МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Харківський національний університет радіоелектроніки
Факультет Комп'ютерних наук
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

другий (магістерський)

(рівень вищої освіти)

Дослідження методів обробки, зберігання та передачі даних у мережевому середовищі для виконання колективних музикальних композицій

Виконав:
Студент 2 курсу, групи ІПЗм-21-1
Булгаков А.О.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-наукова

Керівник доц. Каук В.І.

(посада, прізвище, ініціали)

Допускається до захисту

Зав. Кафедри _____ З.В. Дудар

(підпис) (прізвище, ініціали)

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 121– Інженерія програмного забезпечення
(код і повна назва)

Тип програми освітньо-наукова

Освітня програма Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

« ___ » _____ 2023 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студента Булгакова Антона Олеговича

(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження методів обробки, зберігання та передачі даних у мережевому середовищі для виконання колективних музикальних композицій»

затверджена наказом університету від «29» 03.2023р. № 302 Ст

2. Термін подання студентом роботи до екзаменаційної комісії «___» _____202___р.

3. Вихідні дані до роботи *методи обробки звуку, технологія Java Sound API, мова розробки Java, пояснювальна записка*

4. Перелік питань, що потрібно опрацювати в роботі *мета роботи, аналіз предметної області та постановка задачі, планування експериментальної частини дослідження, архітектура та проектування веб системи для проведення звукозапису, проведення експерименту*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області дослідження	3 квітня 2023	Виконано
2	Постановка задачі	4 квітня 2023	Виконано
3	Дослідження методів роботи зі звуком	6 квітня 2023	Виконано
4	Планування експериментальної частини дослідження	10 квітня 2023	Виконано
5	Розробка програмного забезпечення, експерименту	11 квітня 2023	Виконано
6	Написання пояснювальної записки	24 квітня 2023	Виконано
7	Перевірка записки керівником та норм контролером	17 травня 2023	Виконано
8	Оцінка роботи стороннім рецензентом, отримання відгуку від керівника роботи, попередній захист	18 травня 2023	Виконано
9	Занесення диплома в електронний архів	20 травня 2023	Виконано
10	Здача готової роботи	22 травня 2023	Виконано

Дата видачі завдання 29 березня 2023 р.

Студент _____ Булгаков А.О.
(підпис) (прізвище, ініціали)

Керівник роботи _____ доц. Каук В.І.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Кваліфікаційна робота магістра містить: 70 с., 42 рис., 5 табл., 13 джерел, 5 додатків.

АУДІОФОРМАТ, БІТРЕЙТ, ЗВУКОЗАПИС, ОБРОБКА ЗВУКУ, ПЕРЕТВОРЕННЯ ФУР'Є, РЕПЕТИЦІЇ, ШУМООЧИЩЕННЯ, JAVA, JAVA SOUND API, MIDI, OPUS, SPRING.

Метою дослідження є порівняльний аналіз методів запису, обробки та збереження аудіо даних для колективного виконання музикальних композицій.

Об'єктом дослідження є програмні засоби запису, обробки, збереження та передачі аудіо у мережевому середовищі.

Предметом дослідження є алгоритми роботи з даними аудіо (видалення шумів, автоматична корекція, застосування ефектів), аудіоформати даних.

Методи розробки базуються на мовах програмування Java, JavaScript та TypeScript з використанням фреймворків та бібліотек Spring Framework, Angular Framework, Java Sound API, сервер бази даних PostgreSQL.

В результаті виконання магістерської кваліфікаційної роботи було досліджено методи запису, обробки, збереження та передачі аудіо у мережевому середовищі, які дають можливість створювати програмні системи для проведення колективних музичних репетицій.

AUDIOFORMAT, BITRATE, SOUND RECORDING, SOUND PROCESSING, FOURIER TRANSFORM, REHEARSALS, NOISE REDUCTION, JAVA, JAVA SOUND API, MIDI, OPUS, SPRING.

The purpose of the study is a comparative analysis of the methods of recording, processing and storing of audio data for collective performance of musical compositions.

The object of the research is software tools for recording, processing, storing and

transmitting audio in the network environment.

The development methods are based on programming languages Java, JavaScript and TypeScript with usage of frameworks and libraries Spring Framework, Angular Framework, Java Sound API, database server is PostgreSQL.

As a result of the master's qualification work, the methods of recording, processing, storing and transmitting audio in a network environment, that make it possible to create software systems for conducting collective musical rehearsals, were investigated.

Умови публікації пояснювальної записки

Я, Булгаков Антон Олегович, студент гр. ІПЗм-21-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження методів обробки, зберігання та передачі даних у мережевому середовищі для виконання колективних музикальних композицій», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений (а) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

У сучасному світі щодня відбуваються тисячі сольних та групових виконань музикальних репетицій. У більшості випадків вони відбуваються із застосуванням класичних музикальних інструментів, і не потребують застосування цифрових технологій.

Тим не менш, на сьогоднішній день в Україні можливість виконання музикальних композицій кількома людьми в одному приміщенні обмежена здебільшого з політичних та епідеміологічних причин. Окрім цього, обмеження швидкості Інтернету під час військових дій ускладнює використання мережевих застосувань для спільного виконання. Через це було прийняте рішення про дослідження методів цифрової обробки та зберігання даних для покращення можливостей спеціалізованих програмних застосувань.

В першу чергу мова йде про дослідження алгоритмів фільтрації шумів, які дозволяють видалити з кінцевої аудіодоріжки зайві звуки, такі як сигнал повітряної тривоги. Більш того, контекст дослідження передбачає можливість фільтрації нестаціонарних шумів, що є важко досяжним у системах для проведення відеоконференцій. По-друге, оброблені дані мають зберігатися у такому аудіоформаті, який найбільше підходить для швидкої передачі мережею, при цьому не втрачаючи якості звуку. По-третє, виконання сучасної музики інколи потребує застосування спеціальних ефектів, які дозволяють посилити або навіть повністю змінити оригінальне звучання інструмента. Застосування таких ефектів може бути досяжним без застосування додаткового обладнання.

Зрештою, для перевірки теоретичних припущень, було проведено два експерименти. Проведення експерименту передбачає виконання ряду тестів із використанням різних методів обробки даних, аудіоформатів, протоколів передачі даних та, власне, самих даних. Тому робота містить планування експериментальної частини дослідження, а саме визначення досліджуваних параметрів, визначення методів проведення експерименту, проєктування його реалізації та опис технологій, що будуть використані при його проведенні.

Під час виконання роботи були використані переважно теоретичний та емпіричний методи дослідження, оскільки теорії щодо доцільності застосування тих чи інших методів роботи зі звуком висувалися виходячи з теоретичних даних, а висновки робилися здебільшого після проведення експериментальних тестувань.

Отримані результати також можуть бути використані для подальших досліджень у сфері звукозапису, також досліджена інформація може допомогти у створенні нових систем та застосунків для проведення дистанційних репетицій у режимі реального часу.

Під час виконання роботи було розроблено прототип системи для проведення дистанційних репетицій, який являє собою клієнт-серверне застосування з можливістю звукозапис та обробки аудіо. Записи звуку при проведенні тестування проводилися із застосуванням акустичної гітари та окремого мікрофона.

Середою розробки було обрано IntelliJ Idea з використанням мов програмування Java, JavaScript та TypeScript та веб технологій, менеджери пакетів Gradle та NPM.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області дослідження

На сьогоднішній день неможливо представити Інтернет середовище без систем, у яких використовується потоковий запис звуку. У першу чергу, такими системами є мессенджери та застосування для проведення онлайн конференцій, такі як Skype, Google Meet, Microsoft Teams, Zoom та інші. Слід зауважити, що події останніх років, такі як пандемія Covid-19 і повномасштабне вторгнення в Україну, останнім часом сприяли популяризації цих застосувань, що підтверджує дохід компанії Zoom Video Communications (див. рис. 1.1), чия система Zoom залишається лідером у своїй сфері[1].

