Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка базы данных для хранения и обработки данных для приложения учета и аудита времени, потраченного на рабочие и личные задачи»

Студент ИУ7-66Б (Группа) _____ (Подпись, дата) М. Ю. Вольняга (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) К. Л. Тассов (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире, где время является ценным ресурсом, эффективное управление им является важной задачей как для отдельных людей, так и для организаций. Чтобы грамотно распоряжаться своим временем, необходимо знать, сколько времени реально требуется на выполнение каждой задачи. Именно поэтому разработка приложения для учета и аудита времени, потраченного на определенную задачу, является актуальной [1].

Приложение, которое объединит в себе удобство канбан-досок для визуального планирования задач с интегрированным таймером для учета времени и аналитики, станет уникальным решением и поможет эффективно планировать как личные, так и рабочие задачи. Оно позволит пользователям совместно вести учет времени, что сделает его незаменимым инструментом для широкого круга пользователей, от фрилансеров до компаний, стремящихся к оптимизации своих процессов управления временем и повышению общей продуктивности.

Цель курсовой работы — разработка базы данных для хранения и обработки данных приложения учета и аудита времени, потраченного на рабочие и личные задачи.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- сформулировать описание пользователей проектируемого приложения по учету и аудиту времени, потраченного на рабочие и личные задачи для доступа к базе данных;
- спроектировать сущности базы данных и ограничения целостности учета и аудита времени, потраченного на рабочие и личные задачи;
- выбрать средства реализации и разработать базу данных и приложение;
- провести исследование зависимости времени выполнения запросов от количества записей в базе данных.

1 Аналитический раздел

В данном разделе будет рассмотрено следующее: анализ предметной области, формулировка требований к разрабатываемой базе данных и приложению, формализация и описание информации, анализ существующих баз данных на основе формализации данных, ER-диаграмма, формализация и описание пользователей проектируемого приложения к базе данных, диаграмма вариантов использования.

1.1 Анализ предметной области

Предметная область «эффективное управление временем и задачами» затрагивает процессы планирования, отслеживания и анализа времени, затрачиваемого на выполнение различных задач и проектов.

Существует множество теорий и методологий, способствующих эффективному управлению временем. Среди наиболее известных подходов выделяются Матрица Эйзенхауэра, которая помогает приоритизировать задачи по их срочности и важности, метод Помодоро, предлагающий разбивку работы на короткие интервалы с перерывами, и система GTD, нацеленная на снижение стресса за счет переноса задач из головы в внешнюю систему организации. Современные методологии планирования, такие как Agile, Scrum и Kanban, привносят гибкость и адаптивность в процесс управления проектами и задачами. Они позволяют командам реагировать на изменения и эффективно управлять рабочим процессом, поддерживая постоянную обратную связь и цикличность в деятельности [2; 3].

Анализ аналогичных решений

На рынке существует большое количество приложений, которые имеют функциональность замера времени для задач, наиболее известные: clockify, toggl, timecamp, также есть приложения по планирования задач в стиле канбан-досок, например как trello. Выделим следующие критерии для сравнения:

- наличие таймера для задачи;
- ограниченность в создание проектов;
- наличие аналитики и экспорт ее;
- ограниченность в количестве пользователей у проекта;

– наличие канбан-досок для удобной организации задач.

В таблице 1.1 представлено сравнение существующих решений для бесплатных версий приложений.

Таблица 1.1 – Сравнение существующих решений

Критерий	**clockify**	**toggl**	**timecamp**	**trello**
Таймер	Есть	Есть	Есть	Нет
Количество проектов	Неограниченно	Неограниченно	Ограничено	Ограничено
Экспорт аналитики	PDF, CSV, Excel	PDF, CSV	PDF	Нет
Кол-во пользователей	Неограниченно	Ограничено	Ограничено	Ограничено
Канбан-доски	Нет	Нет	Нет	Есть

Как видно из таблицы 1.1 ни одно из рассмотренных решений не удовлетворяет всем критериям сравнения, также данные приложения являются зарубежными. Таким образом мое решение будет актуальным и в отличие от других решений, будет иметь канбан-доски с интегрированным таймером для каждой задачи, облегчая тем самым учет времени и повышая эффективность планирования

1.2 Формулировка требований к разрабатываемой базе данных и приложению

В рамках курсовой работы необходимо разработать базу данных, в которой будет храниться информация о пользователях, их проектах, задачах, карточках, тегах, списках дел и группах. Также были выделены необходимые требования для приложения. Приложение должно включать следующие функциональности:

– регистрация и аутентификация пользователя;

– изменение списка дел, задач проектов и тегов;

– поиск задач, проектов и тегов;

– замер времени для конкретной задачи;

– просмотр и сохранение аналитики использования времени, замеренного для определенных задач;

– создание и изменение групп пользователей.

