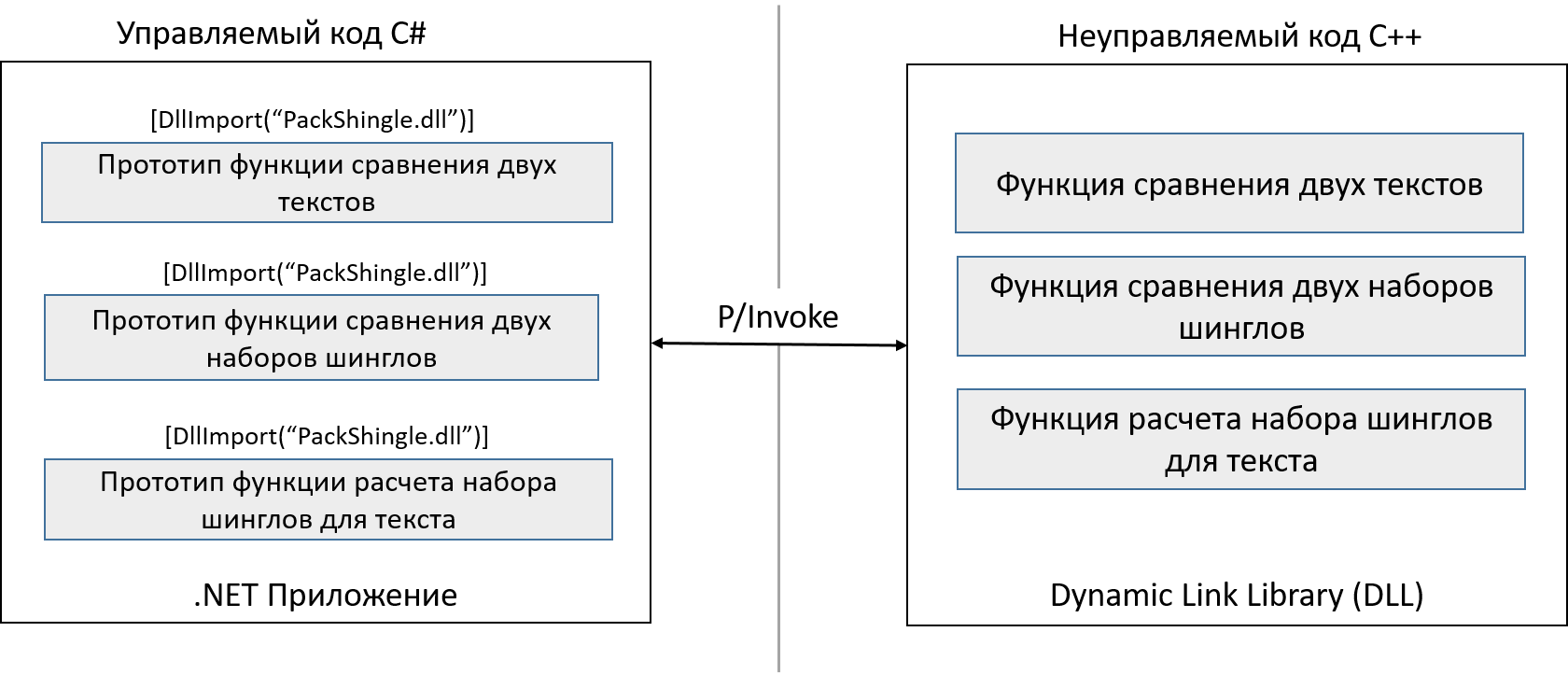


Рисунок 13 – Схема вызова функция неуправляемого кода динамической библиотеки

В динамической библиотеке будут содержаться все функции, реализующие алгоритм шинглов (канонизация, получение набора контрольных сумм и др.). В .NET приложении будут реализованы функции, необходимые для взаимодействия с клиентами, функции работы с базой документов, обертка для использования функций библиотеки.

2.3.3 Веб-приложение MVC

В качестве клиентского приложения необходимо реализовать веб-приложение MVC.

Концепция паттерна (шаблона) MVC (model - view - controller) предполагает разделение приложения на три компонента [17]:

1. Контроллер (Controller) представляет класс, обеспечивающий связь между пользователем и системой, представлением и хранилищем данных. Он получает вводимые пользователем данные и обрабатывает их. И в зависимости от результатов обработки отправляет пользователю определенный вывод, например, в виде представления.
2. Представление (View) - это собственно визуальная часть или пользовательский интерфейс приложения. Как правило, html-страница, которую пользователь видит, зайдя на сайт.
3. Модель (Model) представляет класс, описывающий логику используемых данных.

В этой схеме модель является независимым компонентом – любые изменения контроллера или представления не затрагивают модель. Контроллер и представление являются относительно независимыми компонентами, и нередко их можно изменять независимо друг от друга.

Благодаря этому реализуется концепция разделение ответственности, в связи с чем легче построить работу над отдельными компонентами. Кроме того, вследствие этого приложение обладает лучшей тестируемостью [17].

Конкретные реализации и определения данного паттерна могут отличаться, но в силу своей гибкости и простоты он стал очень популярным в последнее время, особенно в сфере веб-разработки.

Структура приложения разделена на два раздела:

1. блок сравнения текстов (двух текстов, текста с базой);
2. блок работы с файлами (загрузка файлов, поиск, удаление и пр.).

Соответственно, предполагается, что для данных разделов будут созданы контроллеры.

Для создания внешнего вида страниц представлений будут использоваться:

* язык разметки HTML5;
* таблицы стилей CSS3;
* библиотека jQuery;
* CSS фреймворк Bootstrap 3.

jQuery – это библиотека JavaScript, фокусирующаяся на взаимодействии JavaScript и HTML. Библиотека jQuery помогает легко получать доступ к любому элементу DOM, обращаться к атрибутам и содержимому элементов DOM, манипулировать ими [18]. Также библиотека jQuery предоставляет удобный API для работы с AJAX.

Bootstrap (также известен как Twitter Bootstrap) – свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения [19].

Так веб-приложение будет доступно из сети Интернет, необходимо реализовать систему аутентификации и авторизации.

2.3.4 СУБД MySQL

База данных – это совокупность связанных данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования, независимая от прикладных программ. База данных является информационной моделью предметной области. Обращение к базам данных осуществляется с помощью системы управления базами данных (СУБД). СУБД обеспечивает поддержку создания баз данных, централизованного управления и организации доступа к ним различных пользователей.

MySQL - это одна из самых распространенных серверных СУБД. MySQL работает с реляционными базами данных. В реляционной базе данных данные хранятся не все скопом, а в отдельных таблицах, благодаря чему достигается выигрыш в скорости и гибкости. MySQL обладает широким функционалом, свободно распространяемая СУБД, которая успешно работает с различными сайтами и веб-приложениями [20].

Основные преимущества MySQL [20]:

* многопоточность, поддержка нескольких одновременных запросов;
* оптимизация связей с присоединением многих данных за один проход;
* записи фиксированной и переменной длины;
* гибкая система привилегий и паролей;
* гибкая поддержка форматов чисел, строк переменной длины и меток времени;
* интерфейс с большинством распространенных языков программирования;
* быстрая работа, масштабируемость;
* бесплатна лицензия в большинстве случаев;
* хорошая поддержка со стороны провайдеров услуг хостинга;
* быстрая поддержка транзакций через механизм InnoDB.

Для повышения быстродействия системы, в базе данных будут храниться уже рассчитанные наборы контрольных сумм шинглов текстов. Рассчитываться они будут при добавлении документа в базу. Также в базе данных будут хранится следующие данные:

* идентификатор файла;
* название докумена;
* имя и размер файлового документа;
* описание;
* дата загрузки на сервер;
* путь к файлу на сервере.

Таким образом, не нужно будет делать данные вычисления непосредственно во время сравнения. При этом оригинал файла будет сохраняться в директории на сервере. Пути к этим файлам также будут записываться в таблицу базы. Также в БД будет записываться основная информация о документе: имя файла, размер, описание и др. Структура таблицы БД представлена в таблице 2.

Наборы хэшей, рассчитанных для шинглов документов хранятся в БД в стерилизованном виде.

Таблица 2 – Структура БД разрабатываемого приложения

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| Id | INT UNSIGNED | Идентификатор файла |
| Name | NVARCHAR(128) | Название |
| FileName | NVARCHAR(128) | Имя файла |
| Description | NVARCHAR(256) | Описание |
| Date | DATE | Дата загрузки |
| Size | BIGINT | Размер файла |
| Path | NVARCHAR(128) | Путь к файлу на сервере |
| Shingles | LONGTEXT | Набор контрольных сумм |

2.4 Проектирование структуры веб-приложения с учетом возможностей, предоставляемых вычислительным кластером ВолгГТУ

Есть несколько вариантов использование ресурсов кластера ВолгГТУ для развертывания на нем разрабатываемого веб-приложения.

Первый подход был рассмотрен в рамках работы «Сокращение времени оценки схожести текстовых документов на неоднородной многопроцессорной вычислительной системе». Он заключается в использовании нескольких ускорителей Xeon Phi(TM) 1 поколения. При этом база файлов должна равномерно распределяться по ускорителям и храниться в памяти. Алгоритм был распараллелен с применением технологии OpenMP, а также производительность повышалась путем использования векторных регистров. Был разработан прототип программы для сравнения текстов, произведены замеры, показавшие целесообразность использования нескольких сопроцессоров Xeon Phi(TM) [9].

Следующий подход заключается в использовании нескольких платформ с CPU Xeon Phi(TM) второго поколения. База текстов при этом также распределяется между узлами, но может храниться как и в памяти, так и в текстовых файлах или в базе данных MySQL. Преимуществом такого варианта является использование регистров AVX 512, что позволяет увеличить производительность. В составе вычислительного кластера ВолгГТУ имеется 8 узлов CPU Xeon Phi(TM) второго поколения (KNL).

Третий подход основывается на запуске приложения на узле с двумя CPU Intel(R) Xeon(R) E5-2650 v3 с 20 ядрами. Работа с базой данных будет осуществляться как и в предыдущем подходе.