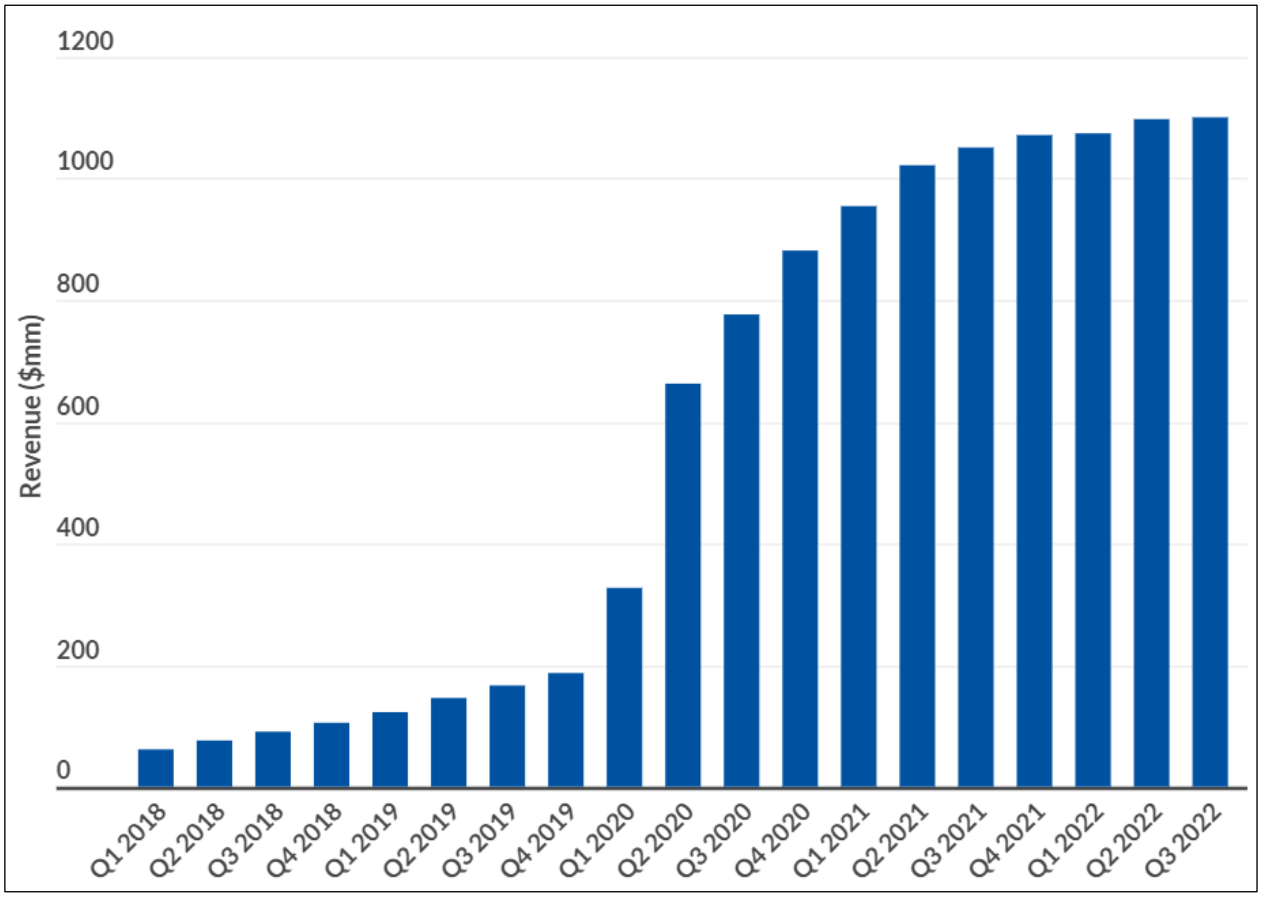

Рисунок 1.1 – Дохід компанії Zoom Video Communications у 2018-2022 рр.[1]

Тим не менш, затримка у цих застосуваннях є занадто великою для того, щоб зробити можливою колективне виконання музикальних композицій. Водночас постає питання синхронізації виконання, бо учасники гри мають

починати свої партії вчасно, а зазвичай навіть одночасно. Існують застосування, які вирішують ці проблеми, такі як JamKazam[2], Jamulus та Jamstud.io.

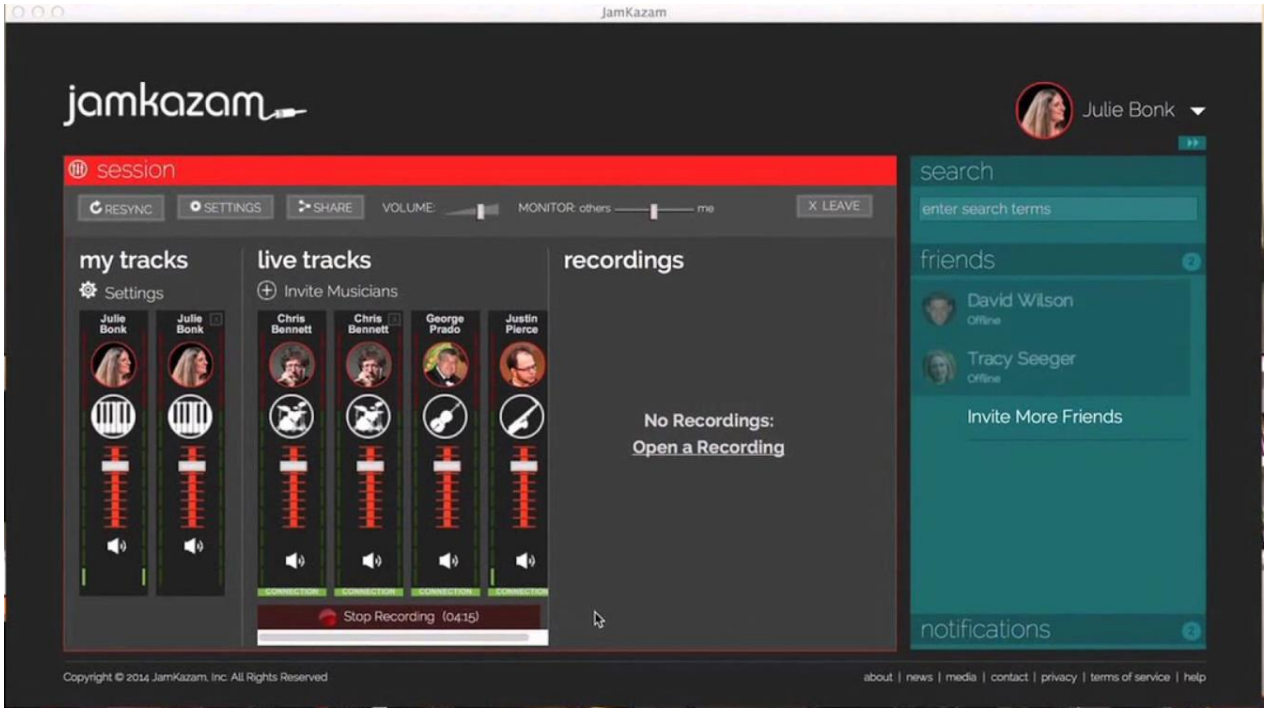

Рисунок 1.2 – Інтерфейс застосування JamKazam[2]

Звичайно, такі застосування не можуть вирішити актуальних проблем, пов’язаних зі швидкістю Інтернет трафіку або якістю мікрофона, але ж вони ефективно розв’язують задачі обробки, збереження та передачі даних мережею, хоча усі вони мають виключно десктопний інтерфейс. Окрім цього, вказані програми містять функції, які дозволяють відредагувати аудіодоріжки – додати звукові ефекти, змінити гучність певного інструменту тощо. Деякі з цих функцій є широко розповсюдженими і наявні у будь-яких аудіоредакторах, інші є спеціалізованими і відносяться до конкретного інструменту (наприклад, «distortion», «reverb» чи «chorus» ефекти для електрогітари).

Аналізуючи ці застосування, можна виділити ряд характеристик, які повинні мати методи, що використовуються для обробки, зберігання та передачі даних для систем з таким, більш традиційним, способом проведення репетицій:

– невеликий обсяг пакетів, що відправляються мережею;
– суттєве стиснення даних;
– фільтрація сторонніх звуків.

Використання веб застосувань відкриває багато можливостей та способів проведення колективного виконання композицій. Команді початківців може бути складно одразу програвати композицію разом, і замість цього можна грати лише свою партію, увімкнувши на фоні «ідеальні» доріжки інших гравців. а потім об’єднувати разом записані доріжки. Це дає змогу не відволікатися на сторонні помилки у грі, а аналізувати їх вже після виконання. У такому випадку пріоритети у виборі методів роботи з даними змінюються наступним чином:

- розмір пакетів даних перестає мати значення;
- стиснення даних є важливим для швидкодії системи, але пріоритетним стає якість запису;
- фільтрація сторонніх звуків стає опціональною, оскільки не заважає грі інших виконавців.

І зрештою, існують ситуації, які пов’язані з обробкою даних при виконанні музикальних композицій, але мають мало спільного з репетиціями. Прикладом може слугувати вокальне виконання пісні під «мінус», тобто караоке. У цьому випадку взаємодія відбувається не у мережі Інтернет, а у локальній мережі (між мікрофоном та пристроями запису та відтворення звуку), а методи обробки та передачі даних мають нові пріоритети:

- миттєве відтворення даних без затримок;
- оригінальне звучання голосу, або ж навпаки – зберігання тембру та зміна висоти звучання для досягнення більшої відповідності з оригіналом;
- зберігання запису взагалі стає опціональним.