Необходимо предусмотреть работу с задачами в виде канбан-досок, так как данная функция является отличительной от других приложений.

1.3 Формализация и описание информации

Разрабатываемая база данных для приложения по учету и аудиту затраченного времени на определенные задачи, должна содержать информацию о пользователях, их проектах, задачах, карточках, тегах, списках дел и группах.

В таблице 1.2 представлены необходимые таблицы и информация, которая должна содержаться в этой таблице для разрабатываемой базы данных.

Таблица 1.2 – Таблиц и их информаций

Таблица	**Информация**
Пользователь	id, имя, фамилия, роль, почта, дата последнего входа и дата регистрации
Проект	id, название, описание
Задача	id, название, статус, описание, дата начала и завершения задачи, сумма времени таймера,
Группа	id, название, описание
Тег	id, название, цвет
Список дел	id, статус, приоритет, дата завершения, контент

На рисунке 1.1 приведена ER диаграмма в нотации Чена.

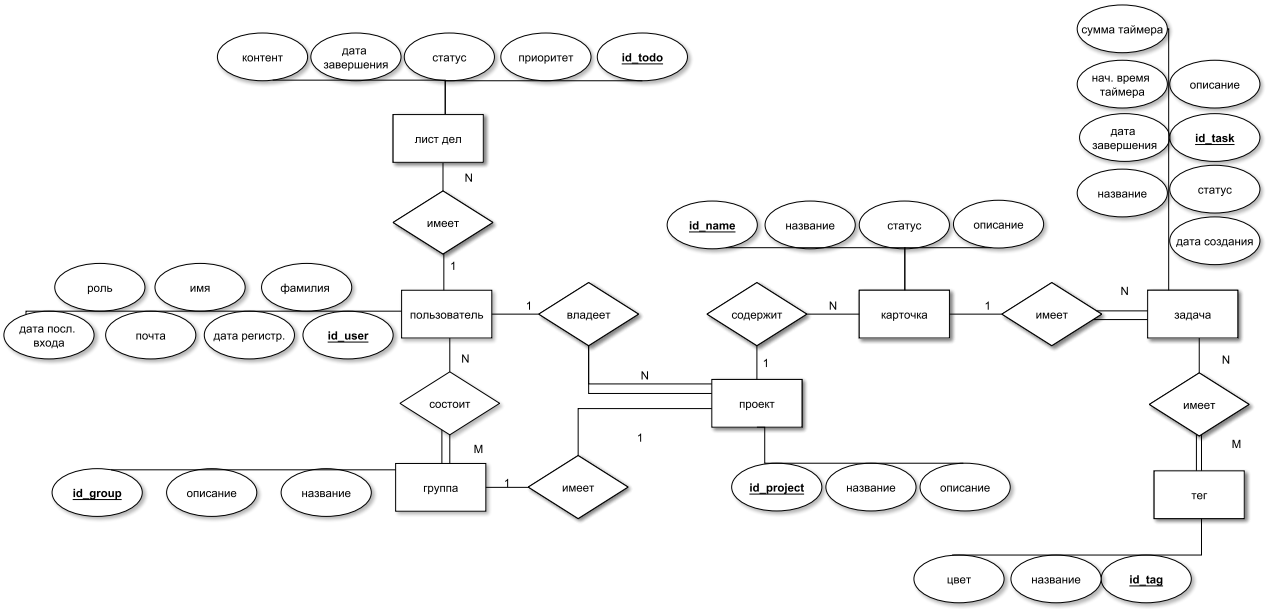

Рисунок 1.1 – ER диаграмма

1.4 Формализация и описание пользователей проектируемого приложения к базе данных

Для взаимодействия с приложением по учету и аудиту времени, было выделено три роли пользователей: неавторизированный, авторизированный и администратор.

В таблице 1.3 представлена функциональность для администратора, неавторизированного и авторизированного пользователя. Также в таблице 1.3 используются сокращения: 1 означает пользователь администратор, 2 — неавторизированный пользователь, 3 — авторизированный пользователь.

