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 ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

Метод распределенного хранения аудио-файлов в NoSQL базе данных.

Студент группы **ИУ7-83Б**	_____	**А. А. Наместник**
	(Подпись, дата)	(И.О. Фамилия)
Руководитель ВКР	_____	**Ю. М. Гаврилова**
	(Подпись, дата)	(И.О. Фамилия)
Нормоконтролер	_____	**Д. Ю. Мальцева**
	(Подпись, дата)	(И.О. Фамилия)

***2022** г.*

РЕФЕРАТ

Расчетно-пояснительная записка содержит 75 страниц, 7 таблиц, 45 источников.

Ключевые слова: MIDI, NoSQL, MongoDB, BsonDocument, хранение.

Цель бакалаврской работы - разработка метода распределенного хранения аудио-файлов в NoSQL-базе данных.

Задачи, решаемые в работе:

а) Рассмотреть способы представления звуковой информации и проанализировать существующие аудио-форматы.

б) Проанализировать способы хранения аудио-файлов в различных базах данных.

в) Разработать метод хранения аудио-файлов.

г) Разработать программное обеспечение, реализующее разработанный метод.

д) Исследовать зависимость времени работы метода от количества дорожек в аудио-файле.

Область применения разрабатываемого метода - работа с аудио-файлами формата MIDI в музыкальной индустрии.

В данной выпускной бакалаврской работе проанализированы существующие аудио-форматы, выбран аудио-формат, дающий наиболее полную информацию о музыке, проанализированы существующие решения хранения такого формата в различных базах данных и выбран способ хранения, наиболее оптимальный для выбранного формата, проанализированы существующие модели данных и выбрана наиболее подходящая под выбранный способ хранения модель, а также выбрана база данных, которая ее использует. Разработано программное обеспечение, реализующее описанный метод. Произведено исследование времени работы метода и проанализированы полученные результаты.

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.

Цель этой работы - разработать и реализовать метод распределенного хранения аудио-файлов в NoSQL-базе данных.

В рамках работы требуется решить следующие задачи.

а) Рассмотреть способы представления звуковой информации и проанализировать существующие аудио-форматы.

б) Проанализировать способы хранения аудио-файлов в различных базах данных.

в) Разработать метод хранения аудио-файлов.

г) Разработать программное обеспечение, реализующее разработанный метод.

д) Исследовать зависимость времени работы метода от количества дорожек в аудио-файле.

1 Аналитический раздел

1.1 Анализ предметной области

1.1.1 Представление звуковой информации

Звук — это физическое явление, представляющее собой распространение упругих волн механических колебаний в твёрдой, жидкой или газообразной среде [1]. Звуковая волна характеризуется меняющейся амплитудой и частотой [2] и воспринимается человеком с помощью органов слуха. Существует два способа представления звуков: прямая передача звука, при которой звук, передаваемый через электроакустический преобразователь, например, микрофон, обрабатывается и хранится таким образом, чтобы отображать исходные механические колебания; через набор параметров, описывающих звуки, которые можно использовать для воссоздания звука.

Физическим характеристикам звуковой волны соответствуют физиологические характеристики, связанные со слуховыми ощущениями человека [3]. Например, амплитуда звуковых колебаний воспринимается человеком как громкость [2], а частота колебаний как высота тона звука [4]. Здоровый молодой человек способен слышать звуковые колебания в диапазоне частот от 20 Гц до 20 кГц [4], что соответствует 20 — 20 000 колебаний в секунду, однако между людьми существуют значительные различия, особенно на высоких частотах, и постепенная потеря чувствительности к более высоким частотам с возрастом считается нормой. При записи и последующем воспроизведении звука производится ряд преобразований сигнала, характер которых изменяется в зависимости от выбранного способа сохранения звука и используемых технологий.

1.1.2 Качество звука

С точки зрения восприятия звука слуховым аппаратом человека, качество звука - это характеристика, определяющая, насколько точно синтезированный звук соответствует реальному источнику звука из окружающего мира [3]. В более узком смысле, под качеством звука понимается способность уха

различать звуки одинаковой высоты и громкости [5]. Однако, если отталкиваться от физических характеристик звука, качество оцифрованного звука зависит от *частоты дискретизации* — количества измерений амплитуды входного сигнала в единицу времени, и от *глубины кодирования* — количества бит, которое необходимо для кодирования дискретных уровней громкости цифрового звука [6].

1.1.3 Преобразования сигнала при записи и воспроизведении звука

Звуковые колебания воздуха преобразуются в механические колебания чувствительного элемента инструмента, использующегося для звукозаписи, например, микрофона, которые впоследствии могут быть преобразованы в электрический сигнал. *Аналоговой* звукозаписью называется запись звуков на физический носитель таким образом, чтобы устройство воспроизведения производило колебания и создавало звуковые волны аналогичные тем, что были получены при сохранении. Для записи аналогового звука и его преобразования в цифровую форму используется микрофон, подключенный к звуковой плате. Чтобы иметь возможность обрабатывать звук с помощью компьютера, аналоговую запись необходимо преобразовать в *дискретную*, то есть представить непрерывный сигнал в виде последовательности электрических импульсов [7]. Чтобы закодировать звук, необходимо произвести временную дискретизацию непрерывного звукового сигнала, а именно измерять амплитуду сигнала через небольшие промежутки времени. На каждом временном отрезке определяется средняя амплитуда сигнала. При восстановлении исходной кривой ее вид будет искажен. Чем промежутки времени меньше, тем выше будет качество закодированного звука. Амплитуда сигнала, определенная в каждый момент времени, должна быть представлена в числовом виде. В простейшем случае можно использовать один бит – есть звук или нет. Но на практике такое кодирование не имеет смысла. Допускается для кодирования амплитуды сигнала взять восемь бит – один байт, что позволяет описать двести пятьдесят шесть уровней громкости. Качество звука при этом получается не слишком высокое. Если и частота дискретизации невелика, то при воспроизведении будут присутствовать сильные искажения. Значительно лучшее качество полу-

чается при использовании двух байт, что позволяет задать более шестидесяти пяти тысяч разных значений амплитуды. В большинстве случаев двух байт достаточно для получения высококачественной записи звука, хотя иногда применяют 24 бита – три байта для кодирования амплитуды сигнала [8]. Процесс преобразования аналоговой записи в дискретную называется оцифровкой.

Существуют различные методы кодирования звуковой информации двоичным кодом, среди которых можно выделить два основных направления: метод FM и метод Wave-Table.

Метод FM (Frequency Modulation) основан на том, что теоретически любой сложный звук можно разложить на последовательность простейших гармонических сигналов разных частот, каждый из которых представляет собой правильную синусоиду, и следовательно, может быть описан кодом [9]. Разложение звуковых сигналов в гармонические ряды и представление в виде дискретных цифровых сигналов выполняют специальные устройства — аналогово-цифровые преобразователи (АЦП).

На рисунках 1.1 и 1.2 представлено преобразование непрерывного звукового сигнала в дискретный сигнал.

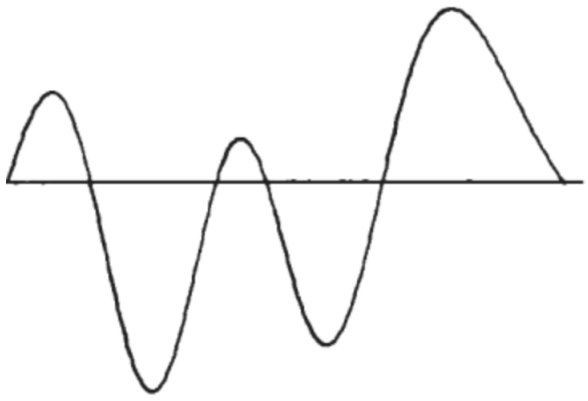

Рис 1.1 — Звуковой сигнал на входе АЦП

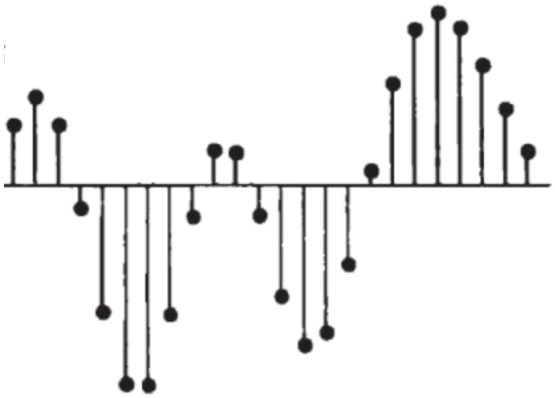

Рис 1.2 — Дискретный сигнал на выходе АЦП

Обратное преобразование для воспроизведения звука, закодированного числовым кодом, выполняют цифро-аналоговые преобразователи (ЦАП).

На рисунках 1.3 и 1.4 представлено обратное преобразование дискретного сигнала в непрерывный звуковой сигнал.

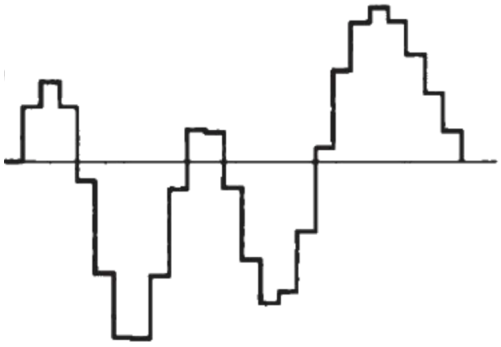

Рис 1.3 — Дискретный сигнал на входе ЦАП

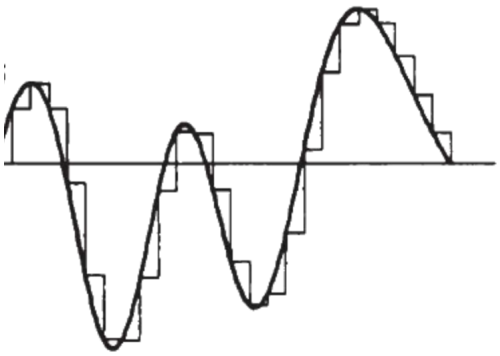

Рис 1.4 — Звуковой сигнал на выходе ЦАП

Недостатком данного метода кодирования является то, что синтезированные звуки получаются не слишком похожими на звучание реальных инструментов или других источников закодированного звука.

Таблично-волновой метод (Wave-Table) основан на том, что в заранее подготовленных таблицах хранятся образцы звуков окружающего мира, музыкальных инструментов и так далее. Числовые коды выражают высоту тона, продолжительность и интенсивность звука и прочие параметры, характеризующие особенности звука. Поскольку в качестве образцов используются «реальные» звуки, качество звука, полученного в результате синтеза, получается очень высоким и приближается к качеству звучания реальных музыкальных инструментов [9].

1.1.4 Хранение аудиоинформации на компьютере

Для хранения звуковых данных на компьютере и звуковых носителях в аудиофайлах используются цифровые аудиоформаты. При этом звуковые данные могут быть сжаты для устранения избыточности с помощью устройства или программы, называемой аудиокодеком. Выделяется три основных типа аудиоформатов:

— аудиоформаты без сжатия;

— аудиоформаты со сжатием без потерь;

— аудиоформаты со сжатием с потерями.

Некоторые примеры аудиоформатов представлены в таблицах 1 — 2 [10].