Таким чином, у різному контексті методи роботи з даними при виконанні композицій у мережевому середовищі мають різні пріоритети, тому їх вибір для кожної задачі може бути різним.

1.2. Природа звуку. Методи обробки звукових сигналів

Звук – це коливання повітря, що впливають на орган слуху людини. Фізично звуки відрізняються трьома характеристиками:

- частотою коливань (висотою);

– амплітудою коливань (гучністю);
– тембром.

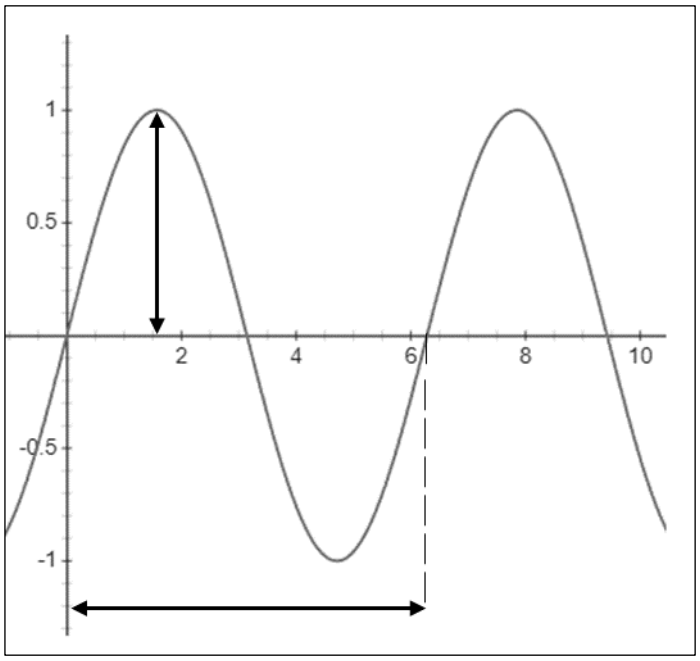

Рисунок 1.3 – Графік гармонічних звукових коливань[Рисунок виконаний самостійно*]

Звук камертона є гармонічним коливанням (див.рис.1.3), оскільки складається лише з одного простого тона[3]. Цей звук не містить кольорового забарвлення, але його легше виділити у звуковому потоці. Майже всі музичні інструменти здійснюють акустичну хвилю складних звуків, яка називається негармонічним коливанням. Такі звуки складаються з кількох простих гармонічних коливань, а різниця у звуках сприймається людиною суб'єктивно.

Для роботи зі звуком як з набором даних використовується звуковий сигнал – зміна фізичної величини, що з фізичної точки зору може бути описана як коливальний рух. Далі поняття «звук» та «звуковий сигнал» можна вважати тотожними.

Таким чином, під час обробки звуку аналізується та змінюється здебільшого частота та амплітуда коливань. Для візуалізації даних та змін у них використовується амплітудно-частотний спектр звуку – графік залежності відносної енергії звукових коливань від частоти.

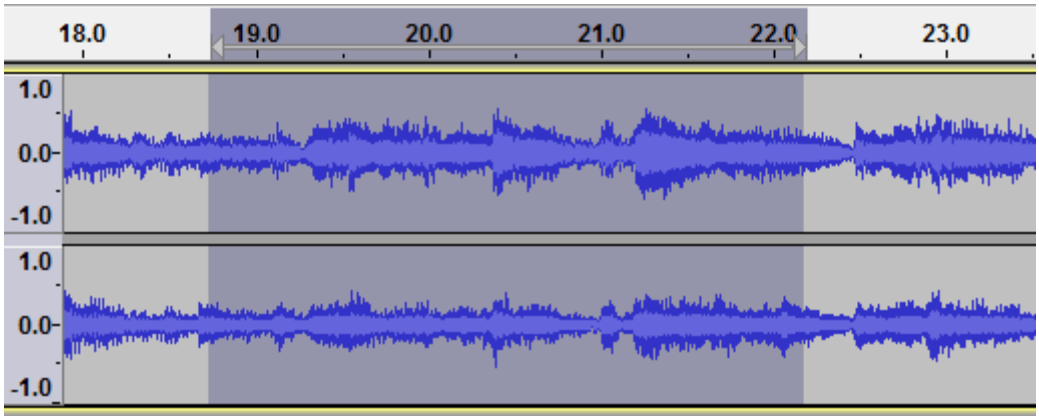

Рисунок 1.4 – Амплітудно-часовий спектр в аудіоредакторі Audacity[Рисунок виконаний самостійно*]

Обробка даних при потоковому записі та передачі звуку здійснюється, здебільшого, для досягнення двох цілей. Першою такою ціллю є покращення якості звуку шляхом шумоочищення – виділення з результативного звукового сигналу корисної частини та фільтрація адитивного сигналу-перешкоди. Другою ціллю є зміна характеристик сигналу з метою отримання нового звучання – будь то імітація гри на іншому музичному інструменті або зміна висоти з метою досягнення кращої відповідності виконання вокаліста цільовій аплікатурі.

1.3 Постановка задачі

Відповідно до теми роботи, задачею цього дослідження є аналіз та порівняння методів, що використовуються під час збереження та обробки даних під час запису та відтворення звуку.

Якщо це можливо, то мають бути з’ясовані основні критерії для порівняння, які описують ці методи. Також повинен бути проведений попередній аналіз, який визначатиме теоретичну перевагу одних методів збереження та обробки даних над іншими.

Після проведення теоретичного аналізу його результати мають бути підтверджені практично за допомогою експерименту. Для цього має бути спроєктований та реалізований прототип програмного застосування для запису

звуку та його обробки. Окрім цього, прототип має отримувати та передавати звук на кілька пристроїв різних користувачів для коректної імітації репетиції.

Створений прототип системи має зберігати записані дані у вигляді, придатному для відтворення різними платформами та для їх аналізу за допомогою аудіоредактора. Окремо мають зберігатися оригінальний звук та його змінена версія.

2 ДОСЛІДЖЕННЯ МЕТОДІВ ОБРОБКИ ТА ЗБЕРІГАННЯ ДАНИХ ПРИ РОБОТІ ЗІ ЗВУКОМ

2.1 Дослідження методів обробки даних

2.1.1. Методи шумоочищення

Шум – це сукупність аперіодичних звуків різної інтенсивності й частоти. Шуми поділяють на стаціонарні та нестаціонарні, широкосмугові та вузькосмугові, постійними і непостійними. Особливість шуму полягає у тому, що він неперервно заповнює деякий інтервал[4].

Формально процес шумоочищення можна записати у вигляді формули 2.1:

$$f_1(t) + f_2(t) \to f_3(t) \simeq f_1(t) \tag{2.1}$$

де $f_1(t)$ – корисна частина сигналу,

$f_2(t)$ – шуми, $f_3(t)$ – сигнал, що є наближенням сигналу $f_1(t)$.

Потокове шумоочищення звуку є найбільш складною задачею цього напрямку, оскільки можливість обробки потокових даних без затримок накладає на алгоритм ряд суттєвих обмежень: він не може бути ітеративним із заздалегідь невідомим числом ітерацій і не може явно використовувати дані, що знаходяться до або після поточного оброблюваного блоку. Першим методом, який розглядається у роботі, є фільтрація нижній та верхніх частот, які можна застосовувати для фільтрації стаціонарних шумів. Стаціонарні шуми – це шуми постійної інтенсивності і спектральної щільності, такі як шелестіння листя або шум вітру. Сутність методу полягає в тому, що налаштовані фільтри не пропускають звукові сигнали, що знаходяться на рівні з низьким вмістом корисної частини та високої кількістю шумів. Для оцінювання вмісту шумів та корисної частини у звуковому сигналі використовується алгоритм дискретного перетворення Фур'є[5], а після визначення «межі» та проведення фільтрації – алгоритм зворотнього перетворення Фур'є. Дискретне перетворення Фур'є також може використовуватись для аналізу звука з метою визначення зіграного

акорда[6], що також може бути корисним при подальшому розвитку системи проведення онлайн репетицій (див.рис.2.1).