Таблица 1.3 – Функциональность администратора, неавторизированного и авторизированного пользователя

Функциональность	**1**	**2**	**3**
Зарегистрироваться и войти в систему	+	+	+
Изменить список дел		+	+
Создать и изменить личный проект, задачи и теги		+	+
Провести поиск задач, проектов и тегов		+	+
Замерить время для конкретной задачи		+	+
Просмотреть и сохранить аналитику		+	+
Создать группу			+
Изменить группу (если владелец)			+
Просмотреть и сохранить аналитику (группы)	+		+
Изменить группы пользователей	+		
Удалить пользователя из системы	+		

1.5 Диаграмма вариантов использования

На рисунке 1.2 приведена диаграмма вариантов использования.

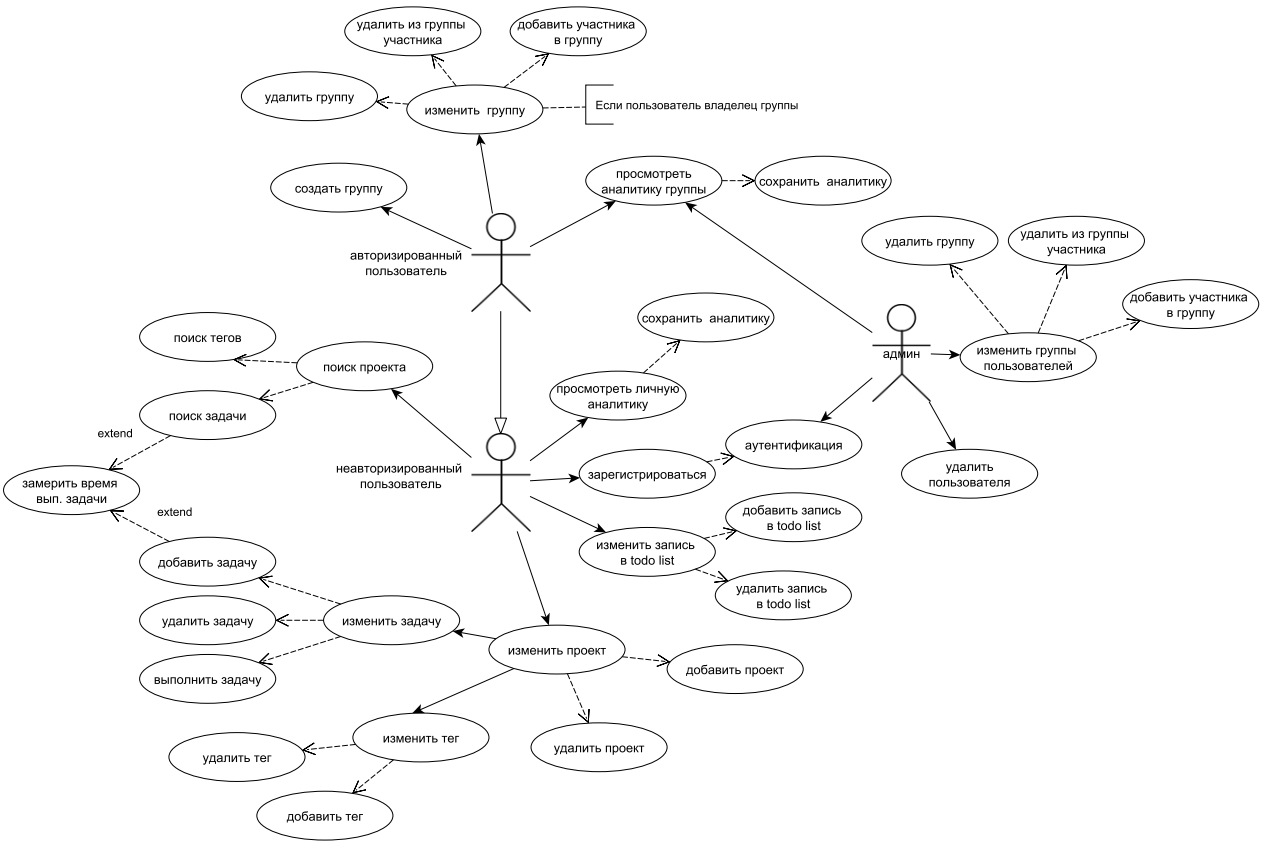

Рисунок 1.2 – Диаграмма использования

1.6 Анализ существующих баз данных на основе формализации данных

В контексте темы курсовой работы и проведенной формализации задачи, данных и пользователей, была выбрана реляционная модель данных.

