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»

ОТЧЕТ ПО ПРЕДДИПЛОМНОЙ ПРАКТИКЕ

Студент Наместник Анастасия Андреевна
фамилия, имя, отчество

Группа ИУ7-83Б

Тип практики Производственная

Название предприятия НУК ИУ МГТУ им. Н. Э. Баумана

Студент _____ Наместник А.А.
подпись, дата *фамилия, и.о.*

Руководитель от предприятия _____ Гаврилова Ю.М.
подпись, дата *фамилия, и.о.*

Руководитель от МГТУ им. Баумана _____ Строганов Ю.В.
подпись, дата *фамилия, и.о.*

Рекомендуемая оценка _____

2022 г.

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»

ЗАДАНИЕ
на прохождение производственной практики

на предприятии НУК ИУ МГТУ им. Н. Э. Баумана

Студент Наместник А.А. ИУ7-83Б

(фамилия, имя, отчество; инициалы; индекс группы)

Техническое задание необходимо обосновать выбор средств программной реализации метода распределенного хранения аудио-файлов в NoSQL-базе данных, описать работу ПО и провести исследование реализованного метода

Дата выдачи задания « ____ » _____ 20__ г.

Студент	_____	Наместник А.А.
	подпись, дата	*фамилия, и.о.*
Руководитель от предприятия	_____	Гаврилова Ю.М.
	подпись, дата	*фамилия, и.о.*
Руководитель от МГТУ им. Баумана	_____	Строганов Ю.В.
	подпись, дата	*фамилия, и.о.*

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Работа с аудио-файлами сложного содержания, включающего не только оцифрованный звук, но и наборы инструкций для воспроизведения звука, требует постоянного обращения к их внутренней структуре, что может быть экономически и, с точки зрения производительности, дорогостоящей операцией. Чтобы избежать накладных расходов, которые может повлечь многократное использование сторонних приложений для работы с аудио-файлами, важно рассмотреть хранение таких файлов в удобном формате, подходящем под устанавливаемые требования использования.

MongoDB - система управления базами данных, которая работает с документоориентированной моделью данных и для хранения данных использует JSON-подобный формат [1].

Целью этой работы является выбор средств программной реализации метода распределенного хранения аудио-файлов в NoSQL-базе данных, описание работы ПО и проведение исследования реализованного метода.

В рамках работы требуется решить следующие задачи.

а) Выбрать язык программирования.

б) Выбрать среду разработки.

в) Выбрать инструменты для замеров времени.

г) Описать инструкции сборки локального сервера MongoDB.

д) Описать структуру проекта.

е) Привести пример работы реализованного метода.

ж) Провести исследование и проанализировать его результаты.

1 Технологический раздел

1.1 Выбор языка программирования

В качестве языка программирования был выбран C++, так как на нем написана серверная часть базы данных MongoDB. Соответственно, метод хранения аудио-файлов для MongoDB был разработан, как патч для действующей версии СУБД.

1.2 Выбор среды разработки

Для программной реализации метода был выбран редактор кода Visual Studio Code [2], так как он обладает следующими достоинствами.

а) Кроссплатформенность.

б) Поддержка большого набора языков программирования, включая C++.

в) Широкие возможности кастомизации.

г) Открытый исходный код.

1.3 Выбор инструментов для замеров времени

Для доступа к базе данных использовался драйвер C# для MongoDB – MongoDB.Driver [3], обеспечивающий асинхронное взаимодействие с MongoDB. MongoDB.Driver предоставляет API для подключения к серверу базы данных и выполнения запросов любой сложности через клиента. Такой подход наиболее удобен, когда планируется последующая программная обработка данных, как в случае с замером времени работы с данными при выбранном методе хранения. Программное обеспечение было реализовано в интегрированной среде разработки Rider [4], которая является кроссплатформенной, предоставляет удобные и быстрые редактор кода и отладчик, поддерживает платформу .Net [5] для создания приложений на языке C#, а также имеет свободный доступ для студентов. Установка драйвера была произведена с помощью пакетного менеджера NuGet [6], имеющего удобный интерфейс в среде Rider, и была интегрирована в сборку проекта с помощью системы управления зависимо-

стями .Net с указанием версии используемого пакета. На рис. 1.1 представлен фрагмент сборки приложения с драйвером MongoDB.Driver.

```
<ItemGroup>
  <PackageReference Include="MongoDB.Driver" Version="2.15.1" />
</ItemGroup>
```

Рис 1.1 — Фрагмент сборки .Net приложения с интеграцией драйвера MongoDB.Driver

Для замеров времени работы реализованного метода при тестировании и сравнении его с аналогом хранения использовалась бибилиотека BenchmarkDotNet [7]. BenchmarkDotNet - это легкая и мощная библиотека .NET с открытым исходным кодом, которая может преобразовывать методы в тесты, отслеживать эти методы, а затем предоставлять анализ полученных данных о производительности. BenchmarkDotNet обеспечивает высокую точность полученных результатов благодаря использованию набора инструментов для анализа производительности Perfolizer [8]. Широкий спектр возможностей, которые предоставляет данная библиотека, включает в себя анализ тестов, предупреждающий ошибки, и экспорт результатов сравнительного анализа методов в необходимом пользователю формате. Результатом запуска метода Run() данного инструмента со стандартной конфигурацией будет сводная таблица показателей производительности тестируемых методов. BenchmarkDotNet была установлена в среду разработки так же с помощью менеджера пакетов NuGet и интегрирована в сборку проекта системой управления зависимостями. На рис. 1.2 представлен фрагмент сборки приложения с библиотекой BenchmarkDotNet.

```
<ItemGroup>
  <PackageReference Include="BenchmarkDotNet" Version="0.13.1" />
</ItemGroup>
```