Таблица 1.1 — Примеры аудиоформатов (часть 1)

Расширение	Тип	Описание
.wav на Windows или .aiff на MAC	Без сжатия	Гибкий аудиоформат, разработанный для хранения любой комбинации частот дискретизации оригинальной звукозаписи. Аудиофайлы с таким форматом достигают большого размера
MP3	Сжатие с потерями	Самый популярный аудиоформат. Значительное уменьшение размера данных достигается за счет потери качества воспроизведения на профессиональных звуковых системах
FLAK	Сжатие без потерь	Не нарушает целостность данных, при этом обладая относительной быстротой кодирования/декодирования, гибкостью и средней степенью сжатия

Таблица 1.2 — Примеры аудиоформатов (часть 2)

Расширение	Тип	Описание
AAC	Сжатие с потерями	Преемник MP3 формата с более эффективной компрессией
WMA	Сжатие с потерями	Разработан под ОС Windows. Полностью поддерживается Windows, однако при низкой степени сжатия потока качество звука заметно снижается
TAK	Сжатие без потерь	При отсутствии потерь в качестве отличается высокой скоростью кодирования/декодирования и степенью сжатия
RAW	Без сжатия	Содержит необработанную информацию для обеспечения целостности данных. Не имеет четкой спецификации

1.1.5 MIDI-файлы

Отдельно от приведенной в таблицах 1.1 – 1.2 классификации рассматриваются аудиоформаты, содержащие не только оцифрованный звук или не содержащие его вовсе, такие как **MIDI** формат [11], хранящий наборы инструкций для воспроизведения звука. Аббревиатура MIDI расшифровывается как Musical Instrument Digital Interface (Цифровой интерфейс музыкальных

инструментов). Этот формат представляет собой интерфейс музыкальных инструментов, с помощью которого можно кодировать нажатие клавиш, проигрываемые ноты, ссылки на инструменты, значения изменяемых параметров звука и другие данные. Такой способ представления данных позволяет синхронизировать звуковое воспроизведение с управлением различным оборудованием. При этом последовательность инструкций, хранимая в MIDI-файле, только аппроксимирует нюансы исполнения, игнорируя, например, динамические изменения в продолжительности ноты. Главной целью разработки такого формата является компактное представление, которое подходит для хранения на дисковом пространстве, но может быть неудобно для хранения в памяти для быстрого доступа [12]. Файлы MIDI содержат один или несколько потоков MIDI с информацией о времени для каждого события. Поддерживаются структуры песен и дорожек, информация о темпе и тактовом размере. Названия дорожек и другая описательная информация могут храниться вместе с MIDI-данными. MIDI формат поддерживает несколько дорожек.

MIDI-файлы строятся из фрагментов. Каждый фрагмент имеет 4-символьный тип и 32-битную длину, которая представляет собой количество байтов в фрагменте. Каждый фрагмент начинается с 4-символьного ASCII типа. За ним следует 32-битная длина, старший байт идет первым (например, длина, равная 6 байтам, сохраняется, как 00 00 00 06). Эта длина относится к количеству байтов данных, которые следуют далее: восемь байтов, отведенных для типа и длины, не включены. Таким образом, фрагмент длиной 6 байтов фактически займет 14 байтов дискового пространства.

MIDI-файлы состоят из фрагментов 2-х типов: фрагмент заголовка и фрагмент дорожки. При этом такой файл всегда начинается с фрагмента заголовка, за которым следует один или несколько фрагментов дорожек.

На рисунке 1.5 представлена концепция структуры файла MIDI [12].

```
MThd  <length of header data>
<header data>
MTrk  <length of track data>
<track data>
MTrk  <length of track data>
<track data>
. . .
```

Рис 1.5 — Общая структура файла MIDI

Фрагмент заголовка предоставляет минимальный объем информации, относящейся ко всему MIDI-файлу и состоит из следующих элементов.

а) chunk type – 4-х символьное представление типа фрагмента (MThd).

б) length – 32-битная величина, значение которой описывает длину в байтах фрагмента без учета первых двух элементов (всегда 6).

в) format – формат MIDI-файла (2 байта).

1) 0 – файл содержит одну единственную дорожку, содержащую MIDI-данные, возможно, для всех 16 каналов. Используется, если секвенсор хранит все MIDI-данные в одном блоке памяти в том порядке, в котором они воспроизводятся.

2) 1 – файл содержит одну или несколько одновременных (то есть все начинаются с предполагаемого времени 0) дорожек, возможно, каждая на одном канале. Вместе эти треки считаются одной секвенцией или паттерном. Используется, если секвенсор разделяет дорожки на разные блоки памяти, но воспроизводит их одновременно.

3) 2 – файл содержит один или несколько последовательных независимых паттернов, состоящих из одной дорожки. Используется, если секвенсор разделяет дорожки на разные блоки памяти, но воспроизводит только один блок за раз (каждый блок считается отдельным «отрывком» или «песней»).

г) ntrks – количество дорожек.

д) division – разрешение, указвающее, как MIDI-тики должны быть переведены во время.

Фрагмент дорожки хранит саму музыкальную информацию и представляет собой поток MIDI-событий (и не MIDI-событий), которому предшествуют значения дельта-времени. Каждый такой фрагмент состоит из следуюших элементов.

а) chunk type – 4-х символьное представление типа фрагмента (MTrk).

б) length – 32-битная величина, значение которой описывает длину в байтах фрагмента без учета первых двух элементов.

в) MTrk event – MIDI- или не MIDI-событие (может быть одно или несколько).

В свою очередь, элемент MTrk event представляется следующими параметрами.

а) delta-time – количество времени до следующего события (Если первое событие на дорожке происходит в самом начале дорожки, или если два события происходят одновременно, значение равно 0).

б) event – либо MIDI-событие (например, такого содержания: идентификатор события, информация о воспроизводимой ноте, ее высоте и положении в октаве и сила извлечения звука), либо не MIDI-событие (события, которые содержат такие данные, как настройки темпа, названия дорожек и тому подобное).

Благодаря такому формату, MIDI-файлы поддерживают хранение музыкальных партитур. Музыкальная партитура [13] – это двухмерное представление звуковой последовательности, составляющей музыкальное произведение. Два основных параметра в таком представлении – это музыкальные ноты, которые должен играть каждый инструмент, и момент, когда звучит каждая нота. Полнофункциональная партитура представляет другие параметры, потому что в партитуру помещается много других элементов, таких как интенсивность каждой ноты и ее изгиб. Тем не менее, два основных параметра достаточны для представления музыки. Таким образом, музыкальная парти-

тура является инструкцией для воспроизведения звука и может храниться в MIDI аудиоформате. Причем в MIDI аудио-файле содержится только одна характеристика: последовательность нот, играемых каждым инструментом, — где ноты имеют начальное сообщение, указывающее на канал (1-16), номер высоты тона в полутонах (1-127), динамику (1-127) и сообщение, свидетельствующее об окончании ноты [11].

Таким образом, MIDI-файлы могут содержать разнообразную информцию о музыке, не ограничиваясь только оцифрованным звуком. При этом формат MIDI разработан так, что даже подробная информация, включающая, например, метаданные и сложные инструкции для воспроизведения нот через различное музыкальное оборудование, имеет компактное представление. Учитывая эти достоинства, дальнейшее рассмотрение аудио-файлов в данной работе ограничивается MIDI форматом, для структуры которого разрабатывается метод распределенного хранения в NoSQL-базе данных.

1.2 Существующие решения

1.2.1 Хранение MIDI-файлов в реляционной базе данных PostgreSQL в виде блоба

PostgreSQL — свободная объектно-реляционная система управления базами данных [14]. PostgreSQL предоставляет средства работы с медиафайлами, рассматривающие такие файлы, как большие объекты, или, иначе говоря, блобы (специальный тип данных, предназначенный, в первую очередь, для хранения аудиофайлов и изображений, а также компилированного программного кода) [15]. Механизм больших объектов разбивает данные большого объема на «фрагменты» и сохраняет эти фрагменты в строках таблицы. При произвольном доступе на запись и чтение быстрый поиск нужного фрагмента обеспечивается индексом-B-деревом в этой таблице. Аудиофайлы, записанные в базу данных таким способом, представляются потоком байтов, разбитым на сегменты, каждый из которых не превышает величину LOBLKSIZE (2,05 Кб) для удобного размещения в отдельных строках таблицы [16]. Потоковый доступ удобен в обработке данных больших объемов, которыми неудобно или невозможно оперировать как одним целым. К тому же это расширяет возможности доступа к отдельным частям данным, что обуславливается механизмом работы со структурой большого объекта: так как данные разбиты на фрагменты, каждый из которых представляет собой кортеж, можно легко получить доступ к любому такому фрагменту без необходимости считывать весь объект. Однако такой подход не позволяет извлечь конкретную часть данных, если не знать структуру хранимого объекта, а именно какой набор байт какой части данных соответствует. Это приводит к тому, что такой способ хранения не подойдет, в случае если необходимо не только хранить и извлекать аудиофайл из базы данных целиком, но и изменять отдельные его части.

На рисунке 1.6 представлена концептуальня схема хранения MIDI-файлов в базе данных в виде блоба [17].

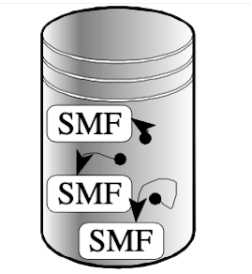

Рис 1.6 — Хранение MIDI-файлов в базе данных в виде блоба

В PostgreSQL большие объекты хранятся в таблице pg_largeobject, имеющей следующую структуру (таблица 1.3) [14].

Таблица 1.3 — Структура таблицы pg_largeobject

Имя	Тип	Описание
loid	oid	Идентификатор большого объекта, включающего эту страницу
pageno	int4	Номер этой страницы в большом объекте (начиная с 0)
data	bytea	Данные, хранящиеся в большом объекте

Для добавления в базу данных большого объекта, сформированного из аудио-файла, существует серверная функция lo_import(), принимающая путь к файлу на диске. Возвращаемым значением этой функции является идентификатор созданного большого объекта (тип данных oid). Пример добавления содержимого аудио-файла в базу данных PostgreSQL при наличии созданной заранее таблицы audio с атрибутом bigObject_Id типа oid представлен на листинге 1.1.

Листинг 1.1 — Пример добавления аудио-файла в базу данных PostgreSQL в виде блоба

```
1 INSERT INTO audio (bigObjectId)
2     VALUES
          (lo_import("/Users/anastasia/Desktop/ProgramEngineering/audio.mid"));
```

В результате, считанные из файла байты, будут добавлены в таблицу pg_largeobject в виде фрагментов большого объекта. Однако, при таком способе хранения аудио-файла для систем управления базами данных (СУБД) он представляется, как аморфный набор данных, так как СУБД не знает внутренней структуры блоба. Таким образом, операции доступа к конкретным данным MIDI-файла оказываются невозможны.

1.2.2 Хранение путей к MIDI-файлу в файловой системе

Одним из возможных решений хранения аудио-файла в любой базе данных является хранение в базе данных пути к файлу на диске.

На рисунке 1.7 представлена концептуальня схема хранения аудио-файлов в базе данных с использованием пути к файлу [17].

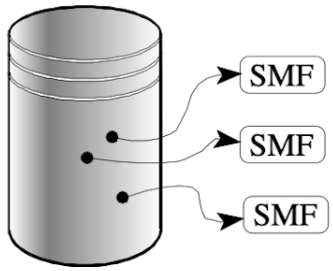

Рис 1.7 — Хранение в базе данных путей к MIDI-файлам, расположенным на дисковом пространстве

Данный подход не выгоден, так как ответственность за хранилище и расположение файлов в нем полностью делегируется на операционную систему, из-за чего СУБД не может гарантировать целостность набора указателей, а также вся дальнейшая работа с MIDI-файлом предоставляется клиенту, что значительно снижает роль базы данных при таком методе хранения.

1.2.3 Хранение MIDI-файла в виде структуры

Спецификация MIDI-файлов [12] позволяет определить внутреннюю структуру аудио-файла, в которой его можно было бы хранить в базе данных. Такой подход удовлетворяет требованию распределенности и возлагает значительную степень ответственности за работу с аудио-информацией на базу данных, освобождая клиента от необходимости использовать дополнительные инструменты, в случае если задача заключается в доступе к определенным элементам структуры MIDI-файла.