Реляционная модель данных представляет собой подход к управлению данными, где все данные хранятся в таблицах (также называемых «отношениями»), состоящих из строк и столбцов. Каждая строка таблицы представляет собой запись с уникальным идентификатором (ключом), а каждый столбец — атрибутом данных. Реляционные базы данных обеспечивают гибкость в обработке данных, возможность выполнения сложных запросов и обеспечения целостности данных через систему отношений между таблицами.

SQLite

SQLite — это встраиваемая, высоконадежная библиотека СУБД, которая предоставляет реляционную базу данных в виде файла. Ее главное преимущество — не требует отдельного серверного процесса или системы управления, что делает SQLite идеальным выбором для мобильных приложений, приложений для настольных компьютеров и небольших веб-проектов, где требуется простота без жертвы функциональности. Она поддерживает основные операции SQL и отличается малым размером, быстродействием и надежностью.

PostgreSQL

PostgreSQL — это мощная, открытая и полностью бесплатная объектно-реляционная система управления базами данных (ОРСУБД). Она предоставляет расширенные возможности SQL, включая сложные запросы, внешние ключи, триггеры, представления, транзакции, индексирование полнотекстового поиска и сохраненные процедуры. PostgreSQL является хорошим выбором для крупных приложений и систем, требующих высокой надежности, масштабируемости и гибкости.

MySQL

MySQL — это популярная открытая реляционная система управления базами данных. Она широко используется в веб-разработке и является частью стека LAMP (Linux, Apache, MySQL, PHP/Python/Perl). MySQL предлагает высокую производительность, надежность и простоту в использовании, а также поддерживает широкий спектр операционных систем. Эта СУБД

идеально подходит для веб-сайтов и приложений среднего и большого размера, требующих эффективного управления данными.

Эти СУБД выбраны из-за их популярности, мощности, гибкости и разнообразия функций, которые они предлагают для разработчиков мобильных приложений, веб-приложений и крупных систем.

Таблица 1.4 – Сравнение реляционных СУБД

Критерий	**SQLite**	**PostgreSQL**	**MySQL**
Лицензия	Public Domain [**sqlite**]	PostgreSQL License, открытая [**postgresql**]	GPL для бесплатной версии или коммерческая лицензия [**mysql**]
Тип хранения	Встраиваемая	Клиент-сервер	Клиент-сервер
Поддержка JSON	Частичная	Полная	Полная
Репликация	Поддержка через сторонние инструменты	Нативная поддержка	Нативная поддержка
Полнотекстовый поиск	Да	Да	Да

Источники:

SQLite: Официальный сайт SQLite.

PostgreSQL: Официальный сайт PostgreSQL.

MySQL: Официальный сайт MySQL.

SQLite был выбран как целевая СУБД для разработки мобильного приложения на Kotlin по нескольким ключевым причинам, учитывая специфику задачи, данные и пользователей:

Встраиваемость и Легковесность: SQLite — это встраиваемая база данных, что означает, что она легко интегрируется непосредственно в мобильное приложение без необходимости во внешнем сервере баз данных. Это делает SQLite особенно подходящим для мобильных приложений, где необходимо минимизировать использование ресурсов и обеспечить высокую производительность.

Независимость и Портативность: SQLite хранит всю базу данных в

одном файле, что облегчает передачу, резервное копирование и синхронизацию данных между устройствами. Это важно для мобильных приложений, которые могут работать в автономном режиме или в условиях ненадежного сетевого соединения.

Простота Управления: SQLite не требует настройки, установки и управления сервером баз данных, что снижает затраты на поддержку и управление приложением. Это делает его идеальным выбором для малых и средних проектов, где может не хватать ресурсов для обслуживания сложной инфраструктуры баз данных.

Поддержка Транзакций и ACID: SQLite полностью поддерживает транзакции и соответствует принципам ACID (атомарность, согласованность, изоляция, долговечность), что гарантирует надежность и целостность данных в приложении.

Широкая Поддержка и Сообщество: SQLite имеет обширную документацию и поддерживается большим сообществом разработчиков. Это обеспечивает доступность ресурсов, библиотек и инструментов для разработки, тестирования и отладки приложений.

Гибкость: Несмотря на свою простоту, SQLite предлагает достаточный набор функций SQL, включая поддержку подзапросов, внешних ключей, триггеров и представлений, что позволяет эффективно управлять данными в мобильном приложении.

В совокупности, эти преимущества делают SQLite оптимальным выбором для мобильных приложений, где требуется надежное, эффективное и удобное в обслуживании решение для хранения данных.