Рис 1.2 — Фрагмент сборки .Net приложения с интеграцией библиотеки BenchmarkDotNet

1.4 Сборка локального сервера MongoDB

Следуя инструкции по сборке MongoDB из исходного кода, расположенной в официальном репозитории базы данных [9], для компиляции сервера MongoDB использовался инструмент для автоматизации сборки программных проектов SCons [10]. Команда, использовавшаяся для запуска компиляции сервера MongoDB в командной строке с помощью SCons, представлена в листинге 1.1, где

а) dbg – вывод отладочной информации.

б) j8 – количество логических ядер, между которыми распределяется нагрузка при компиляции (8).

в) install-mongod – цель, указывающая какой компонент нужно скомпилировать.

Листинг 1.1 — Команда для компиляции сервера MongoDB средствами SCons

```
1 sudo scons --dbg -j8 install-mongod
```

Для сборки SCons использует скрипты SConstruct, расположенные в файловой структуре проекта, одними из задач которых является импортирование необходимых модулей Python и проверка инструментов, участвующих в компиляции, на соответствие требованиям.

Для успешной сборки определены следующие требования.

а) Современный C++ компилятор. Подходят следующие компиляторы.

1) G++ 8.2 [11] или новее.

2) Clang 7.0 [12] или новее.

б) В операционных системах Linux [13] и macOS [14] требуется библиотека и заголовок libcurl [15]. MacOS включает libcurl.

в) Python 3.7 и некоторые модули, устанавливаемые с помощью команды, представленной в листинге 1.2.

г) Около 13 ГБ свободного места на диске для скомпилированных двоичных файлов.

Листинг 1.2 — Команда для установки необходимых для сборки модулей Python

```
python3 -m pip install -r etc/pip/compile-requirements.txt
```

SCons также предоставляет возможность скомпилировать отдельные компоненты MongoDB, задав одну или несколько из следующих целей.

а) install-mongod – сервер базы данных.

б) install-mongos – шардинг.

в) install-servers – включает install-mongod и install-mongos.

г) install-core – включает install-mongod и install-mongos.

д) install-mongosh – полнофукционольная среда для взаимодействия с развертываниями MongoDB, использующая интерфейс командной строки.

е) install-all – все компоненты.

1.5 Структура проекта

В проект базы данных MongoDB было добавлено два исходных файла, реализующих логику добавления и извлечения MIDI-файла, а также заголовочные файлы для них.

Так как MIDI-файл, в соответствии с разработанным методом, хранится в виде предварительно распарсенной структуры, программный код, выполняющий считывание потока байтов из файла и конструирование из него документа, находится в части проекта, отвечающей за выполнение операции вставки в базу данных. На этапе валидирования добавляемого документа выполняется проверка на то, содержит ли он путь к MIDI-файлу, и если да – документ пересобирается с использованием парсера, чтобы впоследствии представлять внутреннюю структуру аудио-файла.

В проект по пути src/mongo/db/ops/ были добавлены следующие файлы для парсинга MIDI-файла на этапе вставки документа в MongoDB.

а) insert_midi.cpp – исходный файл, реализующий чтение MIDI-файла и парсер.

б) insert_midi.h – заголовочный файл.

Для дальнейшей работы с добавленным документом, как с аудио-файлом, предусмотрено обратное преобразование структуры в MIDI-файл в процессе извлечения документы из базы данных. Для этого добавлена проверка на то, что запись в MongoDB является MIDI-файлом, и последовательная запись данных в файл с именем, которое было указано пользователем в качестве второго поля при вставке аудио-файла (name) или, в случае если пользователь указал только путь к файлу, извлечено из пути.

В проект по пути src/mongo/db/commands/ были добавлены следующие файлы для воссоздания MIDI-файла на этапе извлечения документа из MongoDB.

а) find_midi.cpp – исходный файл, реализующий обратное преобразование документа в MIDI-файл.

б) find_midi.h – заголовочный файл.

Для успешной сборки проекта после добавления новых файлов требуется указать их в конфигурации соответствующих файлов SConstruct. Следовательно, были изменены следующие скрипты.

а) src/mongo/db/SConscript.

б) src/mongo/db/commands/SConscript.

1.6 Пример работы реализованного метода

Для демонстрации работы метода был создан клиент MongoDB на языке Python с помощью библиотеки pymongo [16]. Помимо поддержки API доступа к MongoDB и работы с ней, Python предоставляет возможность загружать и проигрывать аудиозаписи различных форматов, как фоновую музыку, с использованием средств библиотеки pygame [17]. Также Python

содержит бибилиотеку tkinter [18], с помощью которой был реализован простой графический интерфейс для удобства работы с базой данных.

Однако, независимо от используемого клиента, структура данных для запроса на добавление MIDI-файла в базу данных примет следующий вид (листинг 1.3).