3 РЕАЛИЗАЦИЯ ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ СРАВНЕНИЯ ТЕКСТОВЫХ ДОКУМЕНТОВ

3.1 Реализация веб-API приложения

3.1.1 Реализация алгоритма сравнения текстов

Как отмечалось выше, метод шинглов для сравнения текстов будет реализовываться в качестве набора функций, собранных в динамической библиотеке (DLL).

Следует отметить, что данная реализация алгоритма основывается на уже достигнутых результатах, таких как приложение в рамках ВКР Серова С.С., приложение в рамках ВКР Полежаева А.В., работа на тему сокращения времени оценки схожести текстовых документов на неоднородной многопроцессорной вычислительной системе.

Алгоритм шинглов включает в себя следующие этапы:

1. канонизация текста;
2. разбиение текста на наборы шинглов;
3. расчет контрольных сумм для шинглов;
4. сравнение массивов контрольных сумм текстов;
5. подсчет совпадений и определение результата.

Для реализации алгоритма шинглов был создан специальный класс Shingle, который включает в себя следующие поля и методы:

class Shingle2

{

int Length;

vector<string> StopWords;

vector<string> StopSym;

public:

string Canonize(string source);

void setLength(int l);

Shingle();

int GetShingles(string Text, int\* buf);

int CompareShingles(int\* bufA, int\* bufB, int sizeA, int sizeB);

int Compare(string TextA, string TextB);

};

Для хранения предлогов, союзов, знаков препинания и подобных элементов, которые необходимо удалить на этапе канонизации, текста используется vector-контейнеры StopWords и StopSym, состоящие из элементов типа string; целочисленная переменная Lenght хранит длину шингла.

В конструкторе Shingle() происходит инициализация длины шинглов и контейнеров StopWords и StopSym при помощи метода push\_back:

StopSym.push\_back(",");

Для сравнения двух текстов вызывается метод Compare, на вход которого подаются две строки. В нем происходит объявление и выделение памяти под массивы контрольных сумм шинглов исходных текстов, после чего последовательно вызывается функции Canonize и GetShingles для каждого текста. Canonize получает на вход исходный текст в формате string. В этом тексте находятся и заменяются пробелами все лишние слова и символы, после чего остается «чистый» канонизированный текст готовый для разбиения на шинглы:

string Canonize(string source)

{

string Result = source;

string\_tolower(Result, std::locale(""));

string res=Result;

for(int i=0; i<StopSym.size(); i++) {

Result=res;

string find\_str = "" + StopSym[i];

res = replaceAll(Result, find\_str, " ");

}

Result.clear();

Result=res;

for(int i=0; i<StopWords.size(); i++) {

res = Result;

Result = replaceAll(res, " " + StopWords[i] + " ", " ");

}

res=Result;

Result=replaceAll(res, " ", " ");

return Result;

}

GetShingles получает на вход канонизированный текст и массив шинглов по указателю. В цикле формируются наборы шинглов из канонизированного текста, после чего для них сразу рассчитываются и добавляются в массив контрольные суммы:

for (int i = 0; i <= (Words.size() - count\_shingle); i++) {

string ShingleText = "";

for (int j = 0; j < count\_shingle; j++)

ShingleText += (Words[i + j] + " ");

int res = crc.FullCRC((const unsigned char\*)ShingleText.c\_str(), (size\_t)ShingleText.length());

buf[i] = res;

}

Здесь Words – это массив слов канонизированного текста, count\_shingle = Lenght – длина шингла. crc – это переменная класса CRC32, которые реализует алгоритм хэширования [21]. Метод FullCRC получает на вход набор шинглов и его длину и возвращает рассчитанный хэш для этого массива, который добавляется в массив контрольных сумм для поступившего текста.

После данных операций сравниваются полученные массивы контрольных сумм функцией CompareShingles в цикле:

for (int i=0; i < sizeA ; i++)

for (int j = 0; j < sizeB; j++)

if ( bufB[j] == bufA[i] )

{

arrMatches[i]++;

break;

}

Здесь sizeA и sizeB – количество контрольных сумм шинглов, рассчитанных для каждого текста, arrMatches – массив, хранящий количество совпадений шинглов второго текста с шинглами первого текста.

Итоговый процент совпадения вычисляется по формуле [22]:

, (1)

где matches – количество найденных совпадений; sizeA и sizeB – количество контрольных сумм для наборов шинглов сравниваемых текстов.

Из всех прецедентов системы наиболее затратными по времени являются «Сравнение двух текстов» и «Сравнение текста с базой документов». Для повышения производительности при реализации обработки шинглов можно использовать разные подходы.

Для рассматриваемой задачи возможно применение различных технологий параллельных вычислений, таких как OpenMP, OpenCL, CUDA и других.

Технологии OpenCL и CUDA связанны с параллельными вычислениями на различных графических ускорителях (GPU), но, хоть это и является достаточно эффективным решением задачи, если рассматривать в перспективе перенос приложения на узел кластера ВолгГТУ, такие ускорители доступны на нем в небольшом количестве.

OpenMP – технология написания параллельных программ для систем с общей памятью, состоящий из набора директив компилятора и библиотечных функций [23]. С OpenMP достаточно легко создавать многопоточные приложения на С/С++, Fortran. Этот способ повышения быстродействия является наиболее широко применимым. Многие технологии параллельных вычислений позволяют добиться одновременной обработки данных путем минимального изменения кода.

Таким образом, оптимальный вариант – использование многопоточной реализации алгоритма шинглов на языке С++.

Для сравнения двух текстов на этапах разбиения канонизированного теста на наборы шинглов, расчета контрольных сумм шинглов и сравнения этих контрольных сумм основные действия производятся в циклах. Поэтому здесь можно внедрить параллелизм данных, используя прагму OpenMP – #pragma omp parallel for. Директива #pragma omp for сообщает, что при выполнении цикла for в параллельном регионе итерации цикла должны быть распределены между потоками группы [9].

Тогда цикл разбиения текста на наборы шинглов и получения контрольных сумм будет выглядеть следующим образом:

#pragma omp parallel for

for (int i = 0; i <= (Words.size() - count\_shingle); i++)

{

string ShingleText = "";

for (int j = 0; j < count\_shingle; j++)

ShingleText += (Words[i + j] + " ");

int res = crc.FullCRC((const unsigned char\*)ShingleText.c\_str(), (size\_t)ShingleText.length());

buf[i] = res;

}

Циклы этапа сравнения контрольных сумм:

#pragma omp parallel for

for (int i = 0; i < sizeA; i++)

arrMatches[i] = 0;

#pragma omp parallel for

for (int i=0; i < sizeA ; i++) // string s in ShingleA)

{

for (int j = 0; j < sizeB; j++)

if ( bufB[j] == bufA[i] ) {

arrMatches[i]++;

break;

}

}

Также в одном из циклов метода сравнения контрольных сумм целесообразно использование прагмы #pragma omp parallel for reduction, которая производит выполнение операций редукции, таких как вычисление глобальной суммы [25]:

matches = 0;

#pragma omp parallel for reduction(+:matches)

for (int i = 0; i < sizeA; i++)

matches += arrMatches[i];

Также для повышения скорости сравнения двух наборов хэш-сумм целесообразно использовать векторизацию. Пример функции сравнения двух массивов хэшей с использованием функций-интринсиков для регистров AVX:

int CompareShingles(int\* bufA, int\* bufB, int sizeA, int sizeB)

{

int \*arrMatches;

arrMatches = (int\*) \_mm\_malloc(sizeA \* sizeof(int), 64);

int matches = 0;

#pragma omp parallel for

for (int i = 0; i < sizeA; i++)

arrMatches[i] = 0;

#pragma omp parallel for

for (int i = 0; i < sizeA; i++) {

int A[16] \_\_attribute\_\_((aligned(64)));

for (int j = 0; j < 16; j++) { A[j] = bufA[i]; }

\_\_m512i first = \_mm512\_load\_epi32(A);

for(int j = 0; j < sizeB; j += 16) {

\_\_m512i second = \_mm512\_load\_epi32(&bufB[j]);

\_\_mmask16 found = \_mm512\_cmpeq\_epi32\_mask(first, second);

if( found ) { arrMatches[i]++; break; }

}

}

matches = 0;

#pragma omp parallel for reduction(+:matches)

for (int i = 0; i < sizeA; i++)

matches += arrMatches[i];

return 2\*100\*matches/(sizeA + sizeB);

}

Данное решение поддерживается как платформами KNL c системой команд AVX-512, так и сопроцессорами Intel(R) Xeon Phi(TM) с системой команд KNC.

Наряду с параллелизмом данных также можно использовать параллелизм задач – параллельное вычисление контрольных сумм для обоих сравниваемых текстов с помощью прагмы #pragma parallel section. В таком случае каждая секция выполняется в отдельном потоке, что позволяет производить декомпозицию по коду:

#pragma omp parallel sections{

#pragma omp section

{

sizeA = GetShingles(Canonize(TextA), ShingleA);

}

#pragma omp section

{

sizeB = GetShingles(Canonize(TextB), ShingleB);

}}

Как отмечалось выше, сравнение документа с базой текстов происходит в цикле, поэтому этот процесс также можно распараллелить при помощи прагмы #pragma omp parallel for. Таким образом, сравнение каждой пары файлов: исходного и файла из базы будет осуществляться отдельным процессом.

3.1.2 API для взаимодействия с клиентами

Сначала необходимо отметить, что для работы с текстовым документом был реализован класс TextFile. Этот класс имеет следующий список полей:

/// id файла

public int Id { get => mId; }

private int mId;

/// Имя файла

public string FileName { get => mFileName; }

private string mFileName;

/// Название файла

public string Name { get => mName; }

private string mName;

/// Описание файла

public string Description { get => mDescription; }

private string mDescription;

/// Дата добавления файла

public DateTime Date { get => mDate; }

private DateTime mDate;

/// Данные

public byte[] Content { get => mContent; }

private byte[] mContent;

/// Текст

public string Text { get => mText; }

private string mText;

/// Набор хэшей шинглов, рассчитанных для файла

public int[] Shingles { get => mShingles; }

public string ShinglesString{

get => JsonConvert.SerializeObject(mShingles);

}

private int[] mShingles;

/// Размер файла в байтах

public long Size { get => mSize; }

private long mSize;

/// Путь к файлу

public string Path { get => mPath; }

private string mPath;

/// Тип файла

public string ContentType { get => mContentType; }

private string mContentType;

/// Каталог с файлами на сервере

public string CatalogFilesDataBase { get => Startup.PathFiles; }

Как уже было сказано, веб-API приложение использует REST-архитектуру.