Вывод

В данном разделе было рассмотрено следующее анализ предметной области, формулировка требований к разрабатываемой базе данных и приложению, формализация и описание информации, анализ существующих баз данных на основе формализации данных, ER-диаграмма, формализация и описание пользователей проектируемого приложения к базе данных, диаграмма вариантов использования.

2 Конструкторский раздел

В данном разделе будут представлены требования к программному обеспечению, описание используемых типов данных и схемы реализуемых алгоритмов.

2.1 Требования к программному обеспечению

К программе предъявлен ряд требований:

– должен присутствовать интерфейс для выбора действий;

– считывание данных должно производиться из файла;

– результат должен записываться в файл;

– должен присутствовать замер реального времени для реализаций алгоритмов;

– результат замера должен выводится в виде таблицы.

2.2 Описание используемых типов данных

При реализации алгоритмов будут использованы следующие структуры и типы данных:

– массив символов для хранения терма;

– вещественное число для хранения *TF-IDF* терма;

– мьютекс — примитив синхронизации.

2.3 Разработка алгоритмов

На рисунке **??** приведена схема дивизимной иерархической кластеризации. На рисунке **??** приведена схема алгоритма k-средних. На рисунке **??** представлена схема алгоритма создания потоков для алгоритма дивизимной иерархической кластеризации. На рисунке **??** представлена схема многопоточного алгоритма кластеризации документов.

Вывод

В данном разделе были представлены требования к программному обеспечению, описание используемых типов данных и схемы реализуемых алгоритмов.

3 Технологический раздел

В данном разделе будут представлены средства реализации, сведения о модулях программы и листинги кода реализации алгоритмов.

3.1 Средства реализации

В качестве языка программирования для разработки программного обеспечения был выбран язык *C++* [**cpp**].

Данный выбор обусловлен следующим:

– язык позволяет реализовать все алгоритмы, выбранные в результате проектирования;

– язык поддерживает все типы и структуры данных, которые были выбраны в результате проектирования;

– язык позволяет работать с нативными потоками [**thread**].

Время выполнения реализаций было замерено с помощью функции *clock* [**clock**]. Для хранения термов использовалась структура данных *string* [**wstring**], в качестве массивов использовалась структура данных *vector* [**vector**]. В качестве примитива синхронизации использовался *mutex* [**mutex**].

Для создания потоков и работы с ними был использован класс *thread* из стандартной библиотеки выбранного языка [**thread**]. В листинге 3.1, приведен пример работы с описанным классом, каждый объект класса представляет собой поток операционной системы, что позволяет нескольким функциям выполняться параллельно [**thread**].

Листинг 3.1 – Пример работы с классом thread

```
#include <iostream>
#include <thread>

void foo(int a){
    std::cout << a << '\n';
}

int main(){
    std::thread thread(foo, 10);
    thread.join();

    return 0;
}
```

3.2 Сведения о модулях программы

Данная программа разбита на следующие модули:

- *main.cpp* — файл, который содержит точку входа в программу;
- *Cluster.cpp* и *Cluster.h* — модуль, который реализует класс *Cluster*;
- *kMeans.cpp* и *kMeans.h* — модуль, содержащий реализацию функции *kMeans*;
- *Klastering.cpp* и *Klastering.h* — модуль, содержащий реализации функций дивизимной иерархической кластеризации: *hierarchicalClustering* и многопоточная версия кластеризации*parallelDocClustering*;
- *io.cpp* и *io.h* — модуль, содержащий реализации функций для работы входными и выходными файлами;
- *Document.cpp* и *Document.h* — модуль, который реализует класс *Document*;
- *Term.cpp* и *Term.h* — модуль, который реализует класс *Term*;

3.3 Реализация алгоритмов

В листинге 3.2 приведена реализация дивизимной иерархической кластеризации. В листинге А.1 приведена реализация алгоритма k-средних. В

листинге 3.3 приведена реализация алгоритма создания потоков для алгоритма дивизимной иерархической кластеризации. В листинге 3.4 приведена реализация многопоточного алгоритма кластеризации документов.