Листинг 1.3 — Данные для запроса на добавление MIDI-файла в MongoDB

```
{path: "/Users/anastasia/Desktop/ProgramEngineering/some_audio.mid",
    name:"my_audio"}
```

Для извлечения данных с последующим сохранением MIDI-файла с названием my _audio достаточно установить в фильтре запроса на извлечение значение True поля MidiSave. Если не нужно сохранять аудио-файл локально, значение поля MidiSave в фильтре можно установить равным False или опустить.

На рисунке 1.3 представлен реализованный для демонстрации графический интерфейс.

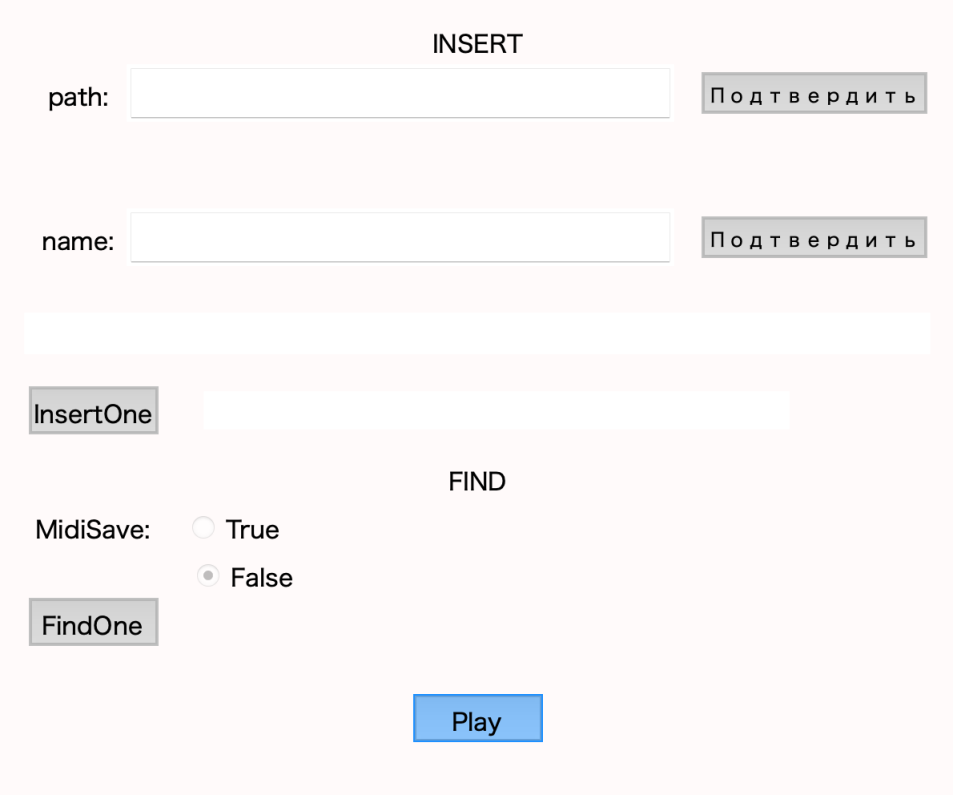

Рис 1.3 — Графический интерфейс для демонстрации работы метода

После заполнения полей path и name будут сформированы данные для запроса к MongoDB на вставку, который бдует выполнен при нажатии на кнопку InsertOne. Результатом успешной операции является идентификатор добавленного в базу данных документа (рис. 1.4).

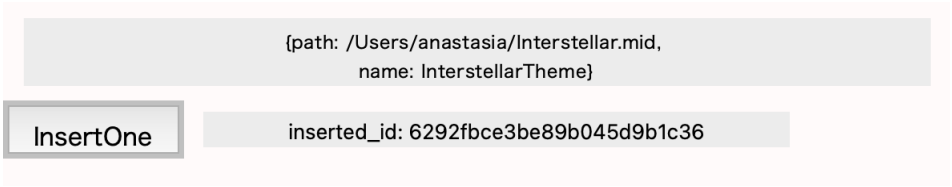

Рис 1.4 — Результат успешного добавления MIDI-файла в MongoDB

Для извлечения документа с сохранением MIDI-файла полю MidiSave устанавливается значение True. В таком случае документ, извлеченный из базы данных, будет сохранен локально, как файл MIDI с именем InterstellarTheme, который может быть воспроизведен с помощью кнопки Play.

Для удобного просмотра документа в базе данных можно также использовать команду mongosh (листинг 1.4), где midiFiles – это коллекция документов.

Листинг 1.4 — Команда mongosh для просмотра документов в MongoDB

```
1 db.midiFiles.find()
```

Результатом выполнения этой команды будет вывод всех документов коллекции (в данном случае, одного только что добавленного документа, так как коллекция была пуста) в консоль (рис. 1.5 – 1.6).