Для работы с файлами реализован контроллер FilesController. Запросы к данному контроллеру будут выглядеть следующим образом [14]:

1. GET api/file/{id} – получение текстового файла из базы по идентификатору.
2. POST api/file – добавление файла, переданного в запросе в базу, данные в теле запроса.
3. PUT api/file – изменение информации о файле, данные в теле запроса.
4. DELETE api/file/id – удалить файл из базы.

Пример метода, принимающего GET-запрос к API для получения файла по идентификатору:

[HttpGet]

[Route("[controller]/{id:int}")]

public JsonResult Get(int id)

{

TextFile textFile = null;

try

{

textFile = new TextFile(id, 1);

}

catch (Exception ex)

{

return Json(new { error = ex.Message });

}

return Json(textFile);

}

Атрибут [HttpGet] устанавливает, что данный метод принимает только GET запросы. Атрибут [Route("[controller]/{id:int}")] устанавливает маршрут, по которому будет доступен текущий метод.

Здесь инициализируется объект класса TextFile. Используемый конструктор принимает на вход идентификатор файла и целочисленное значение mode, в соответствии с которым из базы выбирается необходимый набор полей документа:

/// Получение файла по id.

/// mode = 1: Id, FileName, Path + Content

/// mode = 2: Id, FileName, Name, Description, Date, Size.

/// mode = 3: Id, FileName, Shingles.

public TextFile(int id, int mode)

{

TextFile dbFile = null;

using (var dbProvider = new DbProvider(Startup.ConnectionString))

dbFile = dbProvider.Get(id, mode);

this.mId = dbFile.Id;

this.mFileName = dbFile.FileName;

this.mName = dbFile.Name;

this.mDescription = dbFile.Description;

this.mDate = dbFile.Date;

this.mShingles = dbFile.Shingles;

this.mPath = dbFile.Path;

this.mSize = dbFile.Size;

if (mode == 1)

this.mContent = FileWorker.ReadFile(this.Path);

}

Класс DbProvider реализует методы работы с базой данных, об этом будет подробно рассказана далее. Метод Get данного класса извлекает из базы соответствующий выбранному mode набор полей, а также, если mode = 1, то есть режим получения файла для скачивания (вместе с контентом), происходит чтение текстового файла из директории на сервере по пути Path.

Получив текстовый файл, метод Get контроллера FilesController возвращает его в виде сериализованного объекта json. При возникновении ошибки при получении файла возвращается строка ошибки, также в виде json строки.

Для добавления файла в базу используется следующий метод:

[HttpPost]

[Route("[controller]")]

public JsonResult Post(IFormFile file)

{

TextFile textFile = null;

try

{

// Сохранение полей из IFormFile

textFile = new TextFile(file);

// Сохранение файла в базу

int id = textFile.Save();

textFile = new TextFile(id, 2);

}

catch (Exception ex)

{

return Json(new { error = ex.Message });

}

return Json(textFile);

}

Также, как и метод, принимающий GET запрос, данный метод имеет атрибут, соответствующий типу принимающего запроса. Здесь на вход приходит объект с типом IFormFile, которым представляются все загружаемые файлы в ASP.NET Core [27]. C помощью методов IFormFile можно произвести различные манипуляции файлом – получить его свойства, сохранить, получить его поток и т.д. Некоторые его свойства и методы:

* ContentType: тип файла;
* FileName: название файла;
* Length: размер файла;
* CopyTo/CopyToAsync: копирует файл в поток;
* OpenReadStream: открывает поток файла для чтения.

Объект IFormFile подается в конструктор при инициализации объекта текстового файла:

public TextFile(IFormFile file)

{

if (file == null)

throw new Exception("Пустой файл");

this.mFileName = file.FileName;

this.mContent = FileWorker.GetFileBytes(file);

this.mText = GetFileText(file);

this.mShingles = GetShingles(mText);

this.mContentType = file.ContentType;

this.mSize = file.Length;

this.mDate = DateTime.Now;

}

Здесь, если файл не равен null, происходит инициализация полей объекта. Метод FileWorker.GetFileBytes производит чтение контента файла при помощи класса System.IO.BinaryReader:

try

{

reader = new BinaryReader(file.OpenReadStream());

bytes = reader.ReadBytes((int)file.Length);

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

finally

{

reader.Close();

}

return bytes;

Метод GetFileText производит считывание текста из файла. Следует отметит, что приложение работает не только с txt файлами, но и с docx документами. Однако, для чтения текста из docx файла требуется использование библиотеки DocumentFormat.OpenXml. Если файл имеет тип «text/plain», текст читается как обычный текстовый файл:

bytes = FileWorker.GetFileBytes(file);

text = System.Text.Encoding.UTF8.GetString(bytes);

А если файл имеет тип «application/vnd.openxmlformats-officedocument.wordprocessingml.document», текст из него читается следующим образом:

using (Stream stream = file.OpenReadStream())

{

using (WordprocessingDocument doc = WordprocessingDocument.Open(file.OpenReadStream(), false))

{

bodyDoc = doc.MainDocumentPart.Document.Body;

text = bodyDoc.InnerText;

}

}

Расчет массива хэшей шинглов для текста производится методом GetShingles, который, в свою очередь, вызывает соответствующую функцию динамической библиотеки.

Непосредственно сохранение текстового фала происходит при вызове метода Save. Алгоритм сохранения включает следующую последовательность действий:

1. Сохранение файла в БД вызовом метода DbProvider.Add, который возвращает идентификатор только что добавленного документа;
2. Получение будущего пути к файлу на сервере путем конкатенации строк – указанная директория для хранения оригинала файла (указывается в файле конфигурации appsettings.json) на сервере, идентификатора файла, имя файла с расширением: String.Format("{0}/{1}\_{2}", CatalogFilesDataBase, mId, mFileName);
3. Обновление директории хранения фала в соответствующей строке БД – DbProvider.Update;
4. Сохранение документа в директории на сервере при помощи класса System.IO.FileStream.

Методы Put и Delete контроллера FilesController реализованы аналогичным образом.

Для сравнения текстов реализован контроллер CompareController, в котором находятся два метода: CompareTwoTexts и CompareTextWithBase. Данные методы используют класс LibRepository, который является своеобразной оберткой для динамической библиотеки с функциями вычислительной части алгоритма шинглов. Здесь объявлен набор внешних статических методов, эквивалентных функциям библиотеки:

[DllImport(LibSource)]

private static extern int CmpTwoTexts(string text1, string text2);

[DllImport(LibSource)]

private static extern string CmpTextWithBase(string text);

[DllImport(LibSource)]

private static extern void GetShinglesArray(string text, int[] shingles, int sizeText);

[DllImport(LibSource)]

private static extern int CmpTwoShingles(int[] shingles1, int[] shingles2, int size1, int size2);

Как видно, все они имеют атрибут DllImport. LibSource – приватное статическое константное поле, хранящее путь к динамической библиотеке.

Таким образом, для сравнения двух текстов вызывается метод CmpTwoTexts, для сравнения двух наборов контрольных сумм – метод CmpTwoShingles.

3.1.3 Реализация взаимодействия приложения с базой данных

Первую очередь, необходимо создать базу данных MySQL назовем ее «TextCompareDB». Для этого выполним запрос в MySQL Server:

create database TextCompareDB;

Далее необходимо создать в базе данных таблицу для хранения информации о текстовых документах и рассчитанный набор шинглов. Для этого необходимо выполнить запрос:

create table TextFile(

Id INT UNSIGNED NOT NULL AUTO\_INCREMENT,

Name NVARCHAR(128),

FileName NVARCHAR(128) NOT NULL,

Description NVARCHAR(256),

Date DATETIME,

Shingles LONGTEXT,

Size BIGINT,

Path NVARCHAR(128),

PRIMARY KEY(Id) )

ENGINE=MyISAM DEFAULT CHARSET=utf8;

Теперь можно добавлять файлы в БД, но для этого необходимо реализовать соответствующие методы в веб-API приложении. Все методы взаимодействия с базой будут реализованы в классе DbProvider.

DbProvider наследуется от интерфейса IDisposable. Интерфейс IDisposable объявляет один единственный метод Dispose, в котором при реализации интерфейса в классе должно происходить освобождение неуправляемых ресурсов. Данный метод будет использоваться для отключения от БД и высвобождении используемых ресурсов.

Реализуем конструктор класса DbProvider:

public DbProvider(string strConn)

{

if (string.IsNullOrWhiteSpace(strConn))

throw new Exception("Не передана строка подключения");

ConnectionString = strConn;

Connect();

}

Конструктор получает на вход строку подключения к БД. Далее полученная строка проходит проверку, не передана ли пустая строка, после чего сохраняется во внутреннюю закрытую переменную класса. Если была передана пустая строка, вызывается исключение, символизирующее об ошибке. Приватный метод Connect производит подключение к базе:

private void Connect()

{

dbConnection = new MySqlConnection(ConnectionString);

dbConnection.Open();

SetUtf8();

}

Также необходимо реализовать метод Dispose, в котором будет происходить отключение от базы и высвобождение ресурсов: [28]

public void Dispose()

{ Disconnect();

}

Соответственно, метод Disconnect():

private void Disconnect()

{

if (Connection != null)

{

Connection.Close();

Connection.Dispose();

}

}

Схема запроса к БД будет выглядеть, как показано на рисунке 14.