На рисунке 1.8 представлена концептуальня схема хранения аудио-файлов в базе данных на основе их структуры [17].

Рис 1.8 — Хранение MIDI-файлов в базе данных на основе их структуры

Предварительный анализ исходного MIDI-файла в том виде, в котором он хранится на диске, позволит СУБД впоследствии работать со структурированными данными MIDI-файла и, соответственно, отвечать на запросы на основе содержимого хранимой информации, что является наиболее выгодным решением.

1.3 Виды существующих NoSQL баз данных

NoSQL база данных [18] — это база данных, которая не использует реляционную модель данных, а именно организацию хранения данных в виде таблиц и отношений между ними. Вместо этого такие базы ориентированы на какой-то определенный тип данных, под требования хранения которого оптимизируется система хранения таким образом, чтобы процесс запроса был наиболее подходящим для выбранного типа данных. Это позволяет хранить

любые данные в NoSQL базах нерегламентированно (гибкость модели данных), а также размещать данные большого объема с относительно быстрым доступом к ним за счет масштабируемости. Главное правило проектирования структуры данных в NoSQL базах — она должна подчиняться требованиям приложения и быть максимально оптимизированной под наиболее частые запросы. Термин "NoSQL"расшифровывается как "Not Only SQL"и, подразумевая под собой ответвление SQL-подхода к проектированию баз данных, является собирательным определением альтернатив к организации хранения и доступа к данным средствами SQL.

По модели данных NoSQL-базы данных классифицируются на следующие основные виды.

а) Ключ — значение.

б) Документы.

в) Семейства столбцов.

г) Графы.

В таблицах 1.4 — 1.6 представлена сравнительная характеристика баз данных SQL и NoSQL [19].

Таблица 1.4 — Сравнительная характеристика баз данных SQL и NoSQL (часть 1)

Критерий/Тип БД	Реляционные	NoSQL
Модель данных	Используют реляционную модель, которая преобразует данные в двумерные таблицы, состоящие из строк и столбцов	Используют разнообразные модели данных, такие как пары «ключ-значение», документы и графы, оптимизированные под определенный тип хранимых данных для высокой производительности и горизонтальной масштабируемости
Схема данных	Схема жестко задает таблицы, строки, столбцы, индексы, отношения между таблицами и прочие элементы базы данных. Такая БД обеспечивает целостность ссылочных данных в отношениях между таблицами.	Гибкая схема

Таблица 1.5 — Сравнительная характеристика баз данных SQL и NoSQL (часть 2)

Критерий/Тип БД	Реляционные	NoSQL
Выполнение запросов	Запросы составляются на языке SQL. Эти запросы анализирует и выполняет реляционная база данных	Для записи/извлечения могут использоваться объектно-ориентированные API. Поиск может осуществляться по парам «ключ-значение», наборам столбцов и т. д.
Масштабирование	Обычно масштабируются путем увеличения вычислительных возможностей аппаратного обеспечения или добавления отдельных копий для рабочих нагрузок чтения (преимущественно вертикальное масштабирование)	Обычно поддерживают высокую разделяемость благодаря шаблонам доступа с возможностью масштабирования на основе распределенной архитектуры. Это повышает пропускную способность и обеспечивает устойчивую производительность почти в неограниченных масштабах (горизонтальное масштабирование)

Таблица 1.6 — Сравнительная характеристика баз данных SQL и NoSQL (часть 3)

Критерий/Тип БД	Реляционные	NoSQL
Производительность	Производительность главным образом зависит от дисковой подсистемы. Для обеспечения максимальной производительности часто требуется оптимизация запросов, индексов и структуры таблицы.	Производительность обычно зависит от размера кластера базового аппаратного обеспечения, задержки сети и вызывающего приложения

1.3.1 Модель данных Ключ-Значение

Модель данных, реализующяя хранение по типу ключ-значение связывает каждое значение данных с уникальным ключом. Большинство СУБД с такой моделью данных поддерживают только самые простые операции запроса, вставки и удаления, а также, как правило, используют хеш-таблицу, в которой находится уникальный ключ и указатель на конкретный объект данных. Производительность сильно вырастает за счёт кеширующих механизмов. Как ключи, так и значения могут представлять собой что угодно: от простых до сложных составных объектов. Базы данных с использованием пар ключ-значение поддерживают высокую разделяемость и обеспечивают беспрецедентное горизонтальное масштабирование, недостижимое при использовании других типов баз данных. Однако, чтобы частично или полностью изменить значение, приложение всегда перезаписывает существующее значение целиком. Также СУБД, имеющие такую модель данных, не оптимизированы для запросов по значению.

1.3.2 Модель данных документов

Документоориентированная модель данных хранит данные, представленные парами ключ-значение, которые сжимаются в виде полуструктурированного документа из тегированных элементов, подобно JSON, XML, BSON и другим подобным форматам. В базе данных документов хранится коллекция документов, в которой каждый документ состоит из именованных полей и данных. Данные могут быть простыми значениями или сложными элементами, такими как списки и дочерние коллекции. Документы извлекаются по уникальным ключам.

Одним из ключевых различий между моделью данных по типу ключ-значение и документоориентированным хранением является то, что последнее включает метаданные, связанные с хранимым содержимым, что даёт возможность делать запросы на основе содержимого. Тот факт, что такие базы данных работают без схемы, делает простой задачей добавление полей в документы без необходимости сначала заявлять об изменениях.

Документные базы данных позволяют разработчикам хранить и запрашивать данные в БД с помощью той же документной модели, которую они используют в коде приложения. Гибкий, полуструктурированный, иерархический характер документов и документных баз данных позволяет им развиваться в соответствии с потребностями приложений.Документные базы данных обеспечивают гибкость индексации, производительность выполнения стандартных запросов и аналитику наборов документов.

1.3.3 Модель данных семейства столбцов

В колоночных NoSQL-базах данных данные хранятся в ячейках, сгруппированных в колонки, а не в строки данных. Колонки логически группируются в колоночные семейства. Колоночные семейства могут состоять из практически неограниченного количества колонок, которые могут создаваться во время работы программы или во время определения схемы. Чтение и запись происходит с использованием колонок, а не строк.

В сравнении с хранением данных в строках, как в большинстве реляционных баз данных, преимущества хранения в колонках заключаются в быстром поиске/доступе и агрегации данных. Реляционные базы данных хранят каждую строку как непрерывную запись на диске. Разные строки хранятся в разных местах на диске, в то время как колоночные базы данных хранят все ячейки, относящиеся к колонке, как непрерывную запись, что делает операции поиска/доступа быстрее.

В отличие от модели данных на основе пар ключ-значение и модели данных документов, большинство хранилищ столбцов используют для упорядоченного хранения данных сами значения ключей, а не хеш-коды от них. Многие реализации позволяют создавать индексы по определенным столбцам в семействе столбцов. Индексы позволяют получать данные по значениям столбцов, а не ключам строки.

1.3.4 Модель данных графов

В графовой базе данных нет строгого формата SQL или представления таблиц и колонок, вместо этого используется гибкое графическое представление, которое подходит для решения проблем масштабируемости. База данных графов хранит сведения двух типов: узлы и грани. Ребра задают связи между узлами. Узлы и ребра могут иметь свойства, предоставляющие сведения об этом узле или ребра, аналогичные столбцам в таблице. Грани могут иметь направление, указывающее на характер связи. Обход графа в графовой базе данных можно выполнять либо по определенным типам ребер, либо по всему графу. Обход соединений или взаимосвязей в графовых базах данных выполняется очень быстро, поскольку взаимосвязи между узлами не вычисляются во время выполнения запроса, а хранятся в базе данных. Тем не менее, модель данных графов отличается высокой сложностью реализации.

1.3.5 MongoDB

MongoDB [20] – система управления базами данных, которая работает с документоориентированной моделью данных и для хранения данных использует формат, подобный JSON [21], – формат BSON [22]. Данный формат представляет собой двоично-кодированную сериализацию JSON-подобных документов. Как и JSON, BSON поддерживает встраивание документов и массивов в другие документы и массивы. BSON также содержит расширения, позволяющие представлять типы данных, не входящие в спецификацию JSON. Например, BSON имеет тип BinData [23].

Данные в BSON-формате хранятся в документах, которые хранятся в коллекции. Это означает, что структура сохраняемых данных может быть изменена для каждой записи и не требуется, чтобы она соответствовала предварительно определенному шаблону. Хранилище в BSON-формате также позволяет вкладывать данные, благодаря чему можно хранить сложноорганизованные данные с любой структурой. MongoDB поддерживает поиск по полям, диапазонные запросы и поиск по регулярным выражениям. Могут быть сделаны запросы для возврата определенных полей в документах. Также

MongoDB использует концепцию шардинга для горизонтального масштабирования с помощью разделения данных между несколькими экземплярами БД. Она может работать на нескольких серверах, балансируя нагрузку и/или дублируя данные, чтобы поддерживать работоспособность системы в случае аппаратного сбоя.

Терминология MongoDB отличается от терминологии SQL. Основные понятия баз данных сопоставлены в таблице 1.7.

Таблица 1.7 — Сравнение терминологии MongoDB и терминологии SQL

MongoDB	SQL
Коллекция	Таблица
Документ	Ряд
Поле	Столбец
ObjectId	Первичный ключ

1.4 Выбор NoSQL-базы данных

В качестве NoSQL-базы данных, наиболее подходящей под хранение аудио-файлов формата MIDI на основе их структуры, была выбрана база данных MongoDB, так как она является свободной и одной из самых распространенных документоориентированных баз данных. Это означает, что структура сохраняемых данных может быть изменена для каждой записи в базу и не требует, чтобы она была в предварительно определенном формате. Хранилище в формате BSON, отличающемся гибкой структурой, высокой скоростью сканирования и кодирования и декодирования данных, а также наличием типов данных, не поддерживаемых форматом JSON, позволяет вкладывать данные, что особенно важно для хранения аудио-файлов в MIDI формате, содержащих произвольное количество музыкальных дорожек. Возможность хранения разных типов данных в непредопределенной стрктуре позволяет хранить музыкальные данные в любом формате, адаптируемом под бизнес-требования. В частности, такой тип, как "бинарные данные"(BinData),

очень удобен для представления некоторых элементов MIDI-файлов, например, для массива байтов, отвечающих за MIDI- и не Midi-события во фрагменте музыкальной дорожки. Помимо этого, MongoDB предоставляет возможности репликации и ограниченную поддержку шардинга, что обеспечивает высокие доступность и надежность и балансировку нагрузки. Также у MongoDB есть клиентские библиотеки для основных языков программирования, таких как C [24], C++ [25], C# [26], Python [27], Java [28] и других. Открытый исходный код предоставляет возможность изучить внутреннее устройство СУБД и разработать программное обеспечение, которое впоследствии может быть интегрировано в систему.

1.4.1 Выводы из аналитического раздела

В данном разделе были приведены теоретические сведения о звуковой информации, об аудио-форматах и о существующих NoSQL-базах данных, а также были рассмотрены способы хранения аудио-файлов в базе данных. Было выяснено, что наиболее полноценную и разнородную информацию содержат аудио-файлы формата MIDI, которые наиболее оптимально хранить на основе их внутренней структуры. Также была приведена сравнительная характеристика реляционных и NoSQL-баз данных и сравнительная характеристика видов нереляционных баз данных по модели данных. На основе полученного сравнительного анализа, было установлено, что наиболее подходящим решением для хранения MIDI-файлов на основе их структуры является хранение в документоориентированной NoSQL-базе данных MongoDB.

2 Конструкторский раздел

2.1 Основные операции для работы с данными

Разработанный метод распределенного хранения аудио-файлов в NoSQL-базе данных применяется на этапах добавления и извлечения аудио-файла из базы данных.