Листинг 3.2 – Реализация дивизимной иерархической кластеризации

```
void hierarchicalClustering(const
   std::vector<std::vector<double>> &docs,
                    int depth,
                    int maxDepth,
                    std::vector<Cluster> &clusters) {
    if (depth == maxDepth) {
        Cluster cluster;
        cluster.docs = docs;
        cluster.calculateCentroid();
        clusters.push_back(cluster);
        return;
    }

    // Применяем k-средних для разделения на 2 кластера
    std::vector<std::vector<double>> centroids = kMeans(docs, 2);
    std::vector<std::vector<double>> cluster1Docs, cluster2Docs;

    for (const auto &doc: docs) {
        if (cosineDistance(doc, centroids[0]) <
           cosineDistance(doc, centroids[1])) {
            cluster1Docs.push_back(doc);
        }
        else {
            cluster2Docs.push_back(doc);
        }
    }

    // Рекурсивно разделяем каждый подкластер
    hierarchicalClustering(cluster1Docs, depth + 1, maxDepth,
       clusters);
    hierarchicalClustering(cluster2Docs, depth + 1, maxDepth,
       clusters);
}
```

Листинг 3.3 – Реализация алгоритма создания потоков

```
void parallelDocClustering(const std::vector<std::string>
    &filenames,
                           const std::vector<std::string>
                              &filenames_out,
                           int num_threads) {
    int count_f = filenames.size();
    std::mutex io_mutex;

    // Расчет размера подгруппы для каждого потока
    int chunk_size = count_f / num_threads;
    std::thread threads[num_threads]; // Массив потоков

    int start_index = 0;
    for (int i = 0; i < num_threads; ++i) {
        int end_index = start_index + chunk_size + (i < count_f
            % num_threads ? 1 : 0);

        // Создание потока для обработки подгруппы документов
        threads[i] = std::thread(processDocuments,
            std::ref(filenames), std::ref(filenames_out),
                                 start_index, end_index,
                                    std::ref(io_mutex));
        start_index = end_index;
    }

    // Дождаться завершения всех потоков
    for (int i = 0; i < num_threads; ++i) {
        threads[i].join();
    }
}
```

Листинг 3.4 – Реализация многопоточного алгоритма кластеризации документов

```
void processDocuments(const std::vector<std::string> &filenames,
                      const std::vector<std::string>
                         &filenames_out,
                      int start_index, int end_index,
                      std::mutex &io_mutex) {
    for (int i = start_index; i < end_index; ++i) {
        // Загрузка и обработка документа
        std::vector<Document> documents =
           readDocumentsFromCSV(filenames[i]);
        standardizeDocumentLengths(documents);

        // Выполнение иерархической кластеризации
        std::vector<Cluster> clusters =
           hierarchicalClustering(documents, MAX_DEPTH);

        // Синхронизированное сохранение результатов
        std::lock_guard<std::mutex> lock(io_mutex);
        saveClustersToJson(clusters, documents,
           filenames_out[i]);
    }
}
```

Вывод

В данном разделе были представлены средства реализации, сведения о модулях программы и листинги кода реализации алгоритмов.

4 Исследовательская часть

В данном разделе будут приведены демонстрация работы программы, технические характеристики устройства, сравнительный анализ времени выполнения реализуемых алгоритмов.

4.1 Демонстрация работы программы

На рисунке **??** представлена демонстрация работы разработанного программного обеспечения.

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее.

- Процессор: Ryzen 5 4600H, 6 процессорных ядер архитектуры Zen 2 и 12 потоков, работающих на базовой частоте в 3.0 ГГц (до 4.0 ГГц в Turbo режиме), 12 логических ядер [**ryzen**]
- Оперативная память: 16 ГБайт.
- Операционная система: Windows 10 Pro 64-разрядная система [**windows**].

При замерах времени ноутбук был включен в сеть электропитания и был нагружен только системными приложениями.

4.3 Время выполнения реализаций алгоритмов

Результаты замеров времени выполнения реализации алгоритма иерархической кластеризации документов в зависимости от числа потоков приведены в таблице Б.1. Каждый замер проводился 100 раз, после чего рассчитывалось их среднее арифметическое значение.

На рисунке **??** изображен график зависимостей времени выполнения реализаций от числа потоков.

Результаты замеров времени выполнения реализации алгоритма иерархической кластеризации документов в зависимости от числа входных файлов приведены в таблице Б.2 В таблице Б.2 в многопоточной реализации был

использован 1 вспомогательный поток, данное число было выбрано для демонстрации уменьшения времени получения результата при использовании многопоточности с минимальным числом вспомогательных потоков. Каждый замер проводился 100 раз, после чего рассчитывалось их среднее арифметическое значение.

В таблице Б.2 используются следующие обозначения:

– «ПР» — время выполнения (в мс) последовательной реализации алгоритма;

– «ПРОДП» — время выполнения (в мс) параллельной реализации алгоритма при одном дополнительном потоке.

На рисунке **??** изображен график зависимости времени выполнения реализации от числа файлов.