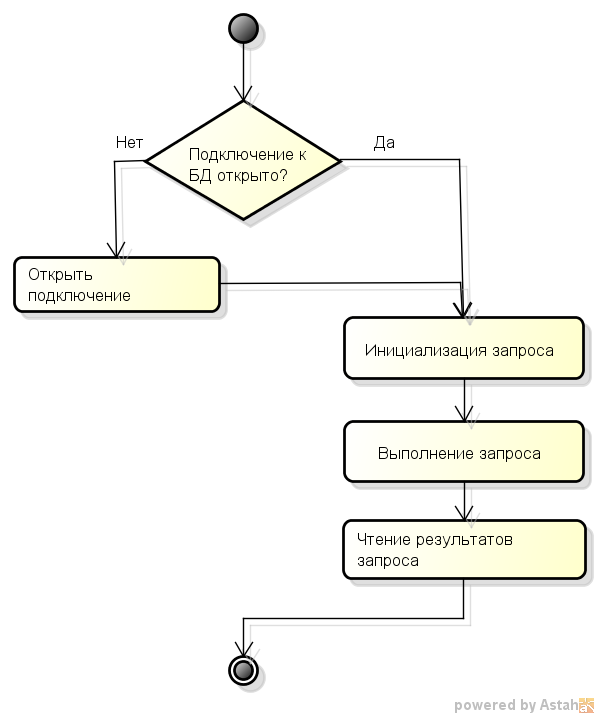


Рисунок 14 – Схема выполнения запроса к БД

Метод проверки состояния подключения выглядит следующим образом:

private void CheckConnect()

{

if (Connection != null)

if (Connection.State == ConnectionState.Open)

return;

Connect();

}

Здесь проверяется состояние подключения, после чего, если подключение не открыто, оно открывается, иначе происходит выход из функции.

Каждая последовательность запросов (или один запрос) будут заключены в оператор using, например:

using(var dbProvider = new DbProvider(connectionString))

{

var file = dbProvider.Get(id);

}

Оператор using гарантирует вызов метода Dispose, в котором в данном случае происходит отключение от БД, даже если при вызове методов в объекте происходит исключение.

Запрос получения списка файлов базы для сравнения с текстом:

public List<TextFile> GetShingles()

{

CheckConnect();

IDbCommand dbCommand = null;

IDataReader reader = null;

List<TextFile> files = new List<TextFile>();

try

{

dbCommand = Connection.CreateCommand();

dbCommand.CommandText = "SELECT Id, FileName, Shingles FROM TextFile;";

reader = dbCommand.ExecuteReader();

while (reader.Read())

{

files.Add(new TextFile

(

Convert.ToInt32(reader["Id"]),

Convert.ToString(reader["FileName"]).Trim(),

null,

null,

DateTime.MinValue,

null,

Convert.ToString(reader["Shingles"]).Trim(),

null,

0

));

}

}

catch (Exception ex)

{

throw ex;

}

finally

{

dbCommand.Dispose();

dbCommand = null;

}

return files;

}

Первым делом проходит проверка подключения, согласно схеме (рисунок 14). Далее инициализируется запрос:

SELECT Id, FileName, Shingles FROM TextFile;

Для сравнения нет необходимости поднимать весь объект файла из базы, достаточно только получить идентификатор, название файла и сериализованный массив хэшей шинглов, рассчитанный для текста.

Далее результат выполнения запроса сохраняется в объект IDataReader, после чего происходит заполнение списка файлов, который подается на выход метода. Следует отметить, что в любом случае, выполнится ли запрос корректно, либо возникнет исключение, ресурсы, занятые командой освободятся в операторе finally.

3.2 Реализация клиентского веб-приложения MVC

3.2.1 Внешний вид веб-приложения

Страница в таком виде, в котором мы ее видим в окне браузера собирается из мастер-страницы, страницы представления и нескольких частичных представлений.

Мастер-страница наывается \_Layout.cshtml и содержит тэги <html>, <head>, <body> и пр. Также тут находится html-разметка шапки и подвала сайта, подключаются некоторые скрипты. О ест те элементы, которые используются на каждой веб-странице приложения. Метод @RenderBody() является плэйсхолдером,  на место которого другие представления, которые используют эту мастер-страницу, будут подставлять свое содержимое [29].

Страница сравнения текстов имеет следующую разметку:

@{ ViewData["Title"] = "About"; }

<div class="row">

<h1>Сравнение текстов</h1>

<div class="tab-container">

<ul class="nav nav-tabs">

<li class="active navbar-right">

<a data-toggle="tab" href="#cmp-two">Сравнение двух текстов</a>

</li>

<li class="navbar-right">

<a data-toggle="tab" href="#cmp-base">Сравнение текста с базой</a>

</li>

</ul>

<div class="tab-content content-box">

<div class="tab-pane active" id="cmp-two">

@Html.Partial("\_CmpTwoTexts")

<div id="loader-cmp-two-text" class="loader">

</div>

</div>

</div>

</div>

</div>

@section Scripts{

<script src="/js/compare.js"></script>

}

@Html.Partial("\_CmpTwoTexts") является тэг-хэлпером, который рендерит и подставляет в данный участок разметки частичное представление с именем «\_CmpTwoTexts» [29].

Структура представления «Сравнение текстов» содержит заголовок и две вкладки: «Сравнение двух текстов» и «Сравнение текста с базой» (рисунок 15). Во вкладке «Сравнение двух текстов» находятся два поля ввода, кнопка «Сравнить два текста», а также кнопки «Из файла» над каждым текстовым файлом. То есть текст для сравнения можно, как и скопировать или ввести в поле, так и загрузить из документа на диске.

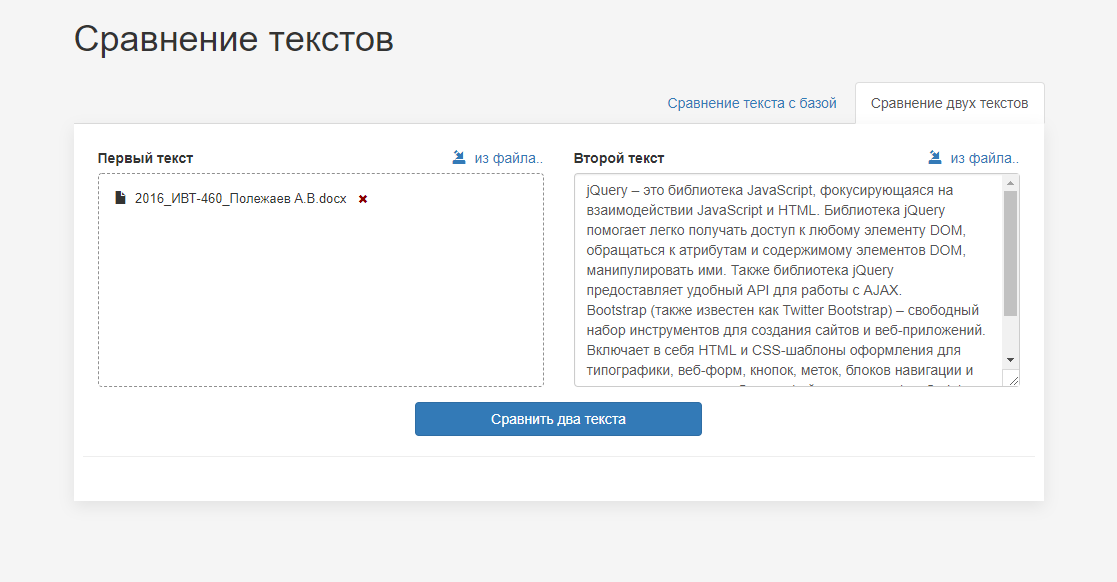


Рисунок 15 – Внешний вид представления сравнения текстов

К кнопке «Сравнить два текста» привязано событие click, которое происходит при клике на этот элемент:

$('#compare-two-texts-btn').click(function (e) {

e.preventDefault();

var data = getDataToCmpTwoTexts();

if (data == null)

return;

$loader.show();

$.ajax({

url: "/Compare/CompareTwoTexts",

type: 'POST',

data: data,

enctype: 'multipart/form-data',

processData: false,

contentType: false,

cache: false,

async: true,

error: isError,

success: isSuccess,

failure: isError

});

return false;

function isSuccess(response) {

$loader.hide();

var data = JSON.parse(response);

if (data.error) { // Ошибка

alert("Ошибка: " + data.error);

return;

} else if (data.success) {

$('#cmp-two-result-field').show();

$('#cmp-two-result').text(data.success + "%");

alert('Время сравнения: ' + data.timeSpan / 1000 + ' сек.');

} else {

alert("Error: Empty response");

return;

}

}

function isError(response) {

$loader.hide();

alert("Status: " + response.status + " - " + response.statusText);

}

});

Здесь происходит получение выбранного файлов, если они были выбраны, или текста, которые помещаются в объект formData, после чего он отправляется ajax запросом в метод действия по адресу «/Compare/CompareTwoTexts». При успешном выполнении запроса вызывается функция isSuccess, в которой в объект json парсится строка ответа и результат помещается в соответствующий блок в представлении. При получении ошибки вызывается всплывающее окно с описанием.

Аналогичным способом осуществляется работа с вкладкой «Сравнить текст с базой». Ответ при этом возвращается не json-объектом, а частичным представлением (рисунок 16), которое состоит из заголовков и таблицы:

<table class="table">

<thead>

<tr>

<th>Название файла</th>

<th>Процент совпадений</th>

<th>Скачать</th>

</tr>

</thead>

<tbody>

@foreach(CompareResult item in Model)

{

<tr class="">

<td>@item.TextDocument.FileName</td>

<td class="percent @filterCategory" data-percent="@item.Percent">@item.Percent%</td>

<td>

<a href="/File/Download/@item.TextDocument.Id">Скачать</a>

</td>

</tr>

}

</tbody>

</table>

Метод @foreach позволяет перебрать все значения списка модели и сформировать таблицу [29].

Также в результатах сравнения появляются 4 кнопки, позволяющие фильтровать результаты по проценту совпадений текстов. Фильтрация происходит в функциях jQuery, путем скрывания и показа необходимых строк таблицы.

Страница «Работа с файлами» также состоит из блока с двумя вкладками: «Загрузка файлов» и «Поиск файлов в базе».

На вкладке с загрузкой файлов находится блок загрузки файлов, такой же, как на страннице сравнения текстов, только здесь можно загружать до 100 файлов одновременно.