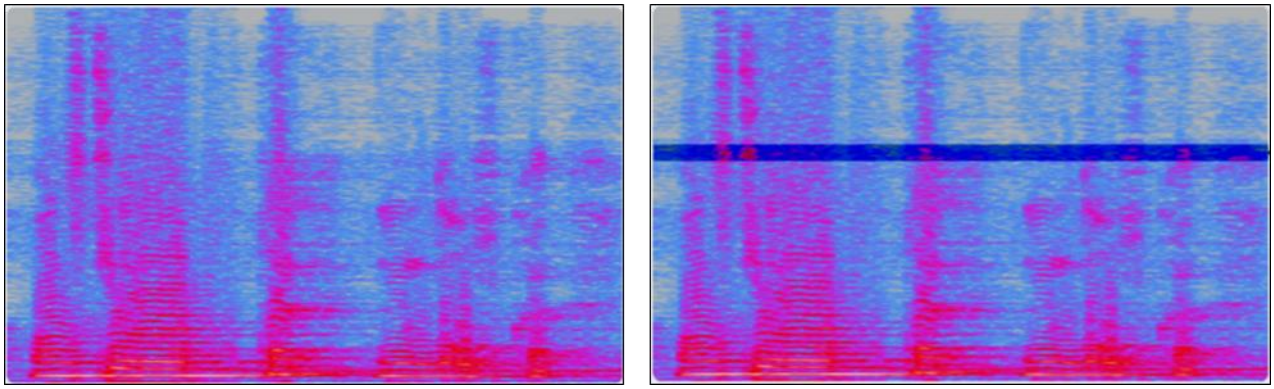

Рисунок 2.1 – Стенограма звукового сигналу та рівень високих частот, які будуть відфільтровані[7]

Другим методом є класична Вінерівська фільтрація, коли частотну характеристику фільтра підлаштовують під характеристики вхідних даних, враховуючи апріорні відомості про відношення сигнал-шум на різних частотах[7].

При обрахуванні цього вінерівського фильтра мінімізується метрика Mean Square Error (MSE) (формула 2.2):

$$H(w) = \frac{P_{ss}(w)}{P_{yy}(w)} = \frac{P_{yy}(w) - P_{dd}(w)}{P_{yy}(w)} \quad (2.2)$$

де $P_{ss}(w)$ – спектр чистого сигналу,

$P_{yy}(w)$ – спектр зашумленого сигналу,

$P_{dd}(w)$- спектр шумів.

Таким чином, застосування Вінерівської фільтрації є виправданим у випадку, коли одна з величин – спектр чистого сигналу або спектр шумів – відома.

Цей алгоритм використовується у стандарті WebRTC для обробки аудіоданих перед їх потоковою передачею.

Третій метод, який використовується для обробки аудіоданих, відрізняється від попередніх тим, що для його застосування використовується два фізичних

засоби звукозапису. Два мікрофони, що знаходяться на деякій відстані один від одного, записують сигнали, що знаходяться в ареалі доступу, причому один із мікрофонів знаходиться в безпосередній близькості до джерела корисного сигналу. Це дозволяє скористатися спектральним відніманням фрагментів одного сигналу з іншого.

Існують також методи фільтрації шумів, засновані на використанні нейронних мереж. У цій роботі вони не були розглянуті.

2.1.2 Фільтрація нестаціонарних шумів

Нестаціонарні шуми – це такі шуми, які тривають короткі проміжки часу. Прикладами таких явищ можуть бути окремі стуки у виробничих умовах, різкі хлопки, сигнали повідомлень деяких месенджерів.

Зазвичай, програмні застосування не дозволяють відфільтровувати такі типи шумів, і з точки зору предметної області, що розглядається, такі шуми можуть бути частиною композиції. Іноді справді буває так, що звук, який умовно можно прийняти за нестаціонарний шум, є частиною композиції (наприклад, постріл з револьвера у композиції Енніо Морріконе «The Good, the Bad and the Ugly»).

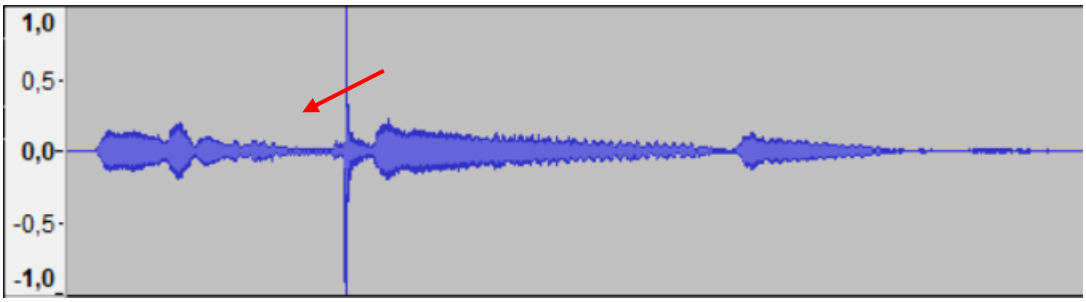

Рисунок 2.2 – Приклад нестаціонарного шуму (різкий хлопок)[Рисунок виконаний самостійно*]

Тим не менш, саме використання технологій у музичній сфері може дозволити автоматично очистити доріжку від таких шумів під час проведення репетиції.

Принцип полягає у тому, щоб завчасно передати програмному застосуванню взірець ідеального виконання у певному розповсюдженому форматі

(див.рис.2.3). Таким форматом може бути MIDI (Musical Instrument Digital Interface) або один з форматів GP (.gp5, .gp6, .gpx тощо).

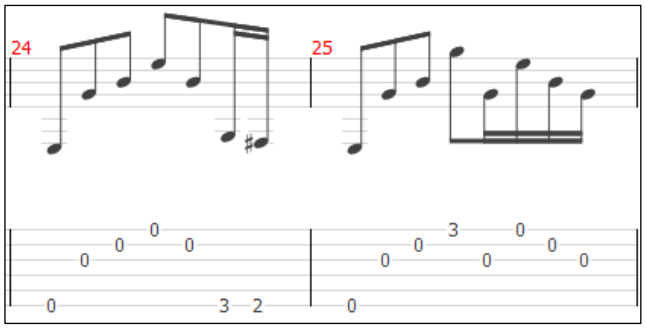

Рисунок 2.3 – Приклад фрагменту-взірця, у якому немає різких звуків[Рисунок виконаний самостійно*]

У такому випадку, при контрастному неспівпадінні очікуваного та реального звуку у момент часу його можна ідентифікувати та «вирізати» з кінцевого результату (див. рис. 2.4).

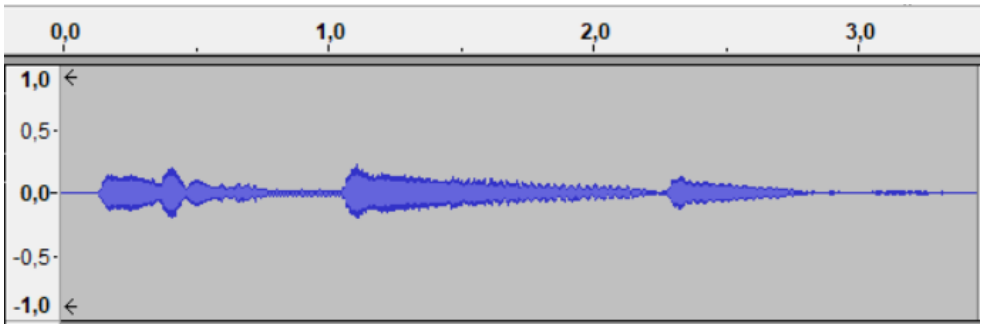

Рисунок 2.4 – Звукова доріжка після очищення шуму[Рисунок виконаний самостійно*]

Водночас, застосування такого підходу передбачає прив’язку до певного формату чи декількох форматів файлів-зразків, що може поскладнити подальшу підтримку системи проведення репетицій.

2.1.3 Звукові ефекти та їх застосування

Існує велика кількість звукових прийомів та ефектів, які музиканти застосовують у своїй практиці. Багато з них досягаються самим способом виконання композиції на музичному інструменті і не можуть бути достовірно

змодельовані програмною системою. До таких прийомів можна віднести глісандо, яке досягається шляхом ковзання пальцем по грифу гітари чи ковзанням куліси на тромбоні, чи гра флажолетами, яка отримується шляхом легкого торкання пальцем до струни на грифі гітари під час програвання звуку.

Другий тип ефектів можна охарактеризувати як електронні. Вони зазвичай використовуються для виконання електронної музики, насамперед гітарної, і у студійних умовах вимагають використання додаткових апаратних засобів: підсилювачів, попередніх підсилювачів, педалей тощо. Ці ефекти можуть бути відтворені виключно цифровими методами, хоча при цьому неможливо досягти того ж ефекту, як під час реальних репетицій та концертних виступів. До найбільш популярних таких ефектів можна віднести наступні:

- овердрайв (overdrive);
- дисторшн (distortion);
- delay (delay);
- хорус (chorus).

Delay – це ефект, який досягається затримкою повторного сигнала, що створює враження подвоєння звуку. Зазвичай ефект не постійний, і його тривалість регулюється під час виконання композиції.

Хорус також досягається дублюванням сигналу, але цей сигнал має інше забарвлення та порівняно коротку затримку, що створює ефект одночасного виконання ноти декількома інструментами.

Окремо слід зазначити такий техніко-композиційний прийом як канон. Це, по факту, повторення однієї і тієї самої партії із затримкою двома виконавцями, але воно може бути легко досягнуто технічно простим повтором аудіодоріжки. Прикладом композиції, у якій застосовано прийом канону, є композиція Євгена Штефана «Зоряний дощ».