2.1.1 Добавление аудио-файла в MongoDB

В процессе обработки операции записи данных в базу данных выполняется валидация глубины документа и проверка на наличие заданного клиентом идентификатора, а такаже присвоение идентификатора, в случае если его нет. Разработанный метод применяется на этом же уровне и состоит из следующих шагов.

а) Прверка на то, что данные являются MIDI-файлом (данные должны соответствовать определенной структуре, а также аудио-файл должен существовать и являться MIDI-файлом).

б) Извлечение пути к аудио-файлу и его имени для дальнейшего извлечения

в) Считывание содержимого аудио-файла.

г) Создание документа со специальной структурой (Parser).

На рисунке 2.1 представлена функциональная декомпозиция этапов добавления аудио-файла в MongoDB.

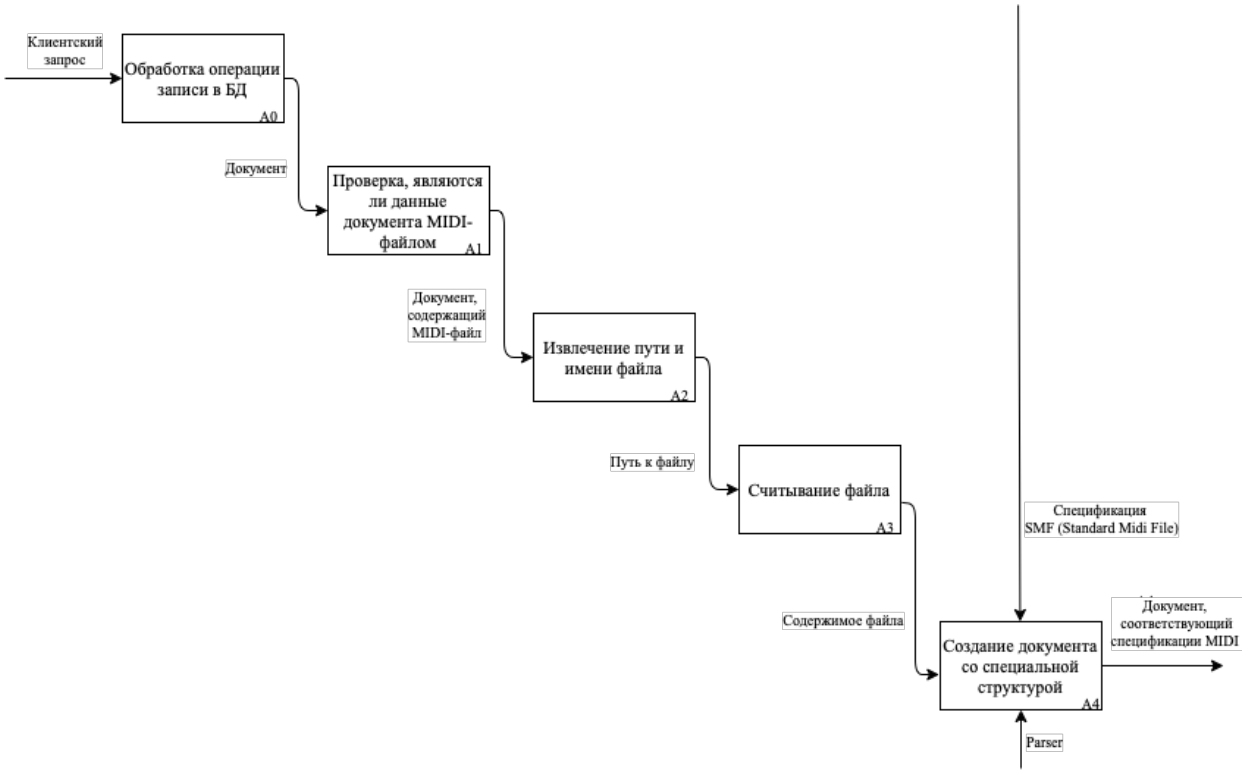

Рис 2.1 — Функциональная декомпозиция этапов добавления аудио-файла в MongoDB

Формирование специальной структуры, соответствующей спецификации MIDI-файла, выполняется с помощью метода, представленного на листингах 4.7 – 4.9. Так как данные в MongoDB хранятся в виде объектов типа BsonDocument, весь MIDI-файл представлен в виде такого документа, где каждое поле является либо объектом типа BsonElement, либо, если имеет несколько полей, подобъектом типа BsonDocument, либо массивом подобъектов.

Тип BsonElement имеют следующие элементы MIDI-файла.

а) ID – идентификатор заголовка MIDI (MThd) или музыкальной дорожки (MTrk). Тип данных: строка.

б) Length – длина фрагмента заголовка или фрагмента музыкальной дорожки. Тип данных: бинарные данные.

в) Format – формат MIDI-файла. Тип данных: бинарные данные.

г) NumTracks – количество музыкальных дорожек. Тип данных: бинарные данные.

д) Devision – разрешение MIDI-файла. Тип данных: бинарные данные.

е) Data – данные музыкальной дорожки. Тип данных: бинарные данные.

Также структура документа MongoDB, содержащего аудио-файл, была дополнена еще одним объектом типа BsonElement – Name – имя MIDI-файла, используемое при извлечении из базы данныхс опцией сохранения на диске. Тип данных: строка.

Тип BsonArray имеет поле «MTrks», содержащее список объектов BsonDocument (каждый объект представляет одну музыкальную дорожку).

На рисунках 2.2 – 2.5 представлена схема метода создания документа MongoDB по спецификации MIDI.

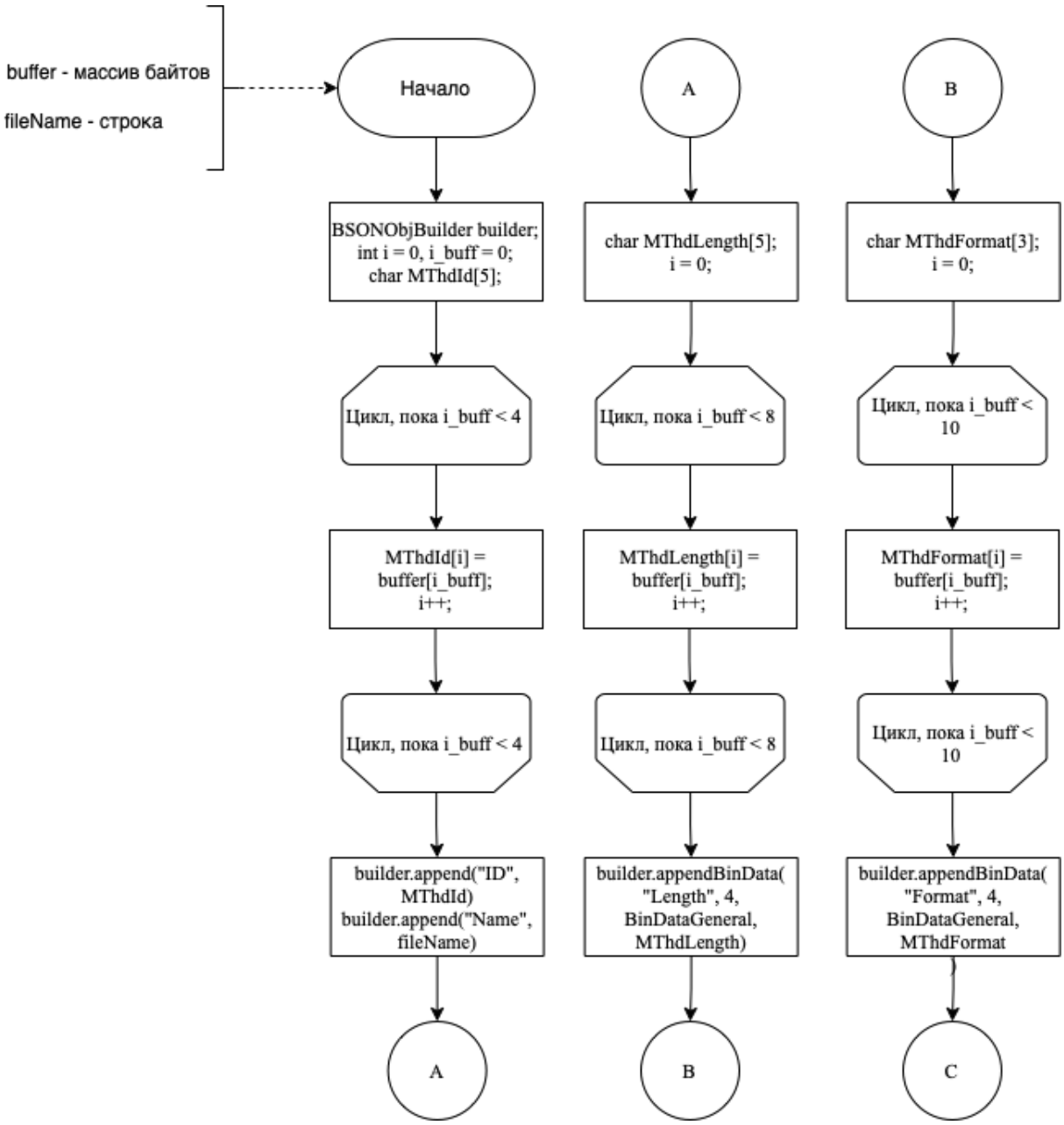

Рис 2.2 — Схема метода создания документа MongoDB по спецификации MIDI (часть 1)

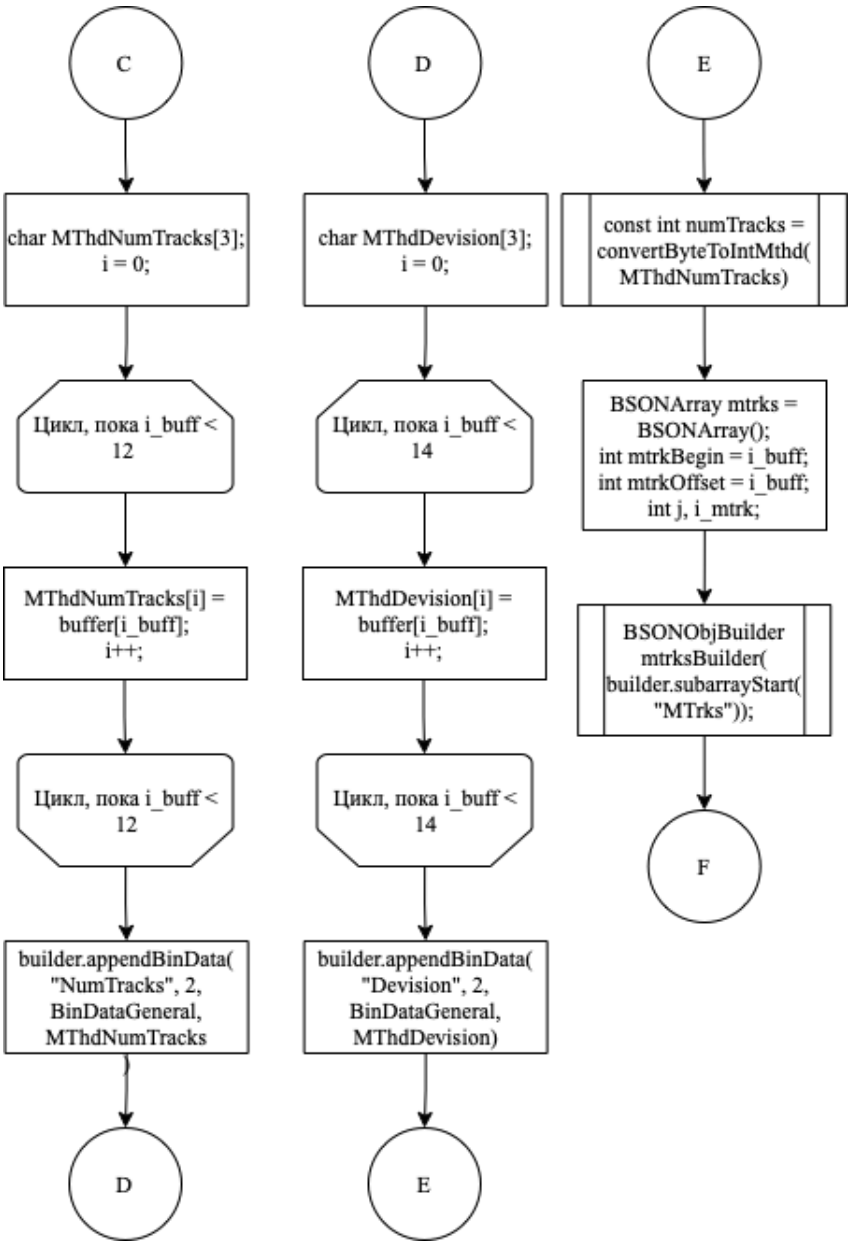

Рис 2.3 — Схема метода создания документа MongoDB по спецификации MIDI (часть 2)