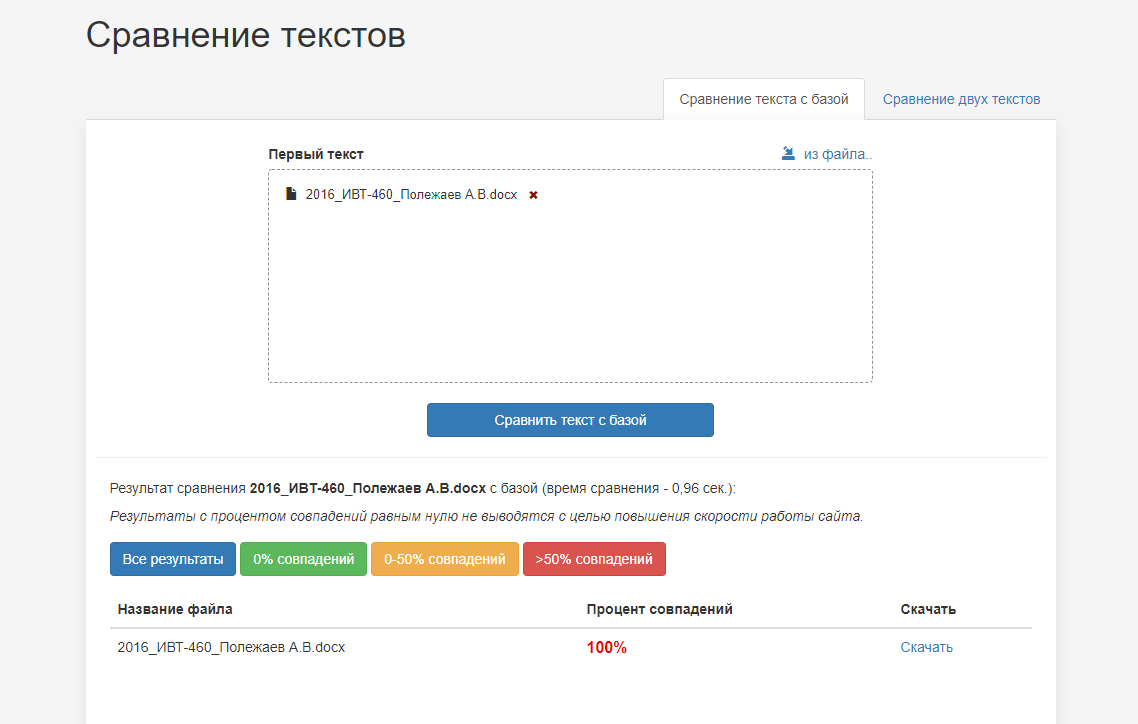


Рисунок 16 – Сравнение текста с базой

Также реализована поддержка Drag&Drop (рисунок 17):

var filesArea = $(".files-area");

filesArea.on('drag dragstart dragend dragover dragenter dragleave drop', function (e) {

e.preventDefault();

e.stopPropagation();

})

.on('dragover dragenter', function () {

filesArea.addClass('is-dragover');

})

.on('dragleave dragend drop', function () {

filesArea.removeClass('is-dragover');

})

.on('drop', function (e) {

var files = e.originalEvent.dataTransfer.files;

addFiles(files, filesArea);

});

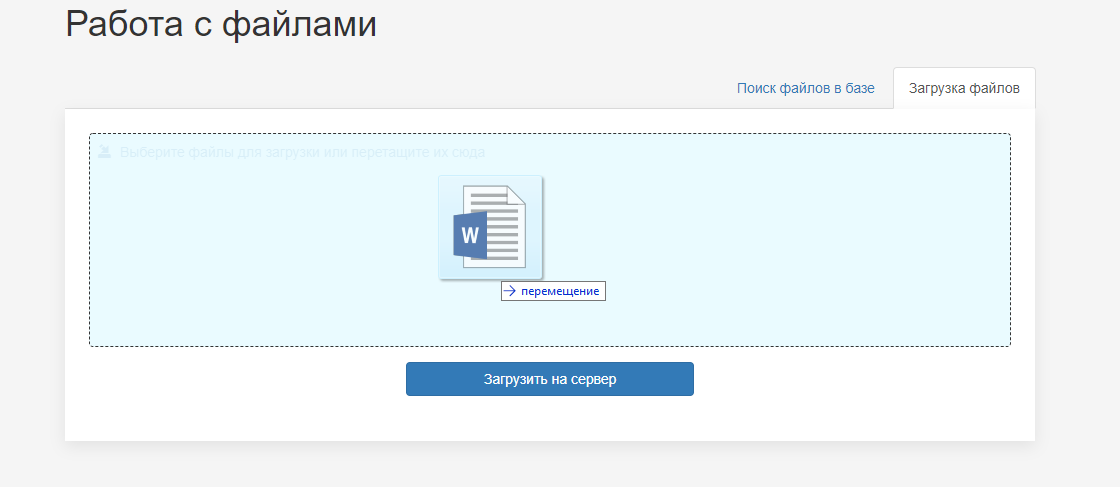


Рисунок 17 – Drag&Drop добавлении файлов на загрузку

Drag&Drop реализован путем привязки и обработки событий: drag, dragstart, dragend, dragover, dragenter, dragleave, drop к соответствующему html элементу представления.

После начала загрузки файлов на сервер, поверх вкладки появляется лоадер с секундомером (рисунок 18), с помощью которого можно следить за прошедшим с начала загрузки временем и убедиться, что веб-приложение не подвисло. Кстати, такой же лоадер предусмотрен при сравнении текстов.

Вкладка «Поиск файлов в базе» состоит из строки поиска и таблицы с результатами, которая генерируется по аналогии с таблицей результатов сравнения текстов с базой. Внешний вид вкладки продемонстирован на рисунке 19.