Рис 1.5 — Результат извлечения MIDI-файла из MongoDB (часть 1)

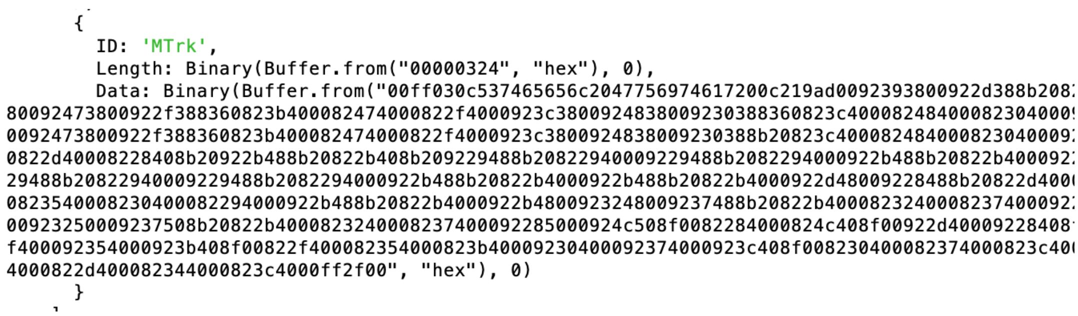

Рис 1.6 — Результат извлечения MIDI-файла из MongoDB (часть 2)

1.7 Выводы из технологического раздела

В данном разделе был обоснован выбор средств программной разработки метода хранения аудио-файлов в NoSQL-базе данных и вспомогательных инструментов для замера времени работы метода и демонстрации его работы. Также была приведена инструкция сборки проекта MongoDB с интегрированным методом и были описаны изменения его структуры. Было проведено

тестирование метода с использованием графического интерфейса и интерфейса командной строки.

2 Исследовательский раздел

2.1 Предмет исследования

Для проведения исследования работы реализованного метода хранения аудио-файлов в NoSQL-базе данных было составлено 9 MIDI-файлов, отличающихся только количеством музыкальных дорожек (MTrk). Таким образом, первый файл содержит одну музыкальную дорожку, второй - 2, ..., девятый - 9. Данные дорожек не отличаются, в зависимости от их количества незначительно меняется заголовок MIDI-файла: поля Format и NumTracks. Так, например, аудио-файл, содержащий одну музыкальную дорожку, имеет в базе данных заголовок, представленный на рисунке 2.1.

```
ID: 'MThd',
Name: 'InterstellarTheme',
Length: Binary(Buffer.from("00000006", "hex"), 0),
Format: Binary(Buffer.from("0000", "hex"), 0),
NumTracks: Binary(Buffer.from("0001", "hex"), 0),
Division: Binary(Buffer.from("01e0", "hex"), 0),
```

Рис 2.1 — Заголовок MIDI-файла, содержащего одну MTrk

Заголвок MIDI-файла с девятью дорожками представлен на рисунке 2.2.

```
ID: 'MThd',
Name: 'InterstellarTheme',
Length: Binary(Buffer.from("00000006", "hex"), 0),
Format: Binary(Buffer.from("0001", "hex"), 0),
NumTracks: Binary(Buffer.from("0009", "hex"), 0),
Division: Binary(Buffer.from("01e0", "hex"), 0),
```

Рис 2.2 — Заголовок MIDI-файла, содержащего девять MTrk

Цель исследования - сравнить временные показатели работы с MIDI-файлом при реализованном методе хранения в MongoDB и при хранении его в базе данных в виде неструктурированного массива байтов. Исследуется работа методов хранения при следующих командах базы данных.

а) Только добавление документа в MongoDB.

б) Только извлечение документа из MongoDB (с последующим доступом к данным самой последней музыкальной дорожки).

в) Добавление и извлечение документа из MongoDB (с последующим доступом к данным самой последней музыкальной дорожки).

В случае реализованного метода распределенного хранения MIDI-файлов, все девять файлов будут храниться в базе данных в виде специальной структуры. Сопоставляемый с ним метод предполагает хранение в виде документа, содержащего один элемент с полем Data, имеющим значение, равное массиву байтов, считанному из аудио-файла. На рисунке 2.3 приведен пример того, как в MongoDB будет представлен такой файл.

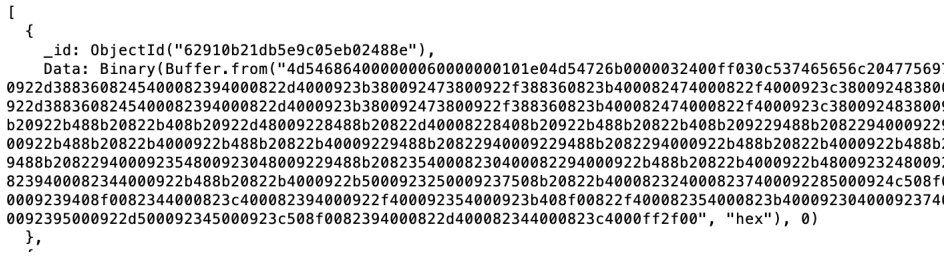

Рис 2.3 — Метод хранения MIDI-файла в MongoDB в виде массива байтов

2.2 Характеристики ЭВМ

Для исследования использовалась ЭВМ со следующими характеристиками.

— MacBook Pro (Retina, 15-inch, Mid 2014).

— 2,5 GHz Intel Core i7.

— Число логических ядер: 8.

2.3 Результаты исследования

2.3.1 Добавление документа в MongoDB

Для добавления MIDI-файла в базу данных для хранения в структурированном виде был реализован метод, формирующий документ MongoDB с помощью пути к аудио-файлу и его имени (листинг 2.1). После инициализации документа созданный объект добавляется в базу данных.