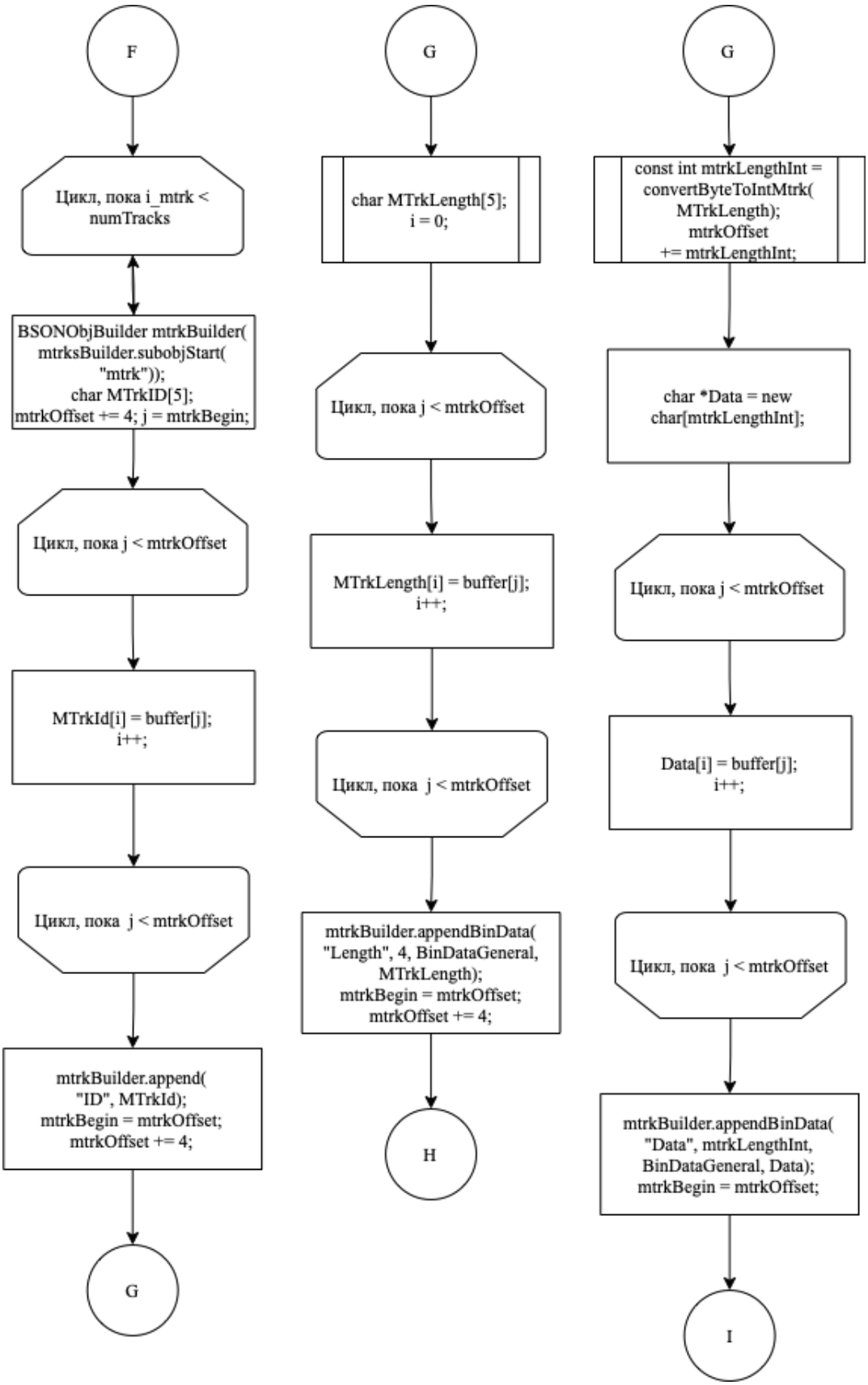

Рис 2.4 — Схема метода создания документа MongoDB по спецификации MIDI (часть 3)

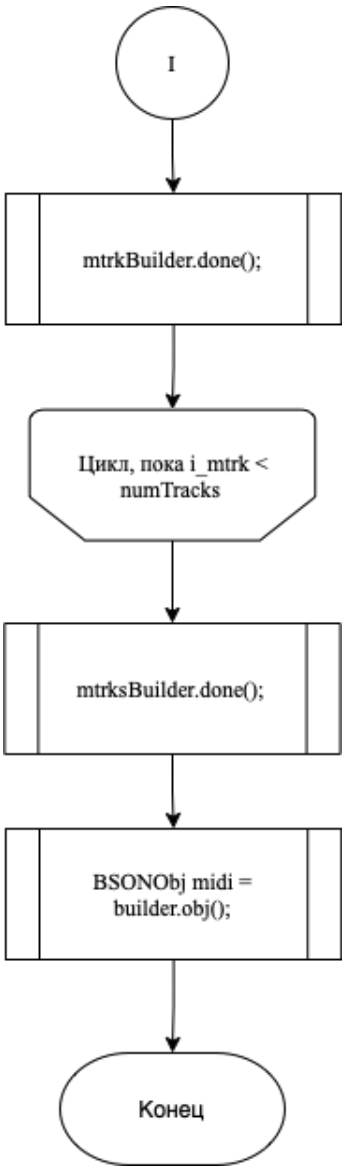

Рис 2.5 — Схема метода создания документа MongoDB по спецификации MIDI (часть 4)

2.1.2 Извлечение аудио-файла из MongoDB

В ходе выполнения операции извлечения документа или набора документов из базы данных записи, содержащие MIDI-информацию, обрабатываются особым образом. Извлечение документа, имеющего структуру MIDI-файла, дополняется опцией сохранения данных локально в виде аудио-файла. После того как к запросу на извлечение данных применяется парсинг системы, извлекающий данные запроса, выполняется извлечение фильтра запроса и проверяется наличие в нем поля MidiSave, являющегося флагом, сигнализирующим о желании пользователя сохранить струкутуру MIDI-файла локально

в формате аудио-файла с расширением «.mid». Затем, когда запрос будет выполнен, в случае если в фильтре запроса есть поле MidiSave и оно имеет значение True, сохранение метода осуществится в три этапа.

а) Прверка на то, что данные являются MIDI-файлом (данные должны соответствовать определенной структуре).

б) Формирование массива байтов из документа.

в) Запись массива байтов в файл.

На рисунке 2.6 представлена функциональная декомпозиция этапов извлечения аудио-файла из MongoDB (если документ MongoDB необходимо сохранить в виде аудио-файла).

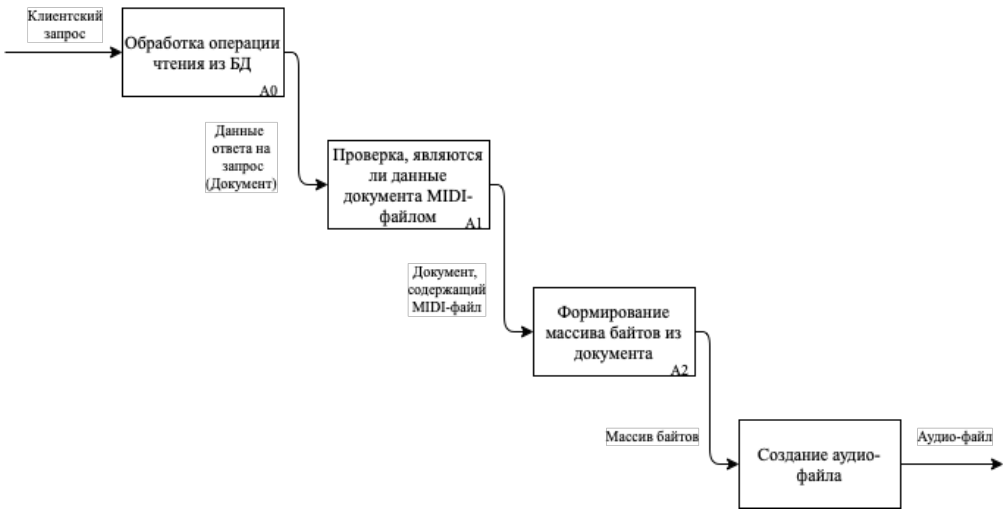

Рис 2.6 — Функциональная декомпозиция этапов извлечения аудио-файла из MongoDB (в случае записи документа MongoDB на диск в виде аудио-файла)

Если в фильтре запроса нет поля MidiSave или оно имеет значение False, документ MongoDB не будет представлен в виде MIDI-файла и будет только возвращен клиенту в стандратном виде BsonDocument.

На рисунках 2.7 – 2.9 представлена схема метода воссоздания MIDI-файла из документа MongoDB (листинги 4.10 – 4.11).