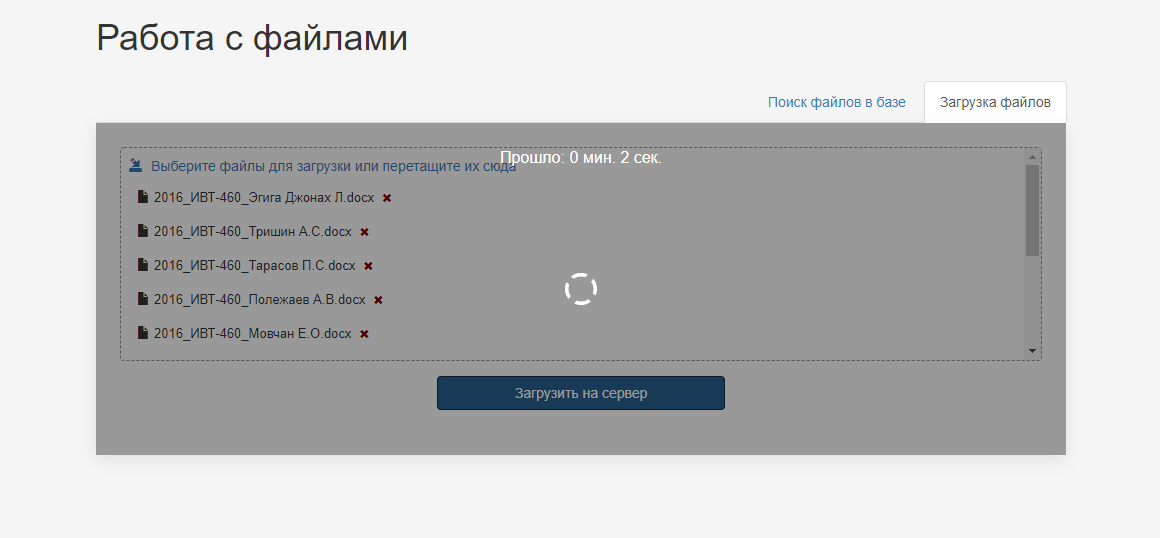


Рисунок 18 – Лоадер с секундомером во время загрузки файлов

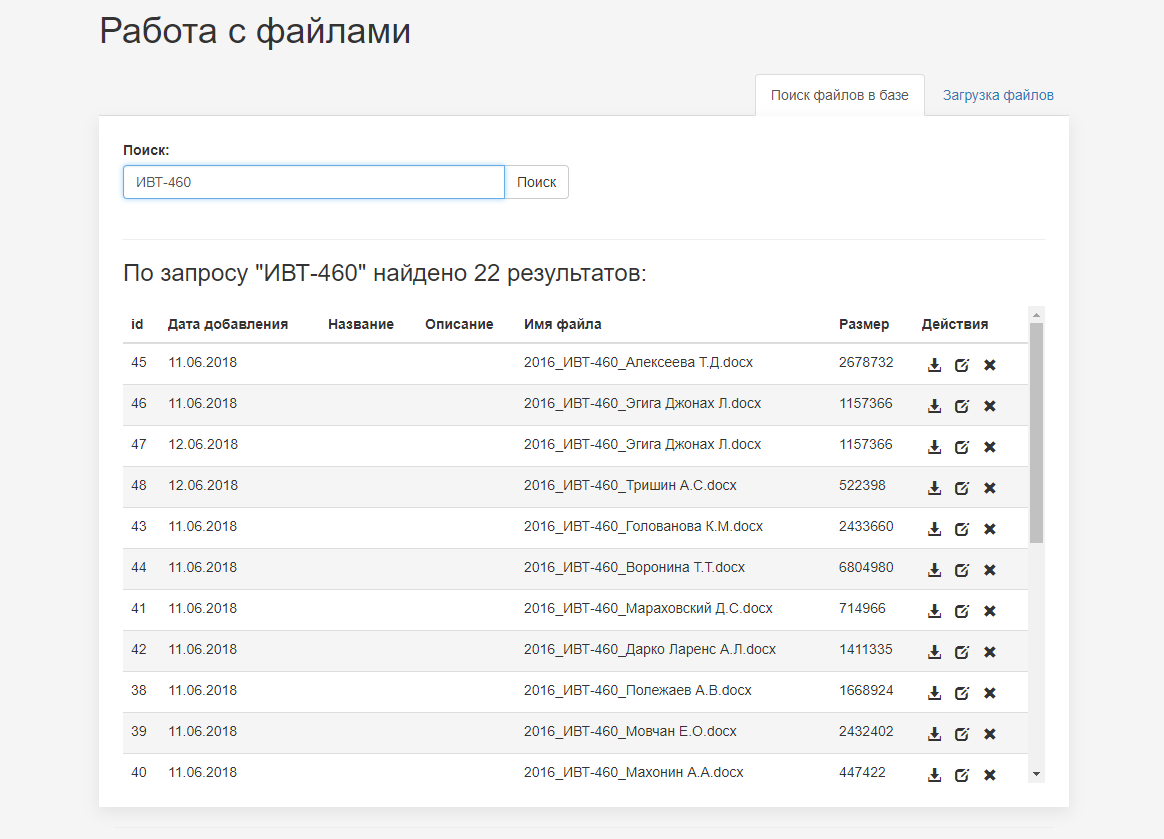


Рисунок 19 – Поиск файлов в базе

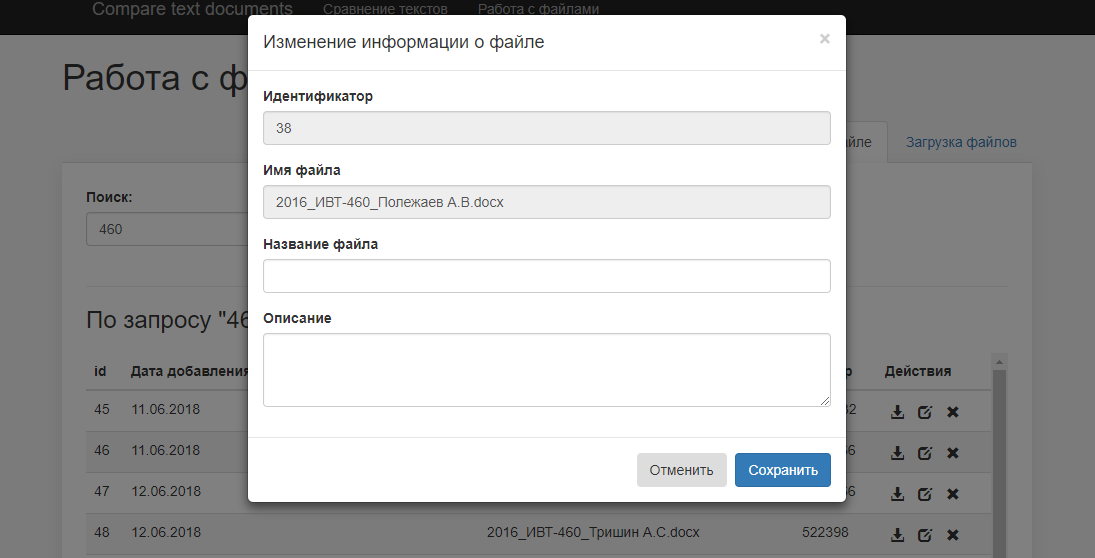


Рисунок 20 – Модальное окно изменения информации о файле

В колонке «Действия» расположены три иконки, которые позволяют скачать, изменить или удалить файл. При клике на «Скачать» или «Удалить» файл, отправляется запрос в метод действия контроллера, после чего пользователю показывается всплывающее окно, сообщающее об успешном выполнении команды, либо выводи описание ошибки. При клике на иконку «Изменить» появляется модальное окно с доступными полями для изменения, внешний вид модального окна показан на рисунке 20.

Модальное окно реализована при помощи CSS фреймворка Bootstrap и вызывается путем вызова функции modal('show') [19].

3.2.2 Система авторизации

Для работы с базовой системой аутентификации весь основной функционал сосредоточен в Nuget-пакете Microsoft.AspNetCore.Authentication.Cookies, который и позволит работать с куками. В проектах ASP.NET Core версии 2.0 и выше этот пакет уже имеется по умолчанию [30].

Для установки аутентификации с помощью куки в методе ConfigureServices в вызов services.AddAuthentication передается значение CookieAuthenticationDefaults.AuthenticationScheme. Далее с помощью метода AddCookie() настраивается аутентификация. По сути в этом методе производится конфигурация объекта CookieAuthenticationOptions, который описывает параметры аутентификации с помощью кук. В частности, в данном случае использовано одно свойство CookieAuthenticationOptions - LoginPath. Это свойство устанавливает относительный путь, по которому будет перенаправляться анонимный пользователь при доступе к ресурсам, для которых нужна аутентификация [30].

И чтобы аутентификация была встроена в конвейер обработки запроса, в методе Configure вызывается компонент middleware app.UseAuthentication().

Методы аутентификации и авторизации реализованы в контроллере AccountController. Метод Login, принимающий GET-запрос возвращает представление с полями ввода логина и пароля, откуда они отправляются пост запросом в POST метод Login:

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Login(string login, string password)

{

if (login == "admin" && password == "1234")

{

await Authenticate(login); // аутентификация

return RedirectToAction("Index", "Compare");

}

return Json(new{error = "Неверный логин или пароль"});

}

Можно было бы хранить логины и пароли в базе данных, реализовать регистрацию, группы пользователей с разными ролями и правами доступа, к примеру, админ, преподаватель, студент, но на данном этапе разработки в этом нет необходимости. Поэтому, логин и пароль, полученные из представления, сравниваются со значениями «admin» и «1234» и при совпадении происходит аутентификация вызовом приватного метода Authenticate:

[HttpPost]

private async Task Authenticate(string userName)

{

// Создаем один claim

var claims = new List<Claim>

{

new Claim(ClaimsIdentity.DefaultNameClaimType, userName)

};

// Создаем объект ClaimsIdentity

ClaimsIdentity id = new ClaimsIdentity(claims, "ApplicationCookie", ClaimsIdentity.DefaultNameClaimType, ClaimsIdentity.DefaultRoleClaimType);

// Установка аутентификационных куки

await HttpContext.SignInAsync(CookieAuthenticationDefaults.AuthenticationScheme, new ClaimsPrincipal(id));

}

При помощи вызова метода Logout происходит выход пользователя с аккаунта сайта и перенаправление на страницу авторизации:

public async Task<IActionResult> Logout()

{

await HttpContext.SignOutAsync(

CookieAuthenticationDefaults.AuthenticationScheme);

return RedirectToAction("Login", "Account");

}

3.2.3 Взаимодействие с API сравнения текстов

Веб-приложение взаимодействует с веб-API путем отправки GET, POST, PUT и DELETE запросов, получая некоторый запрос из страницы представления.

Для сравнения двух текстов либо текстов с базой реализован контроллер CompareController, в котором имеются методы, принимающие POST-запросы с текстом (текстами) и/или файлами (так как могут сравниваться как тексты, введенные в соответствующее поле на странице, так и текстовые документы): CompareTwoTexts и CompareTextWithBase. Если передан текстовый документ, то перед сравнением из него извлекается текст тем же способом, как и в веб-API приложении, который был описан в предыдущих пунктах.

Так, для сравнения двух текстов, после валидации происходит формирования объекта parameters, который представляет собой массив пар ключ-значение:

// Валидация

if (string.IsNullOrEmpty(text1) || string.IsNullOrEmpty(text2))

throw new Exception("Не введен текст для сравнения.");

var parameters = new[]

{

new KeyValuePair<string, string>("text1", text1),

new KeyValuePair<string, string>("text2", text2)

};

Затем данный объект отправляется POST запросом в Web API приложение:

string response = Request.POST (CmpTwoTextsUrl, parameters);

Поле CmpTwoTextsUrl хранит путь к методу сравнения текстов API.

Аналогичным способом происходит и работа с файлами. Метод контроллера получает на вход файл, либо набор некоторых полей, которые отправляются POST, GET, PUT или DELETE запросом к API в зависимости от операции. В ответ приходит строка с сериализованным объектом, или ошибкой, которая возвращается клиенту после десериализации.

4 ТЕСТИРОВАНИЕ РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ

Для тестирования подхода с использованием ускорителей Intel(R) Xeon Phi(TM) 1 поколения (KNС) была создана база из наборов контрольных сумм, количество которых равно 5000 в каждом файле базы. Сравнивался с базой документ, также состоящий из 5000 слов. В таблице 3 представлены результаты оценки схожести текста с базой документов на сопроцессоре Intel(R) Xeon Phi(TM) первого поколения.

Таблица 3 – Время оценки схожести документа с базой на Intel(R) Xeon Phi(TM) 1 поколения (KNС)

|  |  |  |  |
| --- | --- | --- | --- |
| Количество файлов в базе | Однопоточные вычисления | Многопоточные вычисления  (228 потоков) | Ускорение |
| 200 | 7.826 | 0.086 | 91.0 |
| 500 | 19.570 | 0.281 | 69.6 |
| 1000 | 39.114 | 0.652 | 59.9 |

Было получено ускорение более чем в 55 раз для базы из одной тысячи файлов при использовании распараллеливания и работы с AVX регистрами.

В таблице 4 представлены результаты тестирования на CPU Intel(R) Xeon Phi(TM) 2 поколения (KNL).

Таблица 4 – Время оценки схожести документа с базой на Intel(R) Xeon Phi(TM) 2 поколения (KNL)

|  |  |  |  |
| --- | --- | --- | --- |
| Количество файлов в базе | Однопоточные вычисления | Многопоточные вычисления  (256 потоков) | Ускорение |
| 200 | 5.762 | 0.121 | 47.6 |
| 500 | 13.734 | 0.316 | 43.5 |
| 1000 | 25.879 | 0.463 | 55.9 |

Было получено ускорение более чем в 55 раз для базы из одной тысячи файлов при использовании распараллеливания и AVX регистров.

В таблице 5 представлены замеры времени сравнения текста с базой на CPU Intel(R) Xeon(R) E5.

Таблица 5 – Время оценки схожести документа с базой на CPU Intel(R) Xeon(R) E5

|  |  |  |  |
| --- | --- | --- | --- |
| Количество файлов в базе | Однопоточные вычисления | Многопоточные вычисления  (20 потоков) | Ускорение |
| 200 | 0.317 | 0.083 | 3.8 |
| 500 | 0.738 | 0.112 | 6.6 |
| 1000 | 1.484 | 0.162 | 9.2 |

Было получено ускорение более чем в 9 раз для базы из одной тысячи файлов.