Листинг 2.1 — Добавление MIDI-файла в MongoDB с использованием реализованного метода

```
1 private async void AddDocument(string number, string name)
2         {
3             var document = new BsonDocument { { "path",
                  "/Users/anastasia/Desktop/Audiofiles/Interstellar" + number +
                  ".mid" },
4                  { "name", name } };
5             _midiCollection.InsertOne(document);
6         }
```

При вставке массива байтов аудио-файла он считывается из соответствующего файла в переменную array, которая используется для инициализации данных специального типа (бинарный массив) и формирования из них документа MongoDB. По завершении этих действий документ добавляется в базу данных (листинг 2.2).

Листинг 2.2 — Добавление MIDI-файла в MongoDB в виде массива байтов

```
1 private async void AddBytesMongoDb(byte[] array)
2         {
3             var bsonBinaryData = new BsonBinaryData(array);
4             var document = new BsonDocument(new BsonElement("Data",
                  bsonBinaryData));
5             _midiCollection.InsertOne(document);
6         }
```

На рисунке 2.4 представлен результат работы методов хранения. Пояснения исследуемых характеристик приведены на рисунке 2.5.

Method	Mean	Error	StdDev
RawMidiOneTrackTest	5.520 ms	0.0558 ms	0.0522 ms
RawMidiTwoTracksTest	5.488 ms	0.0524 ms	0.0490 ms
RawMidiThreeTracksTest	5.502 ms	0.0379 ms	0.0354 ms
RawMidiFourTracksTest	5.509 ms	0.0425 ms	0.0397 ms
RawMidiFiveTracksTest	5.526 ms	0.0491 ms	0.0459 ms
RawMidiSixTracksTest	5.527 ms	0.0691 ms	0.0647 ms
RawMidiSevenTracksTest	5.525 ms	0.0620 ms	0.0580 ms
RawMidiEightTracksTest	5.498 ms	0.0356 ms	0.0333 ms
RawMidiNineTracksTest	5.503 ms	0.0583 ms	0.0546 ms
ParsedMidiOneTrackTest	5.496 ms	0.0473 ms	0.0442 ms
ParsedMidiTwoTracksTest	5.567 ms	0.0798 ms	0.0708 ms
ParsedMidiThreeTracksTest	5.693 ms	0.0491 ms	0.0460 ms
ParsedMidiFourTracksTest	5.751 ms	0.0648 ms	0.0607 ms
ParsedMidiFiveTracksTest	5.940 ms	0.0663 ms	0.0554 ms
ParsedMidiSixTracksTest	6.097 ms	0.0542 ms	0.0507 ms
ParsedMidiSevenTracksTest	6.263 ms	0.0943 ms	0.0882 ms
ParsedMidiEightTracksTest	6.298 ms	0.0439 ms	0.0411 ms
ParsedMidiNineTracksTest	6.364 ms	0.0446 ms	0.0417 ms

Рис 2.4 — Сводная таблица работы методов при добавлении в MongoDB

```
// * Legends *
  Mean   : Arithmetic mean of all measurements
  Error  : Half of 99.9% confidence interval
  StdDev : Standard deviation of all measurements
  Median : Value separating the higher half of all measurements (50th percentile)
  1 ms   : 1 Millisecond (0.001 sec)
```

Рис 2.5 — Сводная таблица работы методов при добавлении в MongoDB

На рисунке 2.6 полученные временные характеристики представлены в графическом виде.

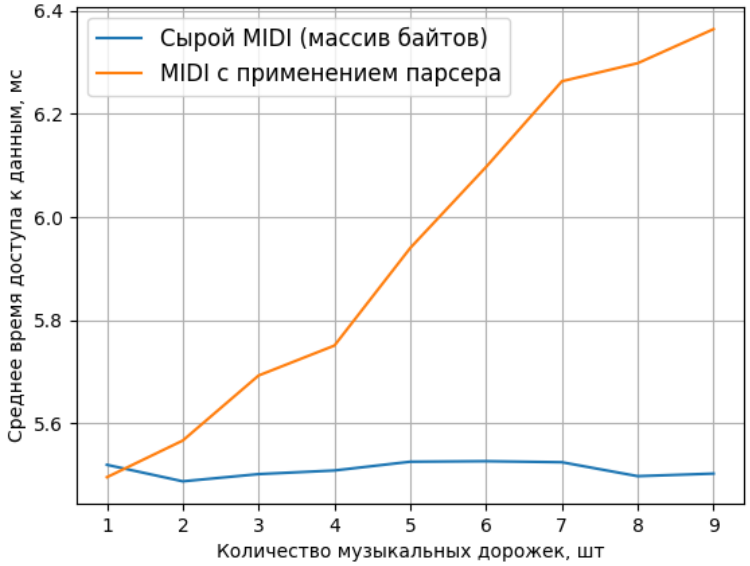

Рис 2.6 — Сравнение работы методов при добавлении в MongoDB

Как видно из рисунков 2.4 и 2.6, наибольшая скорость работы операции добавления документа в MongoDB достигается при использовании метода хранения MIDI-файла на основе его содержания. Однако, при увеличении количества музыкальных дорожек время работы этого метода растет пропорционально числу дорожек, что объясняется тем, что каждый раз при вставке аудио-файла в базу данных применяется парсер. Скорость роста составляет примерно 0,096 мс/шт. Таким образом, реализация хранения в виде массива байтов имеет выигрыш по временной эффективности приблизительно в 1,2 раза.