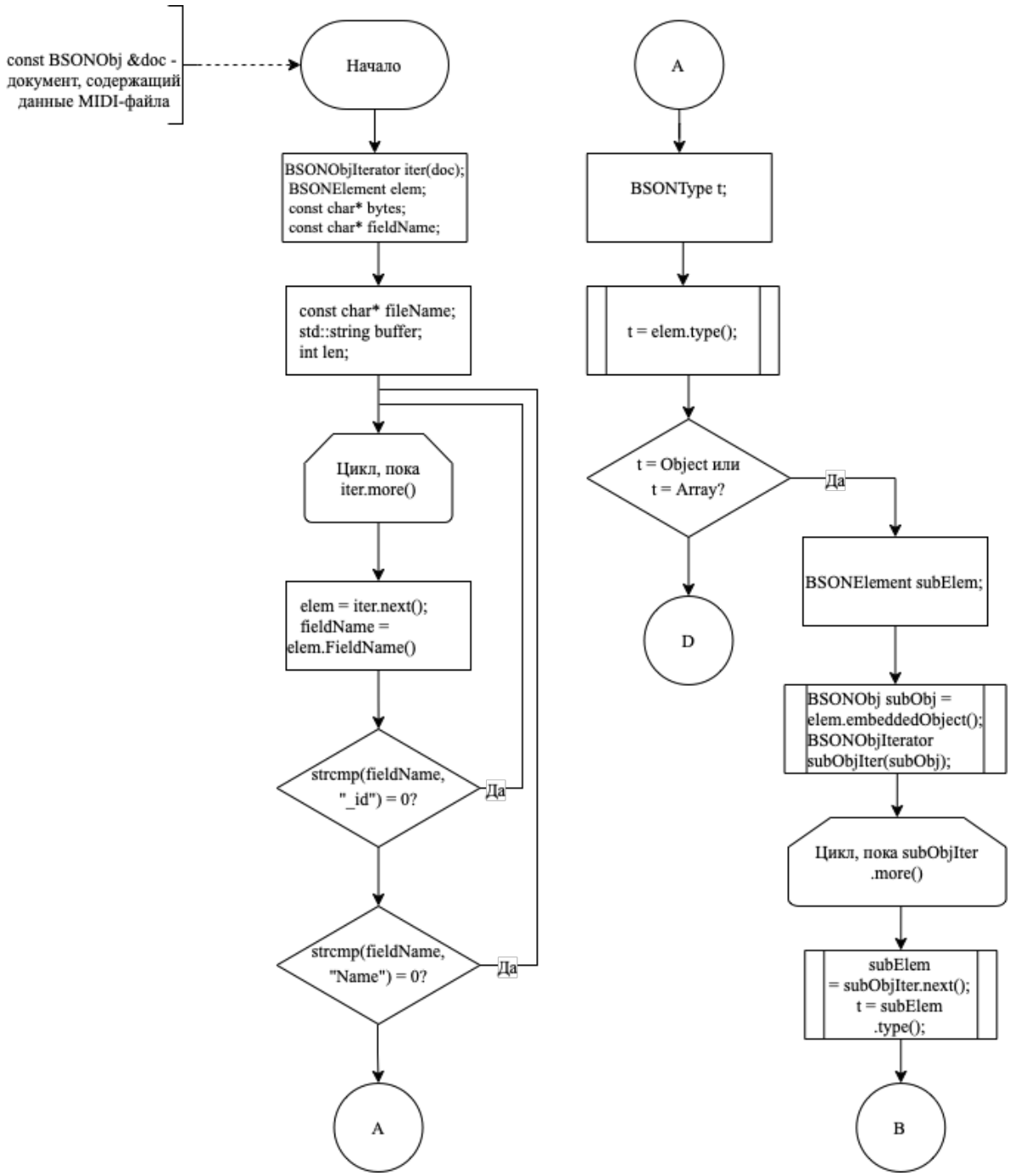

Рис 2.7 — Схема метода воссоздания MIDI-файла из документа MongoDB (часть 1)

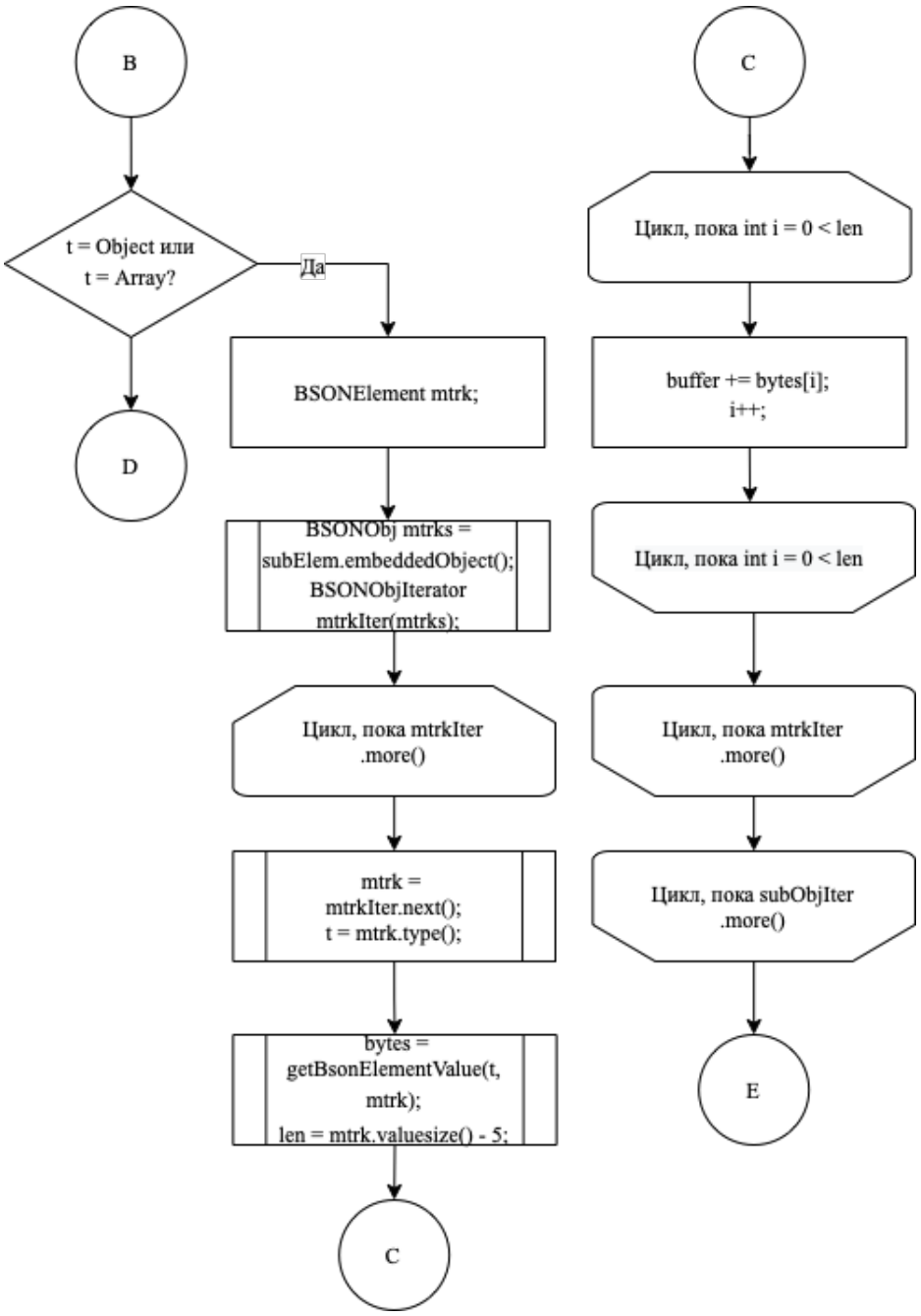

Рис 2.8 — Схема метода воссоздания MIDI-файла из документа MongoDB (часть 2)

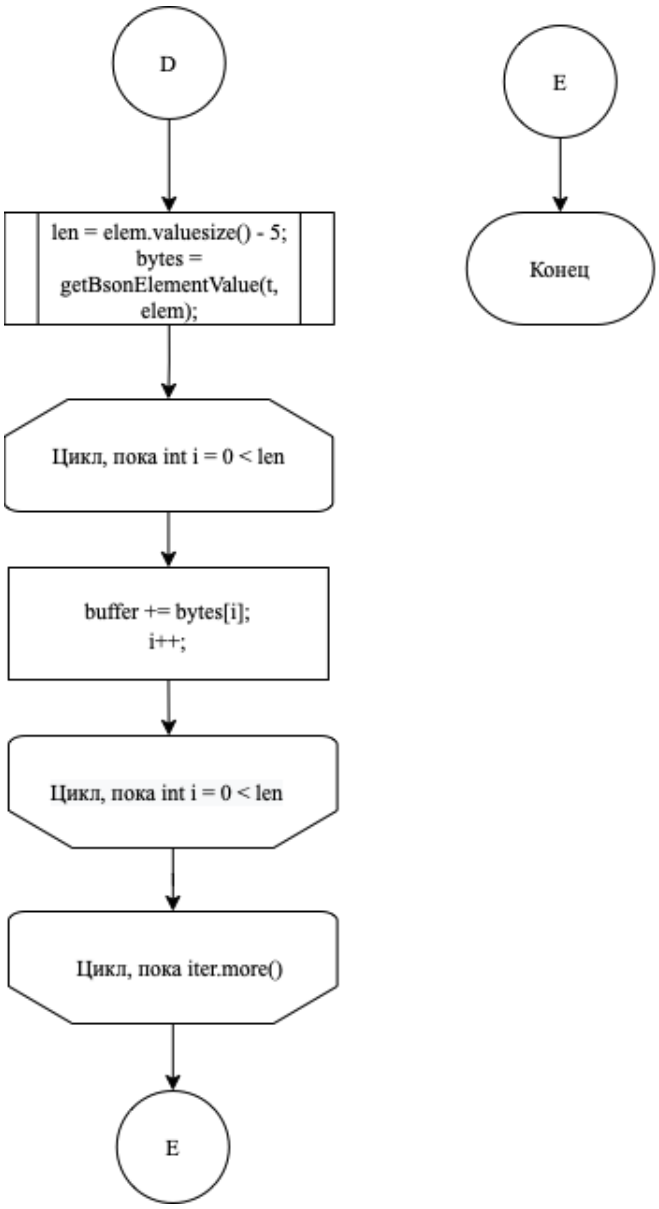

Рис 2.9 — Схема метода воссоздания MIDI-файла из документа MongoDB (часть 3)

2.2 Ограничения предметной области

Ограничения предметной области включают следующие пункты.

а) Добавление MIDI-файла в MongoDB и извлечение документа с его последующей записью на диск в формате MIDI-файла имеют жестко заданные API.

1) API записи аудио-файла в базу данных включает в себя все команды вставки, но имеет строгую структуру, представленную на листинге 2.1, где path – это путь к аудио-файлу, а name (может быть опущено) – имя

файла, используемое при сохранении документа в формате MIDI файла. Если эта структура не будет соблюдена, вставки аудио-файла в базу в предусмотренном в разработанном методе формате не произойдет.

2) API чтения аудио-файла из базы данных включает в себя все команды чтения, но для сохранения документа в виде аудио-файла необходимо задать поле MidiSave фильтра запроса, установив ему значение True.

б) Пользователь не имеет возможности задавать тип данных, в котором будут храниться поля аудио-файла в базе данных.

в) Данные, представляющие музыкальную дорожку, не разбиваются на логические подструктуры.

Листинг 2.1 — Структура запроса на добавление аудио-файла в MongoDB

```
1 {
2     path: string("some data"),
3     name: string("some data")
4 }
```

2.3 Выводы из конструкторского раздела

В данном разделе были описаны основные операции для работы с данными при использовании разработанного метода распределенного хранения аудио-файлов в NoSQL-базе данных, была проведена функциональная декомпозиция этих операций, были представлены схемы работы метода, реализующего создание документа MongoDB, соответствующего спецификации MIDI, из сырого содержимого аудио-файла. Также были определены ограничения предметной облвсти.

3 Технологический раздел

3.1 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.

3.2 Выбор среды разработки

Для программной реализации метода был выбран редактор кода Visual Studio Code [29], так как он обладает следующими достоинствами.

а) Кроссплатформенность.

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

в) Широкие возможности кастомизации.

г) Открытый исходный код.

3.3 Выбор инструментов для замеров времени

Для доступа к базе данных использовался драйвер C# для MongoDB – MongoDB.Driver [30], 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 [31], которая является кроссплатформенной, предоставляет удобные и быстрые редактор кода и отладчик, поддерживает платформу .Net [32] для создания приложений на языке C#, а также имеет свободный доступ для студентов. Установка драйвера была произведена с помощью пакетного менеджера NuGet [33], имеющего удобный интерфейс в среде Rider, и была интегрирована в сборку проекта с помощью

системы управления зависимостями .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 [34]. 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 [35]. 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

3.4 Сборка локального сервера MongoDB

Следуя инструкции по сборке MongoDB из исходного кода, расположенной в официальном репозитории базы данных [36], для компиляции сервера MongoDB использовался инструмент для автоматизации сборки программных проектов SCons [37]. Команда, использовавшаяся для запуска компиляции сервера MongoDB в командной строке с помощью SCons, представлена в листинге 3.1, 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 3.1 — 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 [38] или новее.

2) Clang 7.0 [39] или новее.

б) В операционных системах Linux [40] и macOS [41] требуется библиотека и заголовок libcurl [42]. MacOS включает libcurl.

в) Python 3.7 и некоторые модули, устанавливаемые с помощью команды, представленной в листинге 3.2.