Итоговые результаты тестирования на разных процессорах представлены в таблице 6.

Таблица 6 – Время оценки схожести документа с базой на разных процессорах

|  |  |  |  |
| --- | --- | --- | --- |
| Количество файлов в базе | Xeon(R) E5  (20 потоков) | Xeon Phi(TM)  1 поколения (KNС)  (228 потоков) | Xeon Phi(TM)  2 поколения (KNL)  (256 потоков) |
| 200 | 0.083 | 0.086 | 0.101 |
| 500 | 0.112 | 0.281 | 0.216 |
| 1000 | 0.162 | 0.652 | 0.463 |

Вычисления на центральном процессоре Intel(R) Xeon(R) E5 выполняется быстрее приблизительно в 3 раза по сравнению с Xeon Phi(TM) 2 поколения (KNL) и Xeon Phi(TM) 1 поколения (KNC). Если сравнивать KNL и KNС, то расчеты на первом выполняются в 1.5 раза быстрее, однако, на вычислительном кластере в перспективе возможно использование 6 платформ с сопроцессорами 1 поколения параллельно. Это может увеличить производительность пропорционально количеству ускорителей, то есть в 6 раз, соответственно, в 3 раза быстрее, чем на CPU Xeon(R) E5.

Однако, многочисленные опыты показали, что KNC технически сложен, сложна реализация работы с базой текстов – подход требует распределение базы по сопроцессорам и сохранение ее в памяти при каждом запуске приложения на сервере. Также стает вопрос в сложности реализации взаимодействия сервиса на CPU с вычислительной программой на сопроцессоре, получение результата по окончанию сравнения с базой.

Было выявлено, что значительная часть времени сравнения текста с базой документов занимает считывание наборов контрольных сумм из файлов базы данных. Также необходимо выяснить, какой вид базы документов использовать: хранить наборы контрольных сумм шинглов в фалах или в базе данных MySQL.

На основании этого производилось тестирование и ускорение алгоритмов работы с базой на устройствах: Intel(R) Xeon(R) CPU E5 c 20 ядрами (20 потоков) и Intel(R) Xeon Phi(TM) CPU (KNL) с 64 ядрами (256 потоков).

Было создана база, состоящая из файлов по 5000 слов каждый после чего, в первом случае они прошли разработанную процедуру сохранения в базу данных MySQL. Во втором случае, после получения файлов веб-API приложением, для них так же рассчитывались наборы хэшей, которые затем записывались в текстовый файл и сохранялись в указанной директории на сервере.

При считывании контрольных сумм из базы MySQL был распараллелен цикл их десериализации:

Parallel.ForEach(serializeShaingles, shingles =>

{

files.Add(JsonConvert.DeserializeObject<int[]>(shingles));

});

Для варианта считывания хранения наборов хэшей в текстовых файлах было произведено распараллеливание цикла чтения файлов:

Parallel.ForEach(dirs, dir =>

{

List<int> array = new List<int>();

// Чтение файла

using (var reader = new StreamReader(dir))

{

while (!reader.EndOfStream)

array.Add(Convert.ToInt32(reader.ReadLine()));

}

files.Add(array.ToArray());

});

Результаты замеров времени чтения массива файлов из данных вариантов базы отражены в таблице 7.

Таблица 7 – Время чтения контрольных сумм из базы на Intel(R) Xeon(R) CPU E5

|  |  |  |  |
| --- | --- | --- | --- |
| Количество файлов | Последовательно, с | Параллельно  (20 потоков) , с | Ускорение |
| Хранение контрольных сумм в текстовых документах | | | |
| 200 | 0.424 | 0.043 | 9.86 |
| 500 | 1.078 | 0.076 | 14.18 |
| 1000 | 2.166 | 0.165 | 13.13 |
| Хранение контрольных сумм в БД MySQL | | | |
| 200 | 0.513 | 0.101 | 5.08 |
| 500 | 1.195 | 0.242 | 4.94 |
| 1000 | 2.553 | 0.316 | 8.08 |

Как видно по результатам в таблице, было получено ускорение более чем в 14 раз для 20 потоков при хранении хэшей в файлах, а также было получено ускорение более чем в 8 раз для 20 потоков для способа хранения хэшей в базе данных MySQL.

В таблице 8 показан результат сравнения времени считывания набора контрольных сумм из базы при их хранении в файлах или в БД MySQL.

Как видно из таблицы с результатами, сравнение текстов с базой производится приблизительно на 50% быстрее при чтении наборов контрольных сумм из текстовых файлов.

Таблица 8 – Сравнение способов чтения (параллельно) из разных вариантов базы на Intel(R) Xeon(R) CPU E5

|  |  |  |  |
| --- | --- | --- | --- |
| Количество файлов | Время чтения хэшей, с | | Отношение |
| Текстовые файлы | БД MySQL |
| 200 | 0.043 | 0.101 | 0.43 |
| 500 | 0.076 | 0.242 | 0.31 |
| 1000 | 0.165 | 0.316 | 0.52 |

Таким образом, вариант хранения контрольных сумм в файлах является более оптимальным для CPU Intel(R) Xeon(R) E5.

Ниже представлены результаты тестирования аналогичных вариантов хранения хэшей для веб-API приложения на Intel(R) Xeon Phi(TM) CPU (KNL) с 64 ядрами (256 потоков).

Таблица 9 – Время чтения контрольных сумм из базы на Intel(R) Xeon Phi(TM) CPU (KNL)

|  |  |  |  |
| --- | --- | --- | --- |
| Количество файлов | Последовательно, с | Параллельно  (256 потоков), с | Ускорение |
| Хранение хэшей в текстовых документах | | | |
| 200 | 1.329 | 0.957 | 1.39 |
| 500 | 3.353 | 2.365 | 1.42 |
| 1000 | 7.107 | 5.132 | 1.39 |
| Хранение хэшей в БД MySQL | | | |
| 200 | 2.292 | 1.928 | 1.19 |
| 500 | 4.908 | 3.602 | 1.36 |
| 1000 | 8.973 | 7.273 | 1.23 |

По результатам было получено ускорение примерно в 1.4 раза для 256 потоков для обоих типов базы данных.

В таблице 10 показан результат сравнения времени считывания набора контрольных сумм из базы при их хранении в файлах или в БД MySQL.

Таблица 10 – Сравнение способов чтения (параллельно) из разных вариантов базы на Intel(R) Xeon Phi(TM) CPU (KNL)

|  |  |  |  |
| --- | --- | --- | --- |
| Количество файлов | Время чтения хэшей, с | | Отношение |
| Текстовые файлы | БД MySQL |
| 200 | 0.957 | 1.928 | 0.50 |
| 500 | 2.365 | 3.602 | 0.65 |
| 1000 | 5.132 | 7.273 | 0.71 |

Таким образом, вариант хранения наборов контрольных сумм в файлах является более эффективным.

В итоге было выявлено, что считывание наборов контрольных сумм из базы данных на Intel(R) Xeon(R) CPU E5 происходит быстрее, чем на Intel(R) Xeon Phi(TM) CPU (KNL).

Следует отметить то, что итоговое время сравнения текстов базы от момента клика на кнопку «Сравнить» до вывода результата будет больше результатов, показанных выше, за счет того, что присутствуют такие этапы, как передача файла на сервер, расчет набора контрольных сумм для него, а также передача итогового результата обратно клиенту. Скорость этих операций зависит от скорости интернет соединения между клиентом и сервером, а также производительности устройства клиента.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были решены следующие задачи:

* определено понятие плагиата, рассмотрен алгоритм шинглов для нахождения схожести текстов;
* проанализированы популярные системы проверки на плагиат;
* спроектировано веб-приложение для сравнения текстовых документов с использованием актуальных технологий разработки, выделены функциональные особенности и структура приложения;
* после проектирования было реализовано веб-приложение, состоящее из ASP.NET Core веб-приложения и ASP.NET Core веб-API сервиса;
* было произведено ускорение работы веб-сервиса путем разработки и использования многопоточной версии алгоритмов с применением технологий параллельных вычислений и работы с AVX регистрами;
* веб-сервис был протестирован на различных платформах, имеющихся в составе вычислительного кластера ВолгГТУ.

Плагиат – умышленное присвоение авторства чужого произведения искусства или достижения науки, технических решений или изобретений.

Шингл в переводе с английского – это ячейка, чешуйка, частичка, кирпичик. В сфере анализа текстов шингл означает те составные части, из которых строится текст. Алгоритм шинглов – это алгоритм для поиска копий текста и выявления плагиата в тексте. Он включает в себя следующие этапы:

1) канонизация текста;

2) разбиение текста на наборы шинглов;

3) расчет контрольных сумм для шинглов;

4) сравнение массивов контрольных сумм текстов;

5) подсчет совпадений и определение результата.

На сегодняшний момент существует множество готовых систем, при помощи которых студент может бесплатно или за невысокую стоимость проверять свои работы на плагиат. Но использование таких систем для проверки большого числа работ в рамках учебного заведения не всегда удобно и оптимально. Большинство ресурсов не обладает достаточным качеством проверки или быстродействием и многие из них затратные в плане вложений.

Разработанное приложение имеет следующие функциональные возможности: сравнение двух текстов между собой, сравнение текста с базой документов, загрузка, скачивание и удаление документов базы, получение списка всех документов, содержащихся в базе на данный момент, получение и изменение информации о каком-либо конкретном файле, которая указывается при загрузке документа на сервер (название файла, описание, размер и др.).

В качестве фреймворка для создания приложения был выбран .NET Core 2.0, который позволяет создавать приложения с поддержкой разных платформ (Windows, Linux и macOS).

Структура приложения включает в себя две части: клиентскую и серверную. Разделение было сделано для того, чтобы отделить вычислительную часть, требующую затрат вычислительных ресурсов от клиентской, которую при данной структуре можно реализовать как web приложение, desktop приложение или приложение для Android. Таким образом, к серверу могут подключаться несколько клиентов с различных устройств. В качестве клиентской части было реализовано ASP .NET Core Web приложение, в качестве серверной – ASP .NET Core Web API приложение.