Варто зазначити, що під час штучного застосування ефекту канона з’являється можливість його запису у другий звуковий канал, що дозволить зробити звук об’ємнішим при його відтворенні. Інакше буде створений одноканальний (моно) звук (див. рис. 2.5),

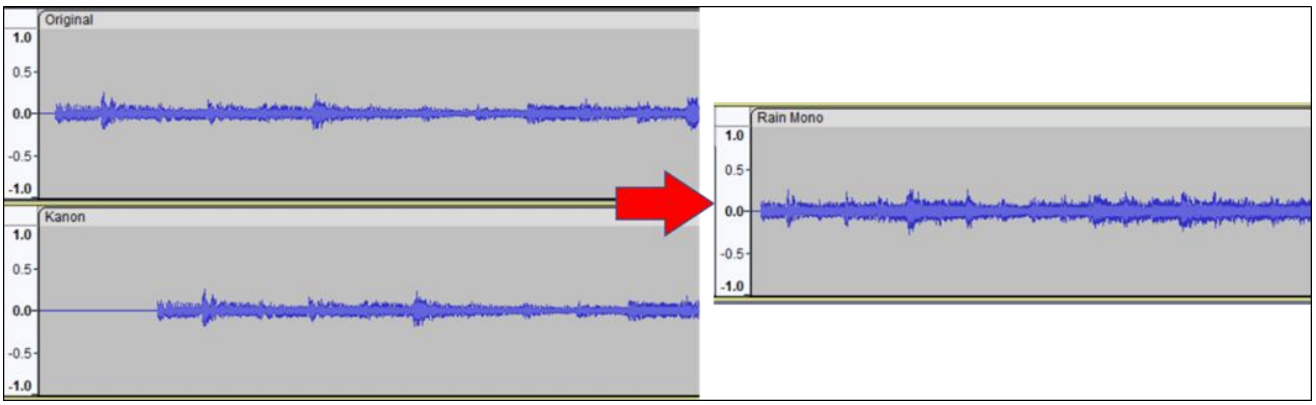

Рисунок 2.5 – Перетворення потенційного стерео звуку в одноканальний[Рисунок виконаний самостійно*]

Овердрайв – це ефект звучання натурального інструмента, який досягається посиленням звучання та перевантаженням посилювача.

Дисторшн – це ефект, схожий за природою з овердрайвом, але більш перевантажений і менш натуральний за звучанням. Фізично це досягається за рахунок обрізання амплітуди звукових коливань.

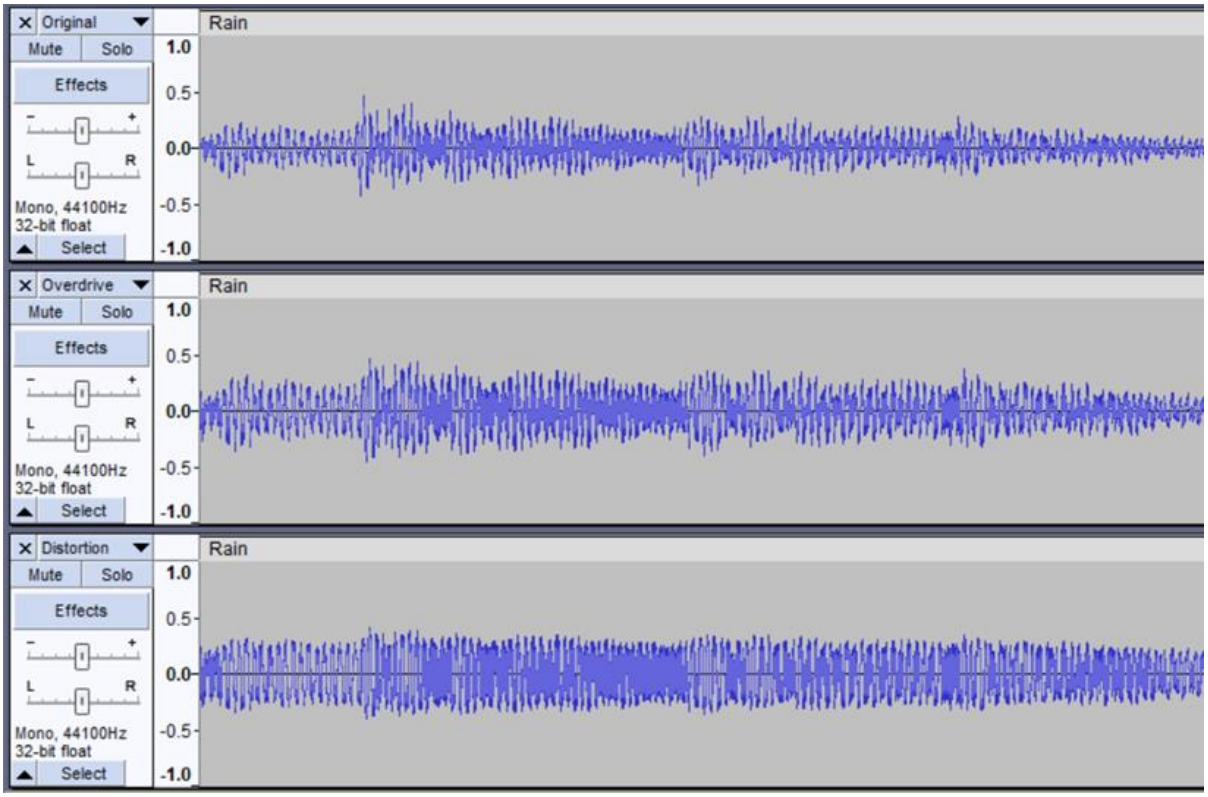

Рисунок 2.6 – Порівняння спектрів оригінального звуку, звуку з ефектом овердрайв та звуку з ефектом дисторшн[Рисунок виконаний самостійно*]

2.2 Дослідження методів зберігання даних

Одним з головних виборів, які робиться при проєктуванні програмних застосувань, що передбачають передачу аудіо даних мережею, є вибір цифрового аудіоформату – формату представлення даних, який використовується при цифровому звукозаписі. Аудіоформати поділяються на три групи – формати без стиснення, формати зі стисненням без втрати якості та формати зі стисненням з втратою якості. У ході роботи були розглянуті тільки останні, оскільки в рамках роботи з мережею обсяг даних, що передаються, має велике значення.

У ході аналізу предметної області були виявлені наступні популярні аудіоформати:

- AAC (Advanced Audio Coding);
- MP3;
- Vorbis;
- Opus;
- WMA (Windows Media Audio).

Власне, задача багатокритеріального вибору була поставлена наступним чином – обрати аудіоформат, який найбільш придатний для передачі даних мережею як для передачі в режимі реального часу, так і для завантаження і відтворення файлів заданого формату після їх формування.

Вказані аудіоформати порівнювалися за наступними характеристиками:

- максимальна частота дискретизації – величина, яка показує, на яку максимальну кількість дискретних частин може бути поділена неперервна звукова доріжка. Обумовлює якість звуку;
- максимальний бітрейт – кількість бітів в одиницю часу, яка визначає деталізованість цифрового запису. Формат з великим бітрейтом буде якісніше відтворювати оригінальний звук, але файли такого формату будуть займати більше місця на диску;

- кількість каналів – кількість «потоків», на які може бути розділена загальна композиція. Більша кількість каналів дозволяє зробити звук об’ємнішим;
- затримка – характеристика, яка визначає час, необхідний для переключення між каналами. Має велику роль при створенні систем, що працюють у режимі реального часу;
- використання лінійного прогнозування – при використуванні лінійного прогнозування збільшується швідкість кодування і, відповідно, якість використання формату у мережевому середовищі;
- наявність ліцензії – запатентовані аудіоформати значно частіше стають рекомендованими для використання у формалізованих структурах (державних установах та банках).

У той час, коли перші 4 ознаки є числовими, і їх легко порівнювати, останні вимагають формування шкал для їх порівняння:

а) використання лінійного прогнозування:
   1) 1 – якщо використовується;
   2) 0 – якщо не використовується;

б) наявність ліцензії:
   1) 0 – якщо не запатентована;
   2) 1 – якщо не запатентована.