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

Листинг 3.2 — Команда для установки необходимых для сборки модулей Python

```
1 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.

3.5 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.

3.6 Пример работы реализованного метода

Для демонстрации работы метода был создан клиент MongoDB на языке Python с помощью библиотеки pymongo [43]. 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 [44]. Также Python

содержит бибилиотеку tkinter [45], 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 3.3).

Листинг 3.3 — 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.

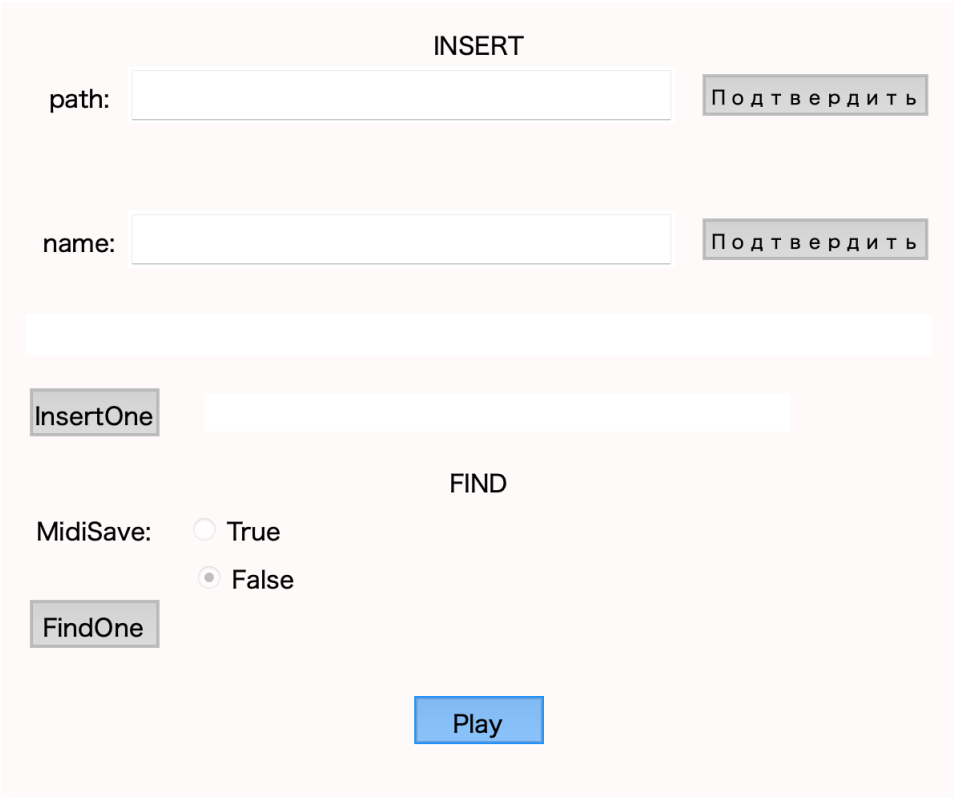

Рис 1.3 — Графический интерфейс для демонстрации работы метода

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).

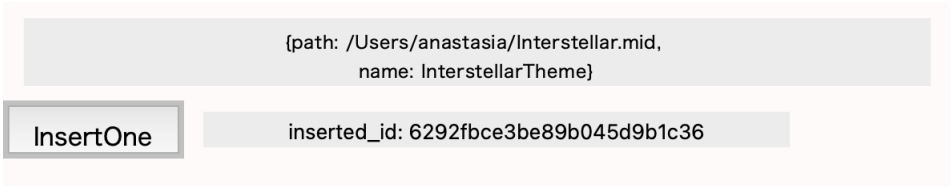

Рис 1.4 — Результат успешного добавления MIDI-файла в MongoDB

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 3.4), где midiFiles – это коллекция документов.

Листинг 3.4 — 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)

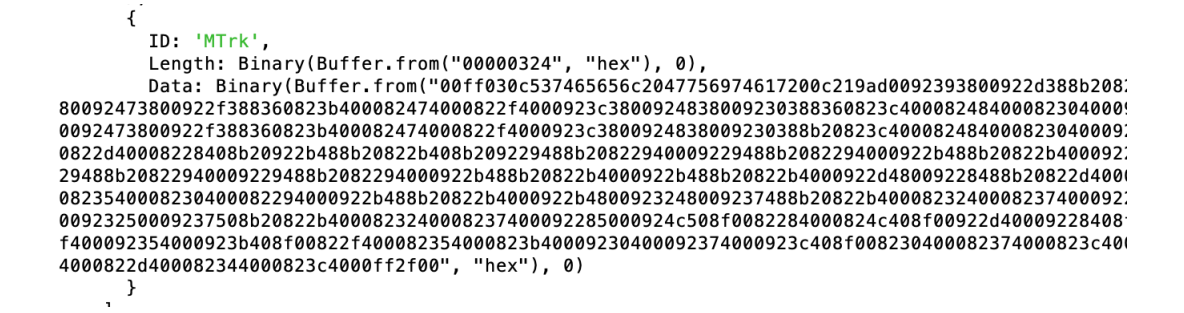

Рис 1.6 — Результат извлечения MIDI-файла из MongoDB (часть 2)

3.7 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.

4 Исследовательский раздел

4.1 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 4.1.

```
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 4.1 — Заголовок MIDI-файла, содержащего одну MTrk

Заголвок MIDI-файла с девятью дорожками представлен на рисунке 4.2.

```
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 4.2 — 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 4.3 приведен пример того, как в MongoDB будет представлен такой файл.

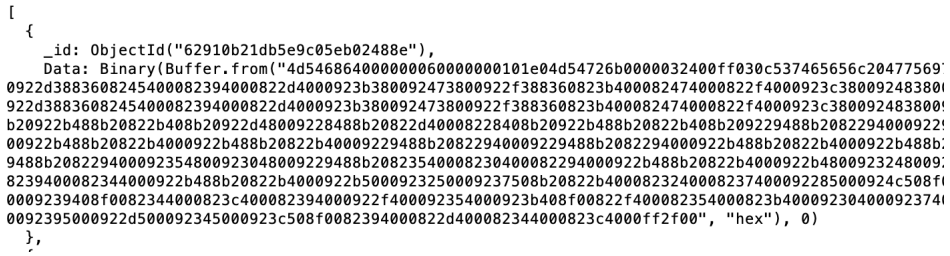

Рис 4.3 — Метод хранения MIDI-файла в MongoDB в виде массива байтов

4.2 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.

4.3 Результаты исследования

4.3.1 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 4.1). После инициализации документа созданный объект добавляется в базу данных.

Листинг 4.1 — 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 4.2).

Листинг 4.2 — 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 4.4 представлен результат работы методов хранения. Пояснения исследуемых характеристик приведены на рисунке 4.5.

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 4.4 — 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 4.5 — Сводная таблица работы методов при добавлении в MongoDB

На рисунке 4.6 полученные временные характеристики представлены в графическом виде.

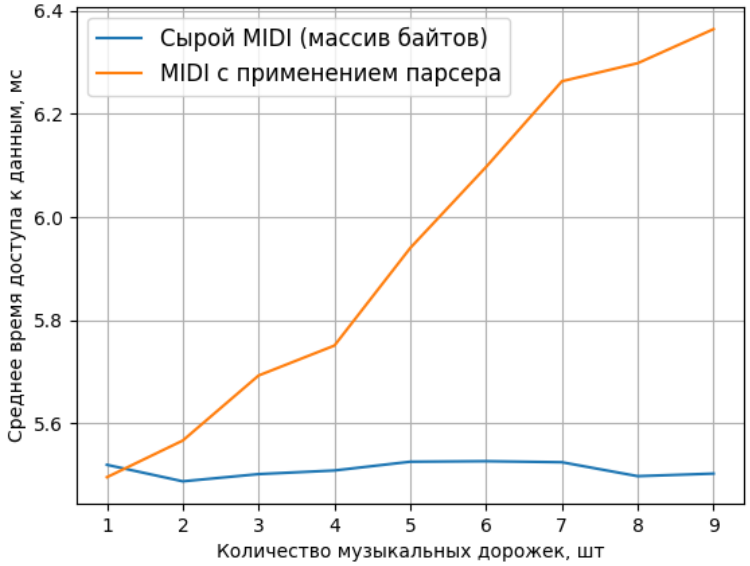

Рис 4.6 — Сравнение работы методов при добавлении в MongoDB

Как видно из рисунков 4.4 и 4.6, 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.

4.3.2 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 4.3).

Листинг 4.3 — 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 4.4).

Листинг 4.4 — 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 4.7 представлен результат работы методов хранения. Пояснения исследуемых характеристик приведены на рисунке 4.5.

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 4.7 — Сводная таблица работы методов при извлечении из MongoDB

На рисунке 4.8 полученные временные характеристики представлены в графическом виде.

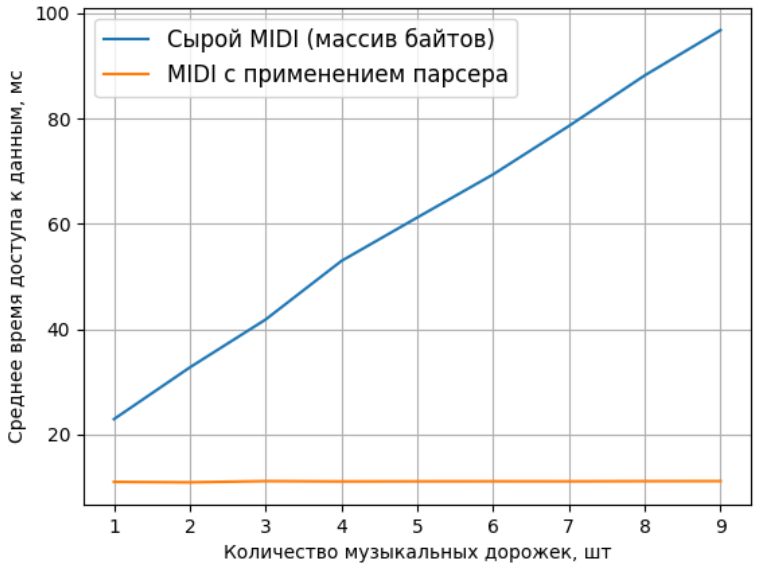

Рис 4.8 — Сравнение работы методов при извлечении из MongoDB

Из рисунков 4.7 и 4.8 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.

4.3.3 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 4.5.

Листинг 4.5 — 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 4.6.

Листинг 4.6 — 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 4.9 представлен результат работы методов хранения. Пояснения исследуемых характеристик приведены на рисунке 4.5.

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 4.9 — Сводная таблица работы методов при извлечении из MongoDB

На рисунке 4.10 полученные временные характеристики представлены в графическом виде.

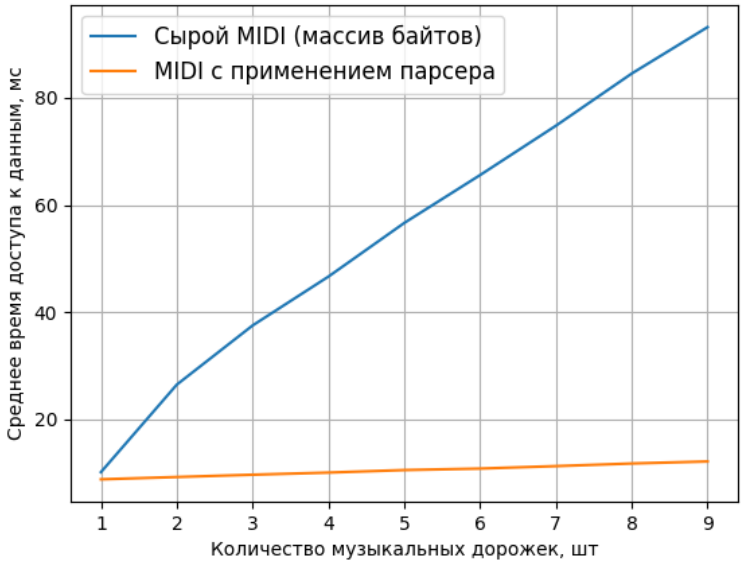

Рис 4.10 — Сравнение работы методов при добавлении и извлечении из MongoDB

Из рисунков 4.9 и 4.10 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.

4.4 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 работе проанализированы существующие аудио-форматы, выбран аудио-формат, дающий наиболее полную информацию о музыке, проанализированы существующие решения хранения такого формата в различных базах данных и выбран способ хранения, наиболее оптимальный для выбранного формата, проанализированы существующие модели данных и выбрана наиболее подходящая под выбранный способ хранения модель, а также выбрана база данных, которая ее использует. Разработано программное обеспечение, реализующее описанный метод. Произведено исследование времени работы метода и проанализированы полученные результаты.

В рамках работы были решены все поставленные задачи.

а) Рассмотрены способы представления звуковой информации и проанализированы существующие аудио-форматы.

б) Проанализированы способы хранения аудио-файлов в различных базах данных.

в) Разработан метод хранения аудио-файлов.

г) Разработано программное обеспечение, реализующее разработанный метод.

д) Исследована зависимость времени работы метода от количества дорожек в аудио-файле.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. What is Sound? [Электронный ресурс]. — Режим доступа: http://www.podcomplex.com/guide/physics.html (Дата обращения 2021-12-01).

2. Громкость и высота звука. Эхо [Электронный ресурс]. — Режим доступа: https://phscs.ru/physics8/volume-sound (Дата обращения 2021-12-10).

3. Ананьев Б. Г. Теория ощущений [Электронный ресурс]. — Режим доступа: http://elib.gnpbu.ru/text/ananyev_teoriya-oschuscheniy_1961/go,0;fs,1/ (Дата обращения 2021-12-20).

4. Frequency Range of Human Hearing [Электронный ресурс]. — Режим доступа: https://hypertextbook.com/facts/2003/ChrisDAmbrose.shtml (Дата обращения 2022-02-16).

5. Timbre [Электронный ресурс]. — Режим доступа: http://hyperphysics.phy-astr.gsu.edu/hbase/Sound/timbre.html (Дата обращения 2022-02-25).

6. Кодирование звуковой информации [Электронный ресурс]. — Режим доступа: https://www.sites.google.com/site/ivanovsinform/teoreticeskij-material/kodirovanie-informacii/kodirovanie-zvukovoj (Дата обращения 2022-04-12).

7. Introduction to Audio Storage [Электронный ресурс]. — Режим доступа: http://mediaintro.teeks99.com/Audio/Audio-1-Storage.html (Дата обращения 2021-11-17).

8. Principles of Data Acquisition and Conversion [Электронный ресурс]. — Режим доступа: https://www.ti.com/lit/an/sbaa051a/sbaa051a.pdf (Дата обращения 2021-11-17).

9. Types of Synthesis [Электронный ресурс]. — Режим доступа: https://www.schoolofsynthesis.com/sites/default/files/Types%20of%20Synthesis_0.pdf (Дата обращения 2021-12-09).

10. Audio and sound file extension list [Электронный ресурс]. — Режим доступа: https://www.file-extensions.org/filetype/extension/name/audio-and-sound-files (Дата обращения 2021-11-24).

11. R. Brown, An Introduction to Music Analysis with Computer [Электронный ресурс]. — Режим доступа: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.511.4241&rep=rep1&type=pdf (Дата обращения 2021-12-10).

12. Standard MIDI-File Format Spec. 1.1, updated [Электронный ресурс]. — Режим доступа: http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.htm (Дата обращения 2021-12-01).

13. *А., Барсова И.* Очерки по истории партитурной нотации (XVI — первая половина XVIII века) / Барсова И. А. — Москва, 1997.

14. PostgreSQL: The World's Most Advanced Open Source Relational Database [Электронный ресурс]. — Режим доступа: https://www.postgresql.org (Дата обращения 2022-05-01).

15. BLOB (binary large object) [Электронный ресурс]. — Режим доступа: https://searchsqlserver.techtarget.com/definition/BLOB (Дата обращения 2021-12-09).

16. Документация к PostgreSQL 9.6.24 [Электронный ресурс]. — Режим доступа: https://postgrespro.ru/docs/postgresql/9.6/index (Дата обращения 2022-05-01).

17. Content-Based Retrieval of Musical Scores in an Object-Oriented Database System [Электронный ресурс]. — Режим доступа: https://www.researchgate.net/publication/220993805_Content-Based_Retrieval_of_Musical_Scores_in_an_Object-Oriented_Database_System (Дата обращения 2021-09-20).

18. Нереляционные данные и базы данных NoSQ [Электронный ресурс]. — Режим доступа: https://docs.microsoft.com/ru-ru/azure/architecture/data-guide/big-data/non-relational-data (Дата обращения 2021-11-05).

19. Что такое базы данных NoSQL? [Электронный ресурс]. — Режим доступа: https://aws.amazon.com/ru/nosql/ (Дата обращения 2021-11-05).

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

21. JSON [Электронный ресурс]. — Режим доступа: https://www.json.org/json-en.html (Дата обращения 2022-05-01).

22. BSON [Электронный ресурс]. — Режим доступа: https://bsonspec.org (Дата обращения 2022-05-01).

23. Specification Version 1.1 [Электронный ресурс]. — Режим доступа: https://bsonspec.org/spec.html (Дата обращения 2022-05-05).

24. C - Project status and milestones [Электронный ресурс]. — Режим доступа: https://www.open-std.org/JTC1/SC22/WG14/www/projects#9899 (Дата обращения 2022-04-28).

25. News, Status Discussion about Standard C++ [Электронный ресурс]. — Режим доступа: https://isocpp.org (Дата обращения 2022-04-28).

26. Документация по C# [Электронный ресурс]. — Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (Дата обращения 2022-04-28).

27. Python [Электронный ресурс]. — Режим доступа: https://www.python.org (Дата обращения 2022-05-01).

28. Java [Электронный ресурс]. — Режим доступа: https://www.oracle.com/cis/java/ (Дата обращения 2022-05-25).

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Листинг 4.7 — Метод создания документа MongoDB по спецификации MIDI (часть 1)