2.3.2 Извлечение документа из MongoDB

Для извлечения MIDI-файла из базы данных и доступа к его последней дорожке, в случае хранения в структурированном виде, требуется только обращение к документу MongoDB с поиском значения нужного поля (листинг 2.3).

Листинг 2.3 — Извлечение MIDI-файла из MongoDB с использованием реализованного метода

```
1 var documents = _midiCollection.Find(new BsonDocument()).ToList();
2 var parsedDocument = documents.Last();
3
4 var result =
      parsedDocument.GetElement("MTrks").Value.AsBsonArray.Last().AsBsonDocument
5     .GetElement("Data").Value.AsByteArray;
```

При извлечении массива байтов аудио-файла, чтобы найти нужные данные, сначала нужно применить парсер, который преобразует бинарный объект в определенную структуру, по которой можно выполнить поиск. Метод ParseMidiFile использует ту же структуру и средства, что и парсер, используемый в разработанном методе хранения (листинг 2.4).

Листинг 2.4 — Извлечение MIDI-файла из MongoDB в виде массива байтов

```
1 var documents = _midiCollection.Find(new BsonDocument()).ToList();
2 var rawDocument = documents.Last();
3
4 var midiFileRaw = rawDocument.GetElement("Data").Value.AsByteArray;
5 var midiFile = _midiFileParser.ParseMidiFile(midiFileRaw);
6 var result = midiFile.MTrks.Last().Data;
```

На рисунке 2.7 представлен результат работы методов хранения. Пояснения исследуемых характеристик приведены на рисунке 2.5.

Method	Mean	Error	StdDev	Median
RawMidiOneTrackTest	22.90 ms	0.441 ms	0.413 ms	22.77 ms
RawMidiTwoTracksTest	32.72 ms	0.328 ms	0.291 ms	32.68 ms
RawMidiThreeTracksTest	41.82 ms	0.497 ms	0.388 ms	41.75 ms
RawMidiFourTracksTest	52.95 ms	1.058 ms	2.880 ms	51.55 ms
RawMidiFiveTracksTest	61.20 ms	0.522 ms	0.489 ms	61.22 ms
RawMidiSixTracksTest	69.38 ms	0.593 ms	0.525 ms	69.36 ms
RawMidiSevenTracksTest	78.59 ms	0.315 ms	0.294 ms	78.57 ms
RawMidiEightTracksTest	88.19 ms	1.081 ms	1.011 ms	87.84 ms
RawMidiNineTracksTest	96.79 ms	0.556 ms	0.493 ms	96.81 ms
ParsedMidiOneTrackTest	10.99 ms	0.178 ms	0.149 ms	10.96 ms
ParsedMidiTwoTracksTest	10.90 ms	0.124 ms	0.116 ms	10.89 ms
ParsedMidiThreeTracksTest	11.13 ms	0.164 ms	0.145 ms	11.15 ms
ParsedMidiFourTracksTest	11.07 ms	0.147 ms	0.130 ms	11.11 ms
ParsedMidiFiveTracksTest	11.09 ms	0.212 ms	0.188 ms	11.12 ms
ParsedMidiSixTracksTest	11.10 ms	0.098 ms	0.092 ms	11.07 ms
ParsedMidiSevenTracksTest	11.09 ms	0.221 ms	0.207 ms	11.09 ms
ParsedMidiEightTracksTest	11.12 ms	0.138 ms	0.129 ms	11.14 ms
ParsedMidiNineTracksTest	11.13 ms	0.210 ms	0.196 ms	11.14 ms

Рис 2.7 — Сводная таблица работы методов при извлечении из MongoDB

На рисунке 2.8 полученные временные характеристики представлены в графическом виде.

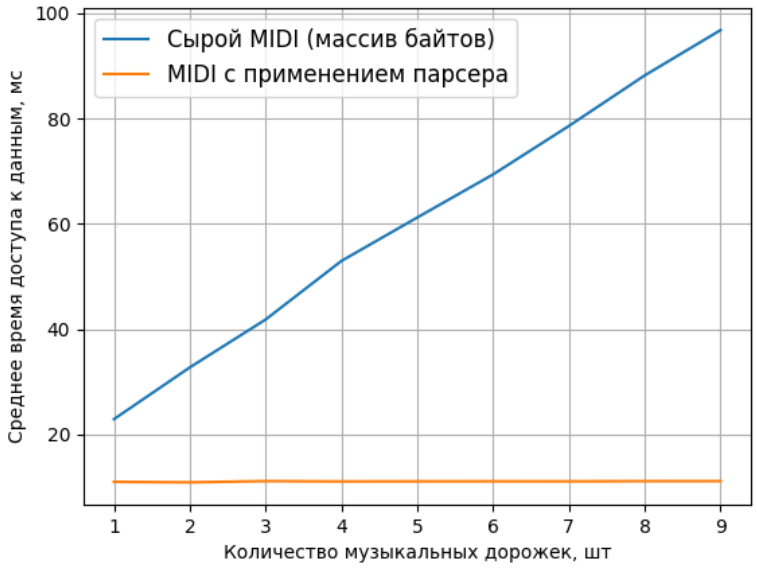

Рис 2.8 — Сравнение работы методов при извлечении из MongoDB

Из рисунков 2.7 и 2.8 видно, что наибольшая скорость выполнения операции чтения документа из MongoDB достигается при использовании метода