Перш за все, було побудовано таблицю (табл.2.1), яка містить дані про аудіоформати та їх характеристики[8]:

Таблиця 2.1 – Характеристики аудіоформатів [Таблиця виконана самостійно*]

	Частота дискретизації, кГц	Максимальний бітрейт, кбіт/с	Кількість каналів	Приблизна затримка, мс	Лінійне прогнозування	Наявність ліцензії
AAC	192	529	48	200	1	0
MP3	48	320	2	100	0	1

Кінець таблиці 2.1 [Таблиця виконана самостійно*]

	Частота дискретизації, кГц	Максимальний бітрейт, кбіт/с	Кількість каналів	Приблизна затримка, мс	Лінійне прогнозування	Наявність ліцензії
Vorbis	192	1000	255	100	0	1
OPUS	48	510	255	32	1	1
WMA	96	768	8	100	0	1

Для того, щоб порівнювати ці характеристики, було проведено нормування цих характеристик за еталоном за формулою 2.3:

$$f = \frac{f_{\text{измер}}}{f_{\text{еталон}}} \tag{2.3}$$

Оскільки для значення затримки кращими є менші показники, то для нормалізації використовувалася обернена формула 2.4:

$$f = \frac{f_{\text{еталон}}}{f_{\text{измер}}} \tag{2.4}$$

Після нормування таблиця набула наступного вигляду (табл.2.2):

Таблиця 2.2 – Характеристики аудіоформатів після нормалізації[Таблиця виконана самостійно*]

	Частота дискретизації	Максимальний бітрейт	Кількість каналів	Приблизна затримка	Лінійне прогнозування	Наявність ліцензії
AAC	1	0.529	0.19	0.16	1	0

Кінець таблиці 2.2

	Частота дискретизації	Максимальний бітрейт	Кількість каналів	Приблизна затримка	Лінійне прогнозування	Наявність ліцензії
MP3	0.25	0.32	0.008	0.32	0	1
Vorbis	1	1	1	0.32	0	1
OPUS	0.25	0.51	1	1	1	1
WMA	0.5	0.768	0.031	0.32	0	1

Далі було здійснено перевірку за Парето. Якщо порівняти між собою формати Vorbis та WMA, то з’ясується, що останній дозволяє запис звуку з меншою дискретизацією, бітрейтом та кількістю каналів, та не має жодних переваг. Таким чином, аудіоформат WMA є неефективним за Парето та його можна видалити з вибірки.

Таким саме чином, порівнюючи аудіоформати Opus та MP3 можна побачити, що останній програє майже за усіма характеристиками. Він так само був видалений з вибірки за принципом Парето (табл.2.3).

Таблиця 2.3 – Аудіоформати після перевірки за Парето [Таблиця виконана самостійно*]

	Частота дискретизації	Максимальний бітрейт	Кількість каналів	Приблизна затримка	Лінійне прогнозування	Наявність ліцензії
AAC	1	0.529	0.19	0.16	1	0
Vorbis	1	1	1	0.32	0	1
OPUS	0.25	0.51	1	1	1	1

Для вирішення задачі було використано лінійну адитивну згортку з ваговими коефіцієнтами. Формула 2.5 визначення кінцевого показника якості для обраного методу:

$$f = max \sum_{j=1}^{n} \alpha_j \beta_j a_{i,j} \quad (2.5)$$

У формулі α_j – нормуючі множники, β_j – вагові коефіцієнти.

Вагові коефіцієнти, які характеризують важливість критеріїв, визначалися згідно з предметною областю:

- частота дискретизації, бітрейт та кількість каналів впливають на якість аудіо даних і мають однаковий коефіцієнт 0.15;
- наявність ліцензії – необов'язковий фактор. Ваговий коефіцієнт – 0.05;
- затримка та лінійне прогнозування – це ті фактори, які зумовлюють швидкість передачі даних у мережевому середовищі. Ваговий коефіцієнт – 0.25.

Після відповідних обрахувань були отримані наступні результати (табл.2.4):

Таблиця 2.4 – Результати обчислення показників якості [Таблиця виконана самостійно*]

	Частота дискретизації	Макс. бітрейт	Кількість каналів	Приблизна затримка	Лінійне прогнозування	Наявність ліцензії	Результат
AAC	1	0.529	0.19	0.16	1	0	0.271
Vorbis	1	1	1	0.32	0	1	0.288
OPUS	0.25	0.51	1	1	1	1	0.442
Вагові коефіцієнти	0.15	0.15	0.15	0.25	0.25	0.05	

Таким чином, здебільшого за рахунок низької затримки та наявності механізму лінійного прогнозування, аудіоформат Opus виявляється тим форматом, який найбільше підходить для цілей швидкої та якісної передачі даних у заданих умовах.

3 ПЛАНУВАННЯ ЕКСПЕРИМЕНТАЛЬНОЇ ЧАСТИНИ ДОСЛІДЖЕННЯ

3.1 Сутність експерименту

Головною метою експерименту є доведення висунутих теорій щодо ефективності методів обробки та збереження аудіо даних у мережевому середовищі. Порівняння цих методів повинне буте здійснено за кількома критеріями:

- швидкість формування фрагменту даних на клієнтській частині програмного застосування;
- швидкість передачі фрагменту даних з клієнтської частини програмного застосування на серверну;
- обсяг пам'яті, яку займають закодовані дані на серверній частині;
- швидкість обробки даних на серверній частині;
- швидкість передачі даних з серверної частини на клієнтську;
- швидкість перекодування даних на клієнтській частині;
- якість отриманого фрагменту даних;
- існування бібліотек для використання методу, що розглядаться.

Таким чином, основним фактором, який впливає на фінальний результат, є час між початком запису фрагменту даних та завершенням перекодування фрагменту даних на боці іншого клієнта (формула 3.1).

$$t_{\text{заг}} = t_{\text{к1}} + t_{\text{тр1}} + t_{\text{с}} + t_{\text{тр2}} + t_{\text{к2}} \tag{3.1}$$

де $t_{\text{к1}}$ – час формування фрагменту даних,

$t_{\text{тр1}}$ – час передачі фрагменту на серверний бік,

$t_{\text{с}}$ – час обробки даних сервером,

$t_{\text{тр2}}$ – час передачі фрагменту на клієнтський бік,

$t_{\text{к2}}$ – час обробки даних другим клієнтом.

Тоді ступінь придатності засобу до застосування у мережевому середовищі (U) можна подати наступною залежністю:

$$U = f(t_{\text{заг}}, m, q, l) \quad (3.2)$$

де m (memory) – обсяг пам'яті, яку займають закодовані дані,

q (quality) – показник якості отриманих даних,

l (library) – кількість бібліотек, які підтримують обробку із використанням заданого методу.

З метою отримання даних про фактори, які впливають на ступінь придатності засобу до застосування, повинні бути розробленими кілька компонентів системи, які, власне, містять алгоритм перекодування та перетворення даних. Ці компоненти можуть бути розробленими власноруч або використовувати існуючі бібліотеки.

3.2 Проєктування програмного прототипу системи

У ході дослідження потрібно створити прототип системи, який дозволить протестувати різні засоби обробки, збереження та передачі даних. У якості такого прототипу було обрано систему для проведення онлайн репетицій. Така система має складатись з кількох фізичних елементів. Ця система є багаторівневою та взаємодіє з різними серверами та методами збереження даних, що показано на діаграмі розгортання системи (див. рис. 3.1).

Для реалізації такого прототипу буде застосований класичний архітектурний патерн MVC, оскільки бізнес-логіка прототипу системи має бути відокремлена від відображення[9]. Окрім цього, існує велика кількість програмних засобів, які дозволяють відносно швидко створити багаторівневу систему на базі цього патерна, зосередивши більше зусиль саме на програмній реалізації методів роботи зі звуком.

Серверна частина застосування повинна надавати такий програмний інтерфейс, який дозволяє передачу даних в обом напрямках. Найкращим способ реалізації у цьому випадку є використання патерну Observer, що дозволить користувачам взаємодіяти з сервером у режимі відправки повідомлень і підписки на події.