```
BSONObj parseMidiFile(const vector<char> buffer, const string fileName) {
    BSONObjBuilder builder;
    int i;
    int i_buff;

    // MThd ID
    char MThdId[5];
    for (i_buff = 0, i = 0; i_buff < 4; i_buff++, i++) {
        MThdId[i] = buffer[i_buff];
    }
    builder.append("ID", MThdId);

    builder.append("Name", fileName);

    // MThd Length
    char MThdLength[5];
    for (i_buff = 4, i = 0; i_buff < 8; i_buff++, i++) {
        MThdLength[i] = buffer[i_buff];
    }
    builder.appendBinData("Length", 4, BinDataGeneral, MThdLength);

    // MThd Format
    char MThdFormat[3];
    for (i_buff = 8, i = 0; i_buff < 10; i_buff++, i++) {
        MThdFormat[i] = buffer[i_buff];
    }
    builder.appendBinData("Format", 2, BinDataGeneral, MThdFormat);

    // MThd NumTracks
    char MThdNumTracks[3];
    for (i_buff = 10, i = 0; i_buff < 12; i_buff++, i++) {
        MThdNumTracks[i] = buffer[i_buff];
    }
    builder.appendBinData("NumTracks", 2, BinDataGeneral, MThdNumTracks);

    // MThd Division
    char MThdDivision[3];
    for (i_buff = 12, i = 0; i_buff < 14; i_buff++, i++) {
        MThdDivision[i] = buffer[i_buff];
    }
    builder.appendBinData("Division", 2, BinDataGeneral, MThdDivision);
```

Листинг 4.8 — Метод создания документа MongoDB по спецификации MIDI (часть 2)

```
1       // MThd MTrks
2       const int numTracks = convertByteToIntMthd(MThdNumTracks);
3       BSONArray mtrks = BSONArray();
4       int mtrkBegin = i_buff;
5       int mtrkOffset = i_buff;
6       int j;
7       int i_mtrk;
8
9       BSONObjBuilder mtrksBuilder(builder.subarrayStart("MTrks"));
10      for (i_mtrk = 0; i_mtrk < numTracks; i_mtrk++) {
11          BSONObjBuilder mtrkBuilder(mtrksBuilder.subobjStart("mtrk"));
12          // MTrk ID
13          char MTrkId[5];
14          mtrkOffset += 4;
15          for (j = mtrkBegin, i = 0; j < mtrkOffset; j++, i++) {
16              MTrkId[i] = buffer[j];
17          }
18          mtrkBuilder.append("ID", MTrkId);
19
20          mtrkBegin = mtrkOffset;
21          mtrkOffset += 4;
22
23          // MTrk Length
24          char MTrkLength[5];
25          for (j = mtrkBegin, i = 0; j < mtrkOffset; j++, i++) {
26              MTrkLength[i] = buffer[j];
27          }
28          mtrkBuilder.appendBinData("Length", 4, BinDataGeneral, MTrkLength);
29          const int mtrkLengthInt = convertByteToIntMtrk(MTrkLength);
30
31          mtrkBegin = mtrkOffset;
32          mtrkOffset += mtrkLengthInt;
33
34          // MTrk Data
35          char* Data = new char[mtrkLengthInt];
36          for (j = mtrkBegin, i = 0; j < mtrkOffset; j++, i++) {
37              Data[i] = buffer[j];
38          }
39          mtrkBuilder.appendBinData("Data", mtrkLengthInt, BinDataGeneral,
                Data);
40
41          mtrkBegin = mtrkOffset;
```

Листинг 4.9 — Метод создания документа MongoDB по спецификации MIDI (часть 3)

```
1            mtrkBuilder.done();
2        }
3        mtrksBuilder.done();
4        BSONObj midi = builder.obj();
5
6        return midi;
7    }
```

Листинг 4.10 — Метод записи документа MongoDB в MIDI-файл (часть 1)

```
1  std::string parseMidiRecord(const BSONObj& doc) {
2      BSONObjIterator iter(doc);
3
4      BSONElement elem;
5      const char* bytes;
6      const char* fieldName;
7      const char* fileName;
8      std::string buffer;
9      int len;
10
11     while (iter.more()) {
12         elem = iter.next();
13         fieldName = elem.fieldName();
14         if (strcmp(fieldName, "_id") == 0)
15             continue;
16         if (strcmp(fieldName, "Name") == 0) {
17             fileName = elem.valueStringDataSafe().rawData();
18             continue;
19         }
20
21         BSONType t;
22
23         t = elem.type();
24         if (t == Object || t == Array) {
25             BSONObj subObj = elem.embeddedObject();
26             BSONElement subElem;
27             BSONObjIterator subObjIter(subObj);
28
29             while (subObjIter.more()) {
30                 subElem = subObjIter.next();
31
32                 t = subElem.type();
33                 if (t == Object || t == Array) {
34                     BSONObj mtrks = subElem.embeddedObject();
35                     BSONElement mtrk;
36                     BSONObjIterator mtrkIter(mtrks);
37
38                     while (mtrkIter.more()) {
39                         mtrk = mtrkIter.next();
40                         t = mtrk.type();
41                         bytes = getBsonElementValue(t, mtrk);
42                         len = mtrk.valuesize() - 5;
```

Листинг 4.11 — Метод записи документа MongoDB в MIDI-файл (часть 2)

```
                        for (int i = 0; i < len; i++) {
                            buffer += bytes[i];
                        }
                    }
                }
            }
        } else {
            len = elem.valuesize() - 5;
            bytes = getBsonElementValue(t, elem);

            for (int i = 0; i < len; i++) {
                buffer += bytes[i];
            }
        }
    }
    return buffer;
}
```