Веб приложение ASP .NET Core использует паттерн MVC. Для создания внешнего вида страниц представлений будут использоваться: HTML5, CSS3, библиотека jQuery, CSS фреймворк Bootstrap 3. Web API приложение использует REST архитектуру. Соответственно, клиентская и серверная часть взаимодействуют, посылая друг другу HTTP-запросы. Рассчитанные наборы контрольных сумм текстового документа, а также основная информация о документе хранится в таблице базы данных MySQL. В то время как сам загруженный файл загружается в директорию на сервере.

Было выявлено три возможных структуры развертывания веб-сервиса на кластере ВолгГТУ, подразумевающие использование сопроцессоров Xeon Phi(TM) первого поколения(KNC), Xeon Phi(TM) 2 поколения (KNL), Intel(R) Xeon(R) E5. Для тестирования был реализован прототип программы сравнения текста с базой документов с использованием технологии параллельных вычислений OpenMP, а также функций-интринсиков для работы с регистрами AVX.

В результате тестирования сравнения текста с базой текстовых документов показали, что вычисления на центральном процессоре Intel(R) Xeon(R) E5 выполняется быстрее приблизительно в 3 раза по сравнению с Xeon Phi(TM) 2 поколения (KNL) и Xeon Phi(TM) 1 поколения (KNC). Если сравнивать KNL и KNС, то расчеты на первом выполняются в 1.5 раза быстрее, однако, на вычислительном кластере в перспективе возможно использование 6 платформ с сопроцессорами 1 поколения параллельно. Это может увеличить производительность пропорционально количеству ускорителей, то есть в 6 раз, соответственно, в 3 раза быстрее, чем на CPU Xeon(R) E5.

Однако, многочисленные опыты показали, что KNC технически сложен, сложна реализация работы с базой текстов – подход требует распределение базы по сопроцессорам и сохранение ее в памяти при каждом запуске приложения на сервере. Также стает вопрос в сложности реализации взаимодействия сервиса на CPU с вычислительной программой на сопроцессоре, получение результата по окончанию сравнения с базой.

Произвелось тестирование времени считывания наборов контрольных сумм из двух вариантов базы: хранение в текстовых файлах и в базе данных MySQL, которое показало незначительное преимущество первого варианта.

Алгоритм чтения наборов контрольных сумм из БД MySQL был ускорен в 8 раз путем добавления распараллеливания десериализации массива хэшей.

Соответственно, можно сделать вывод, что целесообразнее для развертывания веб-приложения использовать платформу с CPU Xeon(R) E5.

Альтернативным вариантом является использование нескольких платформ с CPU Xeon Phi(TM) 2 поколения (KNL) с применением векторизованного алгоритма шинглов использующего AVX-512. Данные платформы имеются в составе вычислительного кластера ВолгГТУ и при этом менее нагружены, чем узел CPU Xeon(R) E5.

Новизна работы заключается в усовершенствование алгоритма шинглов для платформ KNL за счет использования систем команд AVX-512; ускорение работы веб-сервиса за счет распараллеливания процесса десериализации наборов контрольных сумм.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Об утверждении порядка проведения государственной итоговой аттестации по образовательным программам высшего образования – программам бакалавриата, программам специалитета и программам магистратуры : приказ Мин-ва обр. и науки РФ от 29.06.2015 г. № 636. – 2015.

2 Плагиат : Академик [Электронный ресурс]. – Режим доступа : http://dic.academic.ru/dic.nsf/fin\_enc/16386 (Дата обращения 10.02.2018).

3 Алгоритм шинглов : Wikipedia – The Free Encyclopedia [Электронный ресурс]. – Режим доступа : https://ru.wikipedia.org/wiki/Алгоритм\_шинглов (Дата обращения 20.01.2018).

4 Алгоритм Шинглов - поиск нечетких дубликатов текста : Copy Watcher [Электронный ресурс]. – Режим доступа : http://copywatcher.ru/copyright\_statyi/dlya-programmistov-i-vebmasterov/python-algoritm-shinglov-poisk-nechetkih-dublikatov-teksta/ (Дата обращения 20.04.2018).

5 Антиплагиат [Электронный ресурс]. – Режим доступа : http://www.antiplagiat.ru/ (Дата обращения 12.02.2018).

6 Advego Plagiatus - проверка уникальности текста : Advego – биржа контента [Электронный ресурс]. – Режим доступа : http://advego.ru/plagiatus/ (Дата обращения 13.02.2018).

7 Проверка уникальности текста : eTXT.ru [Электронный ресурс]. – Режим доступа : https://www.etxt.ru/antiplagiat/ (Дата обращения 14.02.2018).

8 Автоматический контроль уникальности — защита контента ваших сайтов : Content Watch [Электронный ресурс]. – Режим доступа : https://content-watch.ru/ (Дата обращения 14.02.2018).

9 Серов, С. С. Сокращение времени оценки схожести текстовых документов на неоднородной многопроцессорной вычислительной системе / С. С. Серов, А. В. Андреев, П. Д. Кравченя, Р. И. Гущин, П. П. Чеботарев // Инженерный вестник Дона. – 2015. – №2, ч. 2. – С. 98-107.

10 How to Run an ASP.NET Core Website in Production on Ubuntu Linux : Developing software. Mastering Software Craftsmanship [Электронный ресурс]. – Режим доступа : https://developingsoftware.com/aspnetcore-ubuntu/ (Дата обращения 10.02.2018).

11 ASP.NET Core : The APT.NET Site https://www.asp.net/core/overview/aspnet-vnext (Дата обращения 20.02.2018).

12 Введение в ASP.NET Core : Microsoft Docs [Электронный ресурс]. – Режим доступа : https://docs.microsoft.com/ru-ru/ASPNET/core/?view=aspnetcore-2.1 (Дата обращения 20.02.2018).

13 Выбор между .NET Core и .NET Framework для серверных приложений : Microsoft Docs [Электронный ресурс]. – Режим доступа : https://docs.microsoft.com/ru-ru/dotnet/standard/choosing-core-framework-server (Дата обращения 20.02.2018).

14 REST API - что это? Representational State Transfer : Fb.ru [Электронный ресурс]. – Режим доступа : http://fb.ru/article/338879/rest-api---chto-eto-rest-perevod-representational-state-transfer (Дата обращения 7.03.2018).

15 Маршалинг в C#. Простые типы : Microsin.net [Электронный ресурс]. – Режим доступа : http://microsin.net/programming/pc/marshalling-with-csharp-simple-types.html (Дата обращения 2.02.2018).

16 Platform Invoke Tutorial : Microsoft Developer Network [Электронный ресурс]. – Режим доступа : https://msdn.microsoft.com/en-us/library/aa288468(v=vs.71).aspx (Дата обращения 10.05.2016).

17 ASP.NET MVC : Microsoft Docs [Электронный ресурс]. – Режим доступа : https://docs.microsoft.com/ru-ru/ASPNET/core/?view=aspnetcore-2.1 (Дата обращения 20.02.2018).

18 jQuery : jquery.com [Электронный ресурс]. – Режим доступа : https://jquery.com/ (Дата обращения 15.03.2018).

19 Bootstrap – The world's most popular mobile-first and responsive front-end framework : Bootstrap [Электронный ресурс]. – Режим доступа : https://getbootstrap.com/docs/3.3/ (Дата обращения 15.03.2018).

20 Why MySQL : MySQL [Электронный ресурс]. – Режим доступа : https://www.mysql.com/why-mysql/ (Дата обращения 10.02.2018).

21 Ross, W. A painless guide to CRC error detection algorithms [Электронный ресурс] / W. Ross. – 1993. – Режим доступа : http://web.archive.org/web/20130407000813/http://rsdn.ru/article/files/classes/SelfCheck/crcguide.pdf (Дата обращения 12.05.2016).

22 Cyclic redundancy check : Wikipedia – The Free Encyclopedia [Электронный ресурс]. – Режим доступа : https://ru.wikipedia.org/wiki/Циклический\_избыточный\_код (Дата обращения 10.02.2016).

23 Карпов, А. Технология OpenMP [Электронный ресурс] / А. Карпов. – 2009. – Режим доступа : http://www.viva64.com/ru/a/0057/ (Дата обращения 25.05.2016).

24 Знакомство с уровнями распараллеливания : Блог компании Intel [Электронный ресурс]. – Режим доступа : https://habrahabr.ru/company/intel/blog/80342/ (Дата обращения 28.05.2016).

25 Кудухов, А. Н. Анализ методов и технологий параллельных вычислений для обработки информации // APRIORI. Серия: Естественные и технические науки / А. Н. Кудухов. - 2014. - №1. – с. 28-46.

26 Intel® Xeon Phi™ Coprocessor System Software Developers Guide [Электронный ресурс]. – Режим доступа : http://software.intel.com/en-us/articles/intel-Xeon-Phi-coprocessor-system-software-developers-guide (Дата обращения 3.01.2018).

27 File uploads in ASP.NET Core : Microsoft Docs [Электронный ресурс]. – Режим доступа : https://docs.microsoft.com/en-us/aspnet/core/mvc/models/file-uploads?view=aspnetcore-2.1 (Дата обращения 12.03.2018).

28 Реализация метода Dispose : Microsoft Docs [Электронный ресурс]. – Режим доступа : https://docs.microsoft.com/ru-ru/dotnet/standard/garbage-collection/implementing-dispose

29 ASP.NET Core. Введение в Razor Pages : Metanit.com [Электронный ресурс]. – Режим доступа : https://metanit.com/sharp/aspnet5/29.1.php (Дата обращения 15.04.2018).

30 Introduction to Identity on ASP.NET Core : Microsoft Docs [Электронный ресурс]. – Режим доступа : https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-2.0&tabs=visual-studio%2Caspnetcore2x (Дата обращения 2.05.2018).

31 Знакомство с уровнями распараллеливания : Блог компании Intel [Электронный ресурс]. – Режим доступа : https://habrahabr.ru/company/intel/blog/80342/ (Дата обращения 5.03.2018).