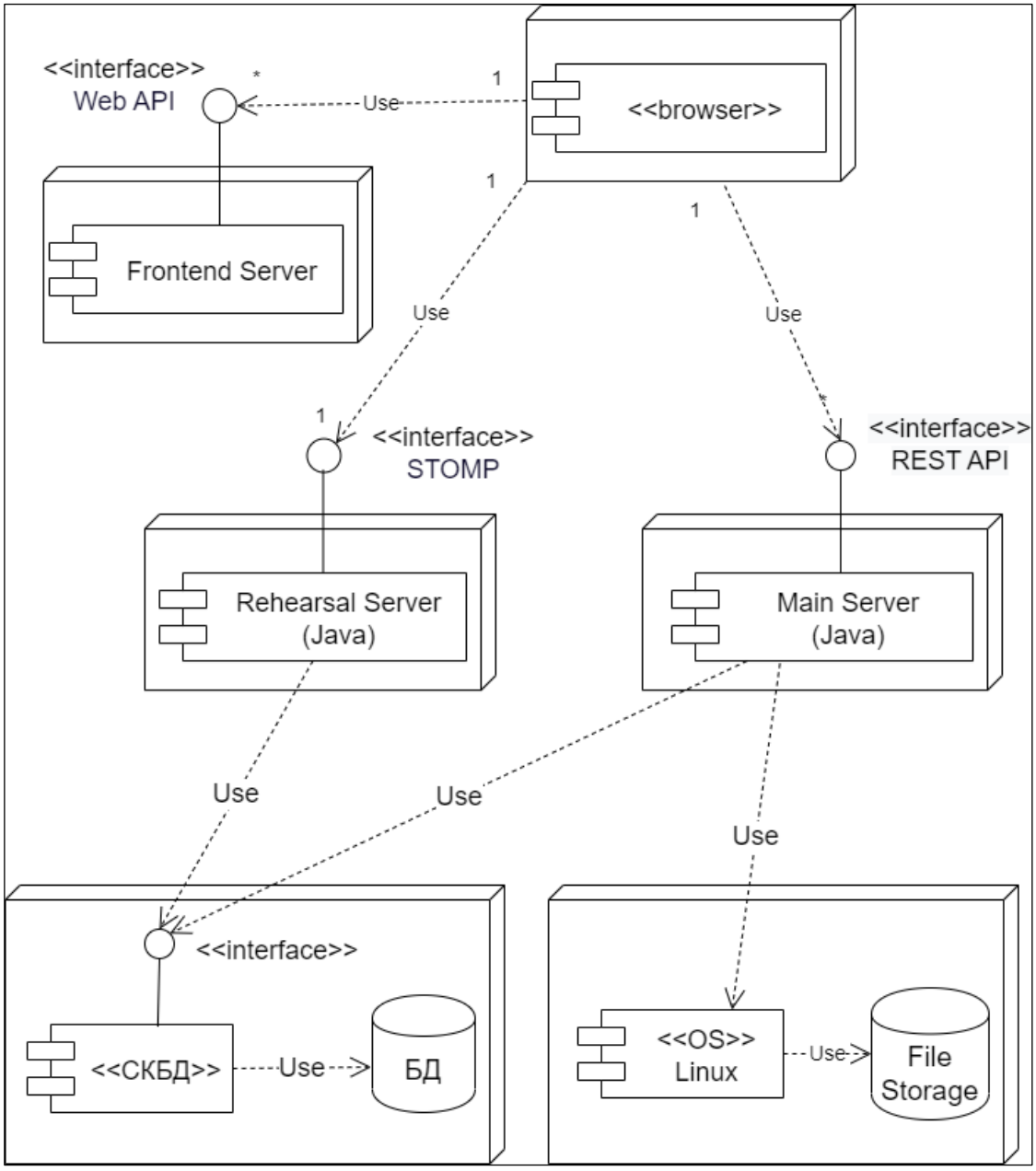

Рисунок 3.1 – Діаграма розгортання[Рисунок виконаний самостійно*]

Для здійснення передачі даних повинна бути створена репетиція – у межах системи це означає створення одночасної сесії для кількох користувачів. Для того, щоб приєднатися до репетиції, необхідна взаємодія кількох частин системи. Нова кімната (лоббі) для проведення репетиції повинна бути створена у базі даних перед приєднанням до неї, після чого до неї зможуть приєднатись учасники. Для подальших процесів має використовуватись окремий, більш потужний, сервер. Процес створення та приєднання різними клієнтами до репетиції зображено на діаграмі послідовностей (див. рис. 3.2).

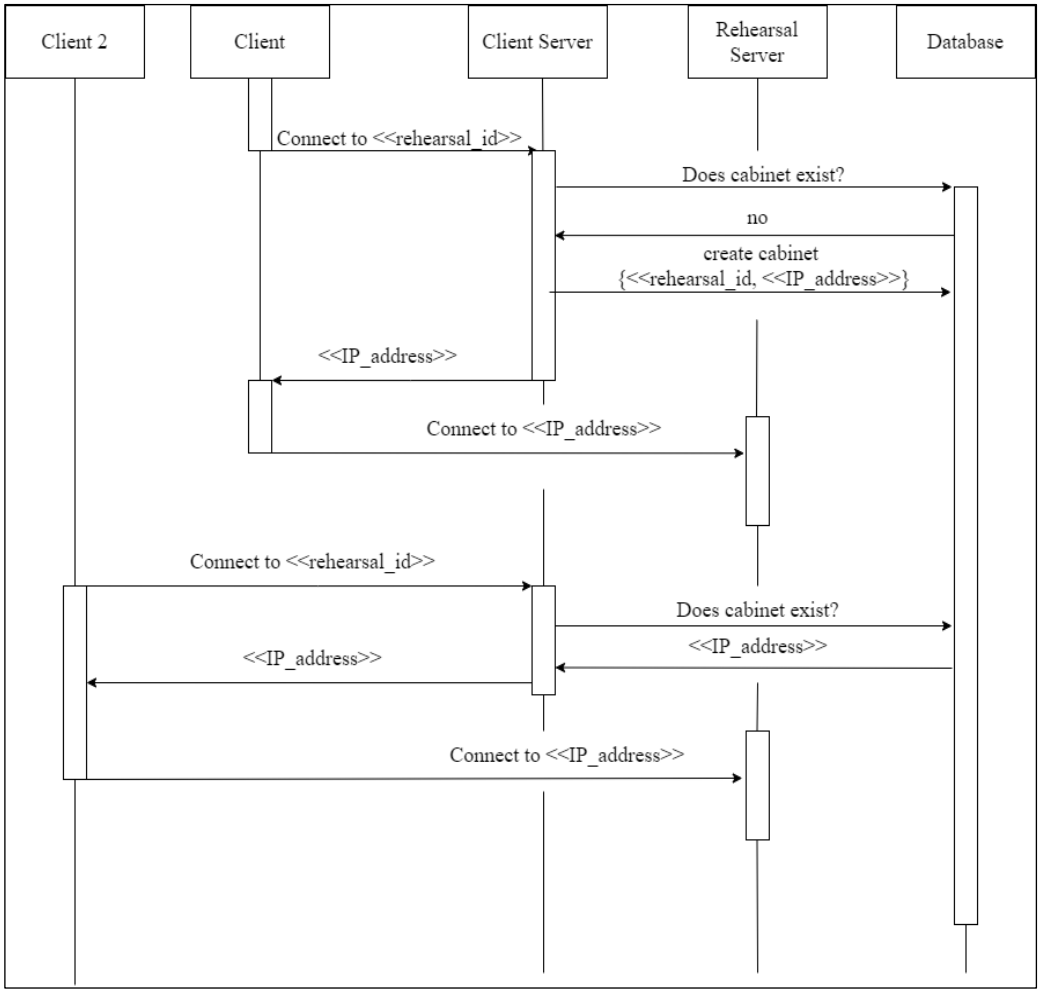

Рисунок 3.2 – Діаграма послідовностей (приєднання до репетиції)[Рисунок виконаний самостійно*]

Під час репетиції буде здійснено потоковий запис та передачу даних за допомогою декількох шарів:

а) клієнтський контролер першого користувача записує фрагмент даних (chunk) і надсилає його закодованим на сервер;

б) серверний контролер здійснює обробку отриманих даних:

1) якщо фрагмент отримано уперше, то створити відповідний файл та посилання на нього у базі даних;
2) записати фрагмент даних у створений файл;
3) перекодувати цей фрагмент даних тим засобом, який тестується;
4) зробити часову відмітку для формування тестових даних;
5) відправити перекодований фрагмент даних іншим клієнтам;

в) клієнтський контролер другого користувача отримує фрагмент даних, програє його, та робить часову відмітку.

3.3 План експерименту. Опис процесу тестування

Для перевірки придатності різних методів роботи зі звуком для використання у мережевому середовищі мають бути проведені два експерименти. Перший з них передбачає тестування запису звуку, що кодується в різних аудіоформатах, а саме OPUS, Vorbis та MPEG, і має довести теоретичне твердження про те, що використання аудіоформату OPUS є найбільш виправданим при створенні веб застосування для проведення репетицій. Для проведення цього експерименту не треба залучати другого користувача, достатньо просто повторити його для кожного з аудіоформатів. План експерименту наступний:

- початок запису звуку на клієнтському боці;
- передача даних на серверний бік після 5 секунд запису;
- збереження блоку даних на серверному боці;
- передача блоку даних на клієнтський бік;
- інтерпретація переданого блоку даних на клієнтському боці.

Другий експеримент має на меті перевірити можливість застосування методів фільтрації шумів та звукових ефектів під час запису чи після нього. Його можна реалізовувати із використанням того аудіоформату, який виявиться найкращим після аналізу результатів першого експерименту. План проведення цього експерименту має виглядати наступним чином:

а) підготовка до експерименту:
   1) користувач 1 заходить у систему та створює репетицію;
   2) користувач 2 заходить у систему та приєднується до репетиції;
   3) користувач 1 починає запис.

а) Проведення експерименту:
   1) користувач 1 програє музичну композицію. Клієнтські контролери формують фрагменти даних та відправляють їх на серверний бік;

2) серверні контролери записують фрагменти даних, перекодовують їх та відправляють на клієнтський бік;
3) клієнтські контролери приймають дані, перекодовують та програють їх.