Вывод

В результате исследования реализуемых алгоритмов по времени выполнения можно сделать следующие выводы.

– Время выполнения последовательной реализации алгоритма остается постоянным и равным 4200 мс (см. рисунок **??**). На графике видно, что для небольшого числа потоков (до 8), параллельная реализация работает быстрее последовательной. Однако, с увеличением количества потоков, параллельная реализация алгоритма становится менее эффективной по времени выполнения, чем последовательная реализация алгоритма. Это можно объяснить тем, что на машине, на которой проводились исследования 12 логических ядер. При увеличении количества потоков сверх количества логических ядер, происходит конкуренция за процессорные ресурсы, что приводит к переключению контекста и, как следствие, к увеличению времени выполнения. Кроме того, на создание и управление потоками тратится дополнительное время, что также влияет на общую производительность.

– Из таблицы Б.2 видно, что при малом количестве входных файлов разница во времени выполнения между последовательной реализацией и

параллельной реализацией с одним дополнительным потоком, не очень заметна, но с увеличением количества файлов параллельная версия с одним дополнительным потоком становится намного эффективней по времени выполнения. Параллельная реализация алгоритма с одним дополнительным потоком на 38% быстрее по сравнению с последовательной реализацией алгоритма, при обработке 50 файлов (см. таблицу Б.2).

– Исходя из данных (см. таблицу Б.2), оптимальное количество потоков для данной задачи находится в диапазоне от 2 до 8 (см. таблицу 4.1). В этом диапазоне время выполнения минимально и значительно меньше, чем у последовательной реализации алгоритма.

Таблица 4.1 – Относительное улучшение во времени выполнения параллельной реализации алгоритма по сравнению с последовательной реализацией алгоритма

Количество потоков	Относительное улучшение
2	2.6%
3	2.9%
4	3.65%
5	3.21%
6	3.18%
7	2.97%
8	1.8%

ЗАКЛЮЧЕНИЕ

Цель лабораторной работы достигнута, исследованы параллельные вычисления на основе нативных потоков.

Для достижения поставленной цели были выполнены следующие задачи:

- описан алгоритм иерархической кластеризации;
- спроектировано программное обеспечение, реализующее алгоритм и его параллельную версию;
- разработано программное обеспечение, реализующее алгоритм и его параллельную версию;
- проанализированы затраты реализаций алгоритмов по времени.

В результате исследования реализуемых алгоритмов по времени выполнения можно сделать следующие выводы.

- Время выполнения последовательной реализации алгоритма остается постоянным и равным 4200 мс (см. рисунок **??**). На графике видно, что для небольшого числа потоков (до 8), параллельная реализация работает быстрее последовательной. Однако, с увеличением количества потоков, параллельная реализация алгоритма становится менее эффективной по времени выполнения, чем последовательная реализация алгоритма. Это можно объяснить тем, что на машине, на которой проводились исследования 12 логических ядер. При увеличении количества потоков сверх количества логических ядер, происходит конкуренция за процессорные ресурсы, что приводит к переключению контекста и, как следствие, к увеличению времени выполнения. Кроме того, на создание и управление потоками тратится дополнительное время, что также влияет на общую производительность.
- Из таблицы Б.2 видно, что при малом количестве входных файлов разница во времени выполнения между последовательной реализацией и параллельной реализацией с одним дополнительным потоком, не очень заметна, но с увеличением количества файлов параллельная версия с одним дополнительным потоком становится намного эффективней по

времени выполнения. Параллельная реализация алгоритма с одним дополнительным потоком на 38% быстрее по сравнению с последовательной реализацией алгоритма, при обработке 50 файлов (см. таблицу Б.2).

– Исходя из данных (см. таблицу Б.2), оптимальное количество потоков для данной задачи находится в диапазоне от 2 до 8 (см. таблицу 4.1). В этом диапазоне время выполнения минимально и значительно меньше, чем у последовательной реализации алгоритма.