хранения MIDI-файла в структурированном виде. При этом скорость работы почти не меняется при увеличении числа музыкальных дорожек, разница во времени выполнения доступа к данным при девяти дорожках и одной дорожке составляет 0,14 миллисекунд. Время работы доступа к данным при методе хранения массива байтов, наоборот, растет из-за накладных расходов применяемого парсера. Скорость роста составляет 8,21 мс/шт. Таким образом, реализация хранения в виде структуры MIDI-файла имеет выигрыш по временной эффективности приблизительно в 8,7 раза.

2.3.3 Добавление и извлечение документа из MongoDB

Для оценки времени работы обеих операций добавления и извлечения документов из MongoDB при разных подходах к хранению данных использовались описанные выше методы в совокупности. Для исследования добавления и извлечения MIDI-файла, хранящегося в виде струкутры, были использованы методы, представленные на листинге 2.5.

Листинг 2.5 — Добавление и извлечение MIDI-файла из MongoDB с использованием реализованного метода

```
1 AddDocument("1", "Interstellar1");
2
3 var documents = _midiCollection.Find(new BsonDocument()).ToList();
4 var parsedDocument = documents.Last();
5
6 var result =
      parsedDocument.GetElement("MTrks").Value.AsBsonArray.Last().AsBsonDocument
7     .GetElement("Data").Value.AsByteArray;
```

Для исследования добавления и извлечения MIDI-файла, хранящегося в виде массива байтов, были использованы методы, представленные на листинге 2.6.

Листинг 2.6 — Добавление и извлечение MIDI-файла из MongoDB в виде массива байтов

```
1
2  AddBytesMongoDb(FileOperations
3      .ReadFileAsync("/Users/anastasia/Interstellar1.mid"));
4
5  var documents = _midiCollection.Find(new BsonDocument()).ToList();
6  var rawDocument = documents.Last();
7
8  var midiFileRaw = rawDocument.GetElement("Data").Value.AsByteArray;
9  var midiFile = _midiFileParser.ParseMidiFile(midiFileRaw);
10 var result = midiFile.MTrks.Last().Data;
```

На рисунке 2.9 представлен результат работы методов хранения. Пояснения исследуемых характеристик приведены на рисунке 2.5.

Method	Mean	Error	StdDev
RawMidiOneTrackTest	10.194 ms	0.0946 ms	0.0885 ms
RawMidiTwoTracksTest	29.551 ms	0.5284 ms	0.4943 ms
RawMidiThreeTracksTest	37.534 ms	0.2818 ms	0.2636 ms
RawMidiFourTracksTest	46.647 ms	0.4267 ms	0.3991 ms
RawMidiFiveTracksTest	56.645 ms	1.0819 ms	1.1576 ms
RawMidiSixTracksTest	65.571 ms	0.8391 ms	0.7438 ms
RawMidiSevenTracksTest	74.773 ms	0.3104 ms	0.2592 ms
RawMidiEightTracksTest	84.500 ms	1.6403 ms	1.7551 ms
RawMidiNineTracksTest	93.152 ms	0.7455 ms	0.6226 ms
ParsedMidiOneTrackTest	8.853 ms	0.0925 ms	0.0772 ms
ParsedMidiTwoTracksTest	9.284 ms	0.0940 ms	0.0879 ms
ParsedMidiThreeTracksTest	9.712 ms	0.1042 ms	0.0974 ms
ParsedMidiFourTracksTest	10.110 ms	0.1214 ms	0.1076 ms
ParsedMidiFiveTracksTest	10.579 ms	0.1144 ms	0.1070 ms
ParsedMidiSixTracksTest	10.858 ms	0.0891 ms	0.0833 ms
ParsedMidiSevenTracksTest	11.315 ms	0.1049 ms	0.0930 ms
ParsedMidiEightTracksTest	11.806 ms	0.1040 ms	0.0972 ms
ParsedMidiNineTracksTest	12.184 ms	0.1459 ms	0.1365 ms

Рис 2.9 — Сводная таблица работы методов при извлечении из MongoDB

На рисунке 2.10 полученные временные характеристики представлены в графическом виде.