в) аналіз результатів експерименту.

Зазначена послідовність дій має бути виконана для шумоочищення з використання Вінерівської фільтрації та шумоочищення фільтрацією низьких та високих частот. Окремо слід розглянути накладання ефектів, таких як дисторшн та «delay», щоб оцінити час на їх застосування і зробити висновок про можливість їх використовування під час проведення репетицій.

На якість отриманих у ході тестувань результатів можуть вплинути людський фактор та, власне, оточення, у якому буде проводитися запис звуку. Для того, щоби мінімізувати вплив цих факторів і зростання похибок необхідно для кожного набору методів, що тестуються, повторити тестування декілька разів.

3.4 Опис обраних технологій

Оскільки створений прототип системи має містити клієнт-серверну архітектуру, то відповідно це впливає на вибір технологій. Клієнтська частина застосування має бути реалізовано із використанням мов TypeScript та фреймворку Angular, який залишається другим за популярністю на сьогоднішній день[10], і дозволяє використати технологію WebRTC для обміну даними з серверною частиною системи. У випадку, коли застосування вказаних технологій не є доцільним чи можливим, їх можна замінити стандартними технологіями веб-розробки, а саме мовами HTML та JavaScript. Для збірки проекту має використовуватись менеджер пакетів NPM.

Для створення серверної частини буде використана мова програмування Java і фреймворк Spring, оскільки останній дозволяє використати принцип «Dependency injection» та легко змінювати метод обробки даних за допомогою конфігурації. Окрім цього, він містить багато компонентів для створення

мережевих застосувань і наразі є найпоширенішим фреймворком для мови Java[11].

Реалізовувати частину, яка пов’язана з обробкою даних, що приходять на сервер, можна за допомогою Java Sound API[12]. Java Sound API – це специфікація, яка забезпечує низькорівневу підтримку операцій з аудіо, таких як відтворення та захоплення (запис) аудіо, мікшування, секвенування MIDI та синтез MIDI[13]. Реалізація цієї специфікації наявна в усіх сучасних версіях мови програмування Java.

У першу чергу, слід виділити класи AudioInputStream та AudioSystem, які найчастіше зустрічаються у проектах, що використовують зазначену технологію. Перший клас дозволяє читати байтовий потік, представляючи його як потік семплів з урахуванням заданого формату. Дані про цей формат зберігаються у класі AudioFormat, який зберігає частоту дискретизації аудіо, кількість бітів у семплі, спосіб запису байтів та інше. Клас AudioSystem дозволяє легко здійснювати операції, пов’язані з читанням та записом файлів аудіо у файлову систему.

Окрім цього, Java Sound API дозволяє потоково обробляти дані, обертаючи їх в об’єкти SourceDataLine. Для даних, записаних заздалегідь, можна використовувати об’єкти класа Clip. Це може бути корисно у випадку, якщо один з виконавців пішов з репетиції до її завершення. Його звукова доріжка може бути збережена окремо, та відтворена одночасно з іншими.

Об’єкти SourceDataLine та Clip подаються на вхід мікшеру (об’єкт класу Mixer), який об’єднує ці доріжки та дозволяє змінювати звук. Для цього в Java Sound API існують спеціальні засоби, такі як Control. За допомогою об’єктів класу Control можна змінювати амплітуди вхідного та вихідного сигналів, а також впливати на реверберацію звуку – його повторення внаслідок відбиття звуку від поверхонь у закритих приміщеннях.

Зрештою, за допомогою класу TargetDataLine, звуковий сигнал може бути знов поданий у вигляді байтового потоку, щоб потім бути відтвореному на

пристрої, записаному у файл чи переданого мережею. Описаний цикл схематично зображений на рисунку 3.3.

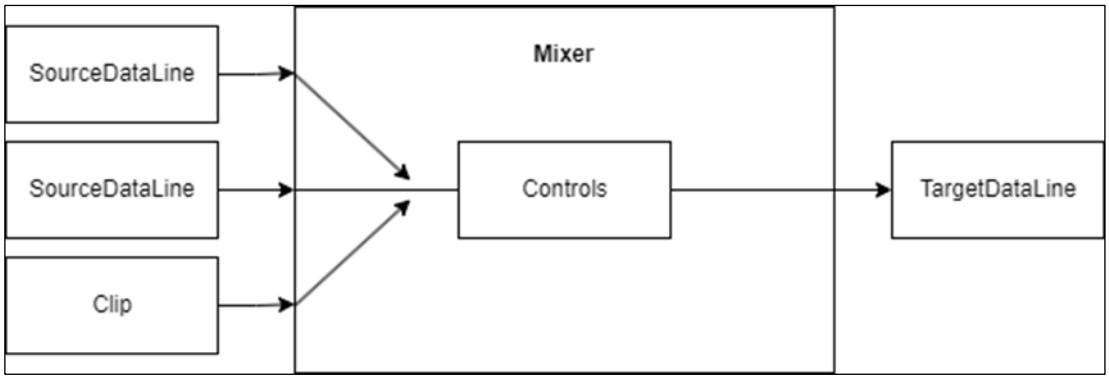

Рисунок 3.3 – Взаємодія об’єктів Java Sound API[Рисунок виконаний самостійно*]

Використання описаних технологій у поєднанні має дозволити створити такий прототип системи, який зможе приймати сигнал з клієнтської частини, оброблювати його на серверній частині, та передавати нові дані усім іншим користувачам, які підписані на встановлену розсилку.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Перевірка придатності аудіоформатів до використання у мережевому середовищі

Для тестування придатності аудіоформатів до використання у мережевому середовищі було розроблено окреме клієнт-серверне застосування. Серверна частина застосування являє собою компонент, написаний із застосуванням фреймворка Spring Boot, який надає WebSocket інтерфейс за для збереження та читання даних, закодованих у форматі Base64. Клієнтська частина, написана із застосуванням класичних веб технологій HTML, CSS і JavaScript, розгорталася на локальному сервері за допомогою платформи NodeJS і бібліотеки «lite-server». Без розгортання у цьому випадку обходитись не вийшло би, оскільки браузери вимагають окремого дозволу на використання мікрофона та камери, який закріплюється за сервером (див. рис. 4.1).

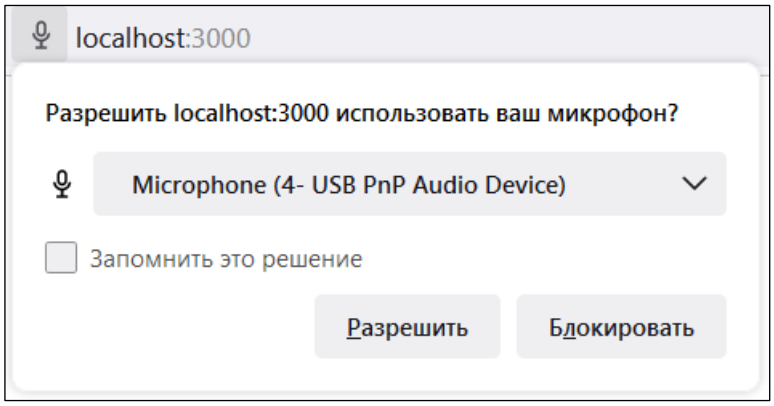

Рисунок 4.1 – Браузер Firefox вимагає дозволу на використання мікрофона[Рисунок виконаний самостійно*]

Для запису звуку у форматах, відмінних від OPUS, довелося застосувати сторонню бібліотеки «WebAudioRecorder», оскільки тільки OPUS (з медіаконтейнерами WEBM та OGG) нативно підтримується у браузерах Google Chrome та Firefox. Алгоритм програми виглядає наступним чином:

– ініціалізується об’єкт, що інкапсулює операції, пов’язані зі звукозаписом. Для формату OPUS це стандартний об’єкт класу MediaRecorder, для форматів Vorbis та MPEG-1 це об’єкт класу WebAudioRecorder;

– починається запис звуку. Одночасно запускається таймер, який через 5 секунд зупиняє запис і ставить першу часову відмітку;
– після зупинення запису формується об’єкт класу Blob, який містить закодовані дані. Ці дані додатково кодуються у форматі Base64 для строкового представлення і покращення подальшої передачі;
– дані, закодовані у Base64, передаються на сервер разом з метаданими – бітрейтом та зазначенням аудіоформату;
– отримуються дані з сервера, фактично відбувається емуляція їх отримання іншим користувачем;
– дані розкодовуються з формату Base64, формується посилання для скачування записаного треку.

Передача даних відбувається через веб сокети з використанням бібліотеки SockJS, яка допомагає утримувати двостороннє з’єднання, та текстового протоколу STOMP. У додатку Д наведено клас Stmp, який відповідає за встановлення такого з’єднання.