ПРИЛОЖЕНИЕ А

Листинг А.1 – Реализация алгоритма k-средних

```
1  std::vector<std::vector<double>> kMeans(const
       std::vector<std::vector<double>> &docs, int k) {
2      int n = docs.size(); //1
3      std::vector<std::vector<double>> centroids(k,
           std::vector<double>(docs[0].size())); //2
4      std::vector<int> assignments(n, 0); //3
5      // Инициализация центроидов
6      for (int i = 0; i < k; ++i) { // 4
7          centroids[i] = docs[rand() % n]; //5
8      }
9      bool changed; // 6
10     do { // 7
11         changed = false; // 8
12         // Назначение точек кластерам
13         for (int i = 0; i < n; ++i) { //9
14             double bestDist = -1.0; //10
15             int bestCluster = 0; //11
16             for (int j = 0; j < k; ++j) { //12
17                 double dist = cosineDistance(docs[i],
                       centroids[j]);// 13
18                 if (dist > bestDist) {//14
19                     bestDist = dist;//15
20                     bestCluster = j;//16
21                 }
22             }
23             if (assignments[i] != bestCluster) { //17
24                 assignments[i] = bestCluster;//18
25                 changed = true;//19
26             }
27         }
28         // Обновление центроидов
29         std::vector<int> counts(k, 0); //20
30         std::vector<std::vector<double>> newCentroids(k,
               std::vector<double>(docs[0].size(), 0.0));//21
31         for (int i = 0; i < n; ++i) {//22
32             for (size_t j = 0; j < docs[i].size(); ++j) {//23
33                 newCentroids[assignments[i]][j] +=
```

```
                    docs[i][j];//24
34              }
35              counts[assignments[i]]++;//25
36          }
37          for (int j = 0; j < k; ++j) {//26
38              if (counts[j] != 0) {//27
39                  for (size_t m = 0; m < newCentroids[j].size();
                        ++m) {//28
40                      newCentroids[j][m] /= counts[j];//29
41                  }
42              }
43              else {//30
44                  newCentroids[j] = centroids[j]; //31
45              }
46          }
47          centroids = newCentroids;//32
48      } while (changed);//33
49
50      return centroids;//34
51  }
```

ПРИЛОЖЕНИЕ Б

Таблица Б.1 – Зависимость времени выполнения (в мс) от количества потоков

Количество потоков	Время выполнения (мс)
1	4271
2	4160
3	4146
4	4115
5	4134
6	4135
7	4144
8	4194
9	4200
10	4178
11	4207
12	4209
13	4242
14	4205
15	4240
16	4269
17	4240
18	4283
19	4287
20	4314
21	4317
22	4304
23	4454
24	4371
25	4426
26	4355
27	4376

28	4368
29	4426
30	4447
31	4503
32	4439
33	4468
34	4453
35	4476
36	4498
37	4521
38	4610
39	4615
40	4598
41	4577
42	4537
43	4564
44	4614
45	4606
46	4579
47	4893
48	4799
49	4819
50	4823
51	4825
52	4812
53	4818
54	4836
55	4849
56	4838
57	4834
58	4840
59	4815
60	4820

61	4772
62	4802
63	4806
64	4810
65	4826
66	4836
67	4823
68	4834
69	4850
70	4842
71	4857
72	4870
73	4874
74	4900
75	5056
76	5012
77	5004
78	5002
79	4947
80	4934
81	4915
82	4940
83	4944
84	4948
85	4959
86	4966
87	4978
88	4992
89	5149
90	5138
91	5126
92	5154
93	5158

94	5159
95	5161
96	5174

Таблица Б.2 – Зависимость времени выполнения (в мс) от количества входных файлов

Количество файлов	ПР(мс)	ПРОДП(мс)
1	44	47
2	88	70
3	132	108
4	178	118
5	225	153
6	271	178
7	327	220
8	359	220
9	403	249
10	445	273
11	489	310
12	542	327
13	582	363
14	623	375
15	666	424
16	729	426
17	765	480
18	811	470
19	850	516
20	909	529
21	940	564
22	987	598
23	1028	603
24	1076	631

25	1129	664
26	1159	682
27	1207	718
28	1247	729
29	1289	759
30	1342	766
31	1375	823
32	1433	803
33	1483	982
34	1531	924
35	1561	906
36	1600	928
37	1646	993
38	1693	994
39	1735	1042
40	1785	1090
41	1829	1105
42	1871	1149
43	1942	1155
44	1977	1151
45	2023	1206
46	2061	1218
47	2109	1297
48	2181	1305
49	2490	1441
50	2256	1398

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Time Management - A Case Study [Электронный ресурс]. — Режим доступа: `https://www.researchgate.net/publication/323825275_Time_Management_-_A_Case_Study` (дата обращения: 17.03.2024).
2. Introducing the Eisenhower Matrix [Электронный ресурс]. — Режим доступа: `https://www.eisenhower.me/eisenhower-matrix/` (дата обращения: 17.03.2024).
3. *Брайан Трейси.* Тайм-менеджмент. — 2015. — С. 1—144.