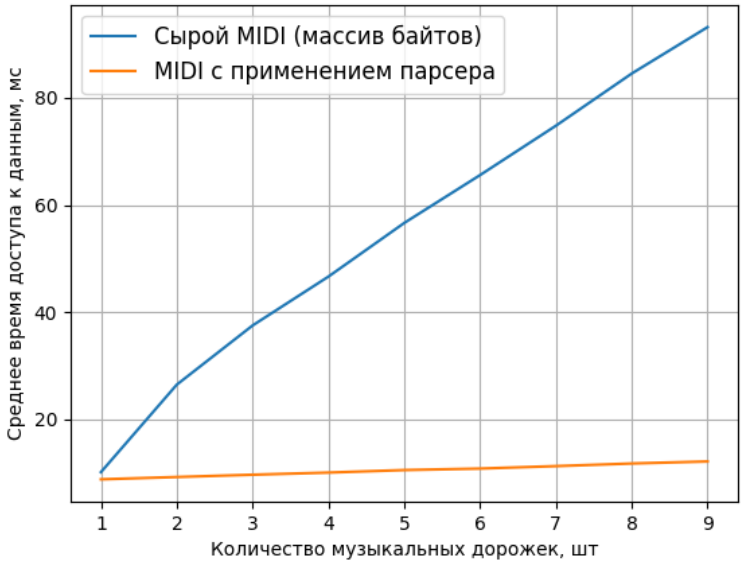

Рис 2.10 — Сравнение работы методов при добавлении и извлечении из MongoDB

Из рисунков 2.9 и 2.10 очевидно, что наибольшая скорость работы операций записи и чтения документа из MongoDB достигается при использовании метода хранения MIDI-файла в структурированном виде. Время работы метода медленно растет при увеличении числа музыкальных дорожек, а скорость роста составляет приблизительно 0,37 мс/шт. Время работы операций записи и чтения при методе хранения массива байтов резко растет, и скорость роста составляет примерно 9,22 мс/шт. Таким образом, реализация хранения MIDI-файла в виде его структуры имеет выигрыш по временной эффективности приблизительно в 7,6 раза.

2.4 Выводы из исследовательского раздела

В данном разделе было проведено исследование временных характеристик работы операций добавления и извлечения документа из MongoDB при двух способах хранения: с использованием разработанного метода и без использования разработанного метода (хранение в виде массива байтов). По результатам исследования, метод распределенного хранения MIDI-файла в виде его структуры показал немного меньшую эффективность при операции добавления в MongoDB, но значительно большую эффективность при

операции извлечения данных, а также при работе обеих этих операций в совокупности.

ЗАКЛЮЧЕНИЕ

В ходе работы были выбраны средства программной реализации метода распределенного хранения аудио-файлов в NoSQL-базе данных, была описана работа ПО, а также было проведено исследование реализованного метода.

В рамках работы были решены все поставленные задачи.

а) Выбран язык программирования.

б) Выбрана среда разработки.

в) Выбраны инструменты для замеров времени.

г) Описаны инструкции сборки локального сервера MongoDB.

д) Описана структуру проекта.

е) Приведен пример работы реализованного метода.

ж) Проведено исследование и проанализированы его результаты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. MongoDB [Электронный ресурс]. — Режим доступа: https://www.mongodb.com (Дата обращения 2022-05-23).

2. VSC [Электронный ресурс]. — Режим доступа: code.visualstudio.com (Дата обращения 2022-05-25).

3. MongoDB C#/.NET Driver [Электронный ресурс]. — Режим доступа: https://www.mongodb.com/docs/drivers/csharp/ (Дата обращения 2022-04-29).

4. Rider [Электронный ресурс]. — Режим доступа: https://www.jetbrains.com/ru-ru/rider/ (Дата обращения 2022-05-25).

5. NET | Free. Cross-platform. Open Source [Электронный ресурс]. — Режим доступа: https://dotnet.microsoft.com/en-us/ (Дата обращения 2022-03-19).

6. NuGet [Электронный ресурс]. — Режим доступа: https://www.nuget.org (Дата обращения 2022-04-29).

7. BenchmarkDotNet [Электронный ресурс]. — Режим доступа: https://benchmarkdotnet.org/ (Дата обращения 2022-05-20).

8. perfolizer | Performance analysis toolkit | Code Analyzer library [Электронный ресурс]. — Режим доступа: https://kandi.openweaver.com/csharp/AndreyAkinshin/perfolizer (Дата обращения 2022-05-20).

9. Building MongoDB [Электронный ресурс]. — Режим доступа: https://github.com/mongodb/mongo/blob/master/docs/building.md (Дата обращения 2022-05-05).

10. SCons: A software construction tool - SCons [Электронный ресурс]. — Режим доступа: https://scons.org (Дата обращения 2022-05-05).

11. GCC, the GNU Compiler Collection [Электронный ресурс]. — Режим доступа: https://gcc.gnu.org (Дата обращения 2022-05-05).

12. Clang: a C language family frontend for LLVM [Электронный ресурс]. — Режим доступа: https://clang.llvm.org (Дата обращения 2022-05-05).

13. Linux [Электронный ресурс]. — Режим доступа: https://www.linux.org.ru (Дата обращения 2022-05-07).

14. macOS Monterey [Электронный ресурс]. — Режим доступа: https://www.apple.com/ru/macos/monterey/ (Дата обращения 2022-05-07).

15. libcurl - the multiprotocol file transfer library [Электронный ресурс]. — Режим доступа: https://curl.se/libcurl/ (Дата обращения 2022-05-05).

16. pymongo 4.1.1 [Электронный ресурс]. — Режим доступа: https://pypi.org/project/pymongo/ (Дата обращения 2022-05-25).

17. Pygame [Электронный ресурс]. — Режим доступа: https://www.pygame.org/news (Дата обращения 2022-05-25).

18. Tkinter [Электронный ресурс]. — Режим доступа: https://docs.python.org/3/library/tkinter.html (Дата обращения 2022-05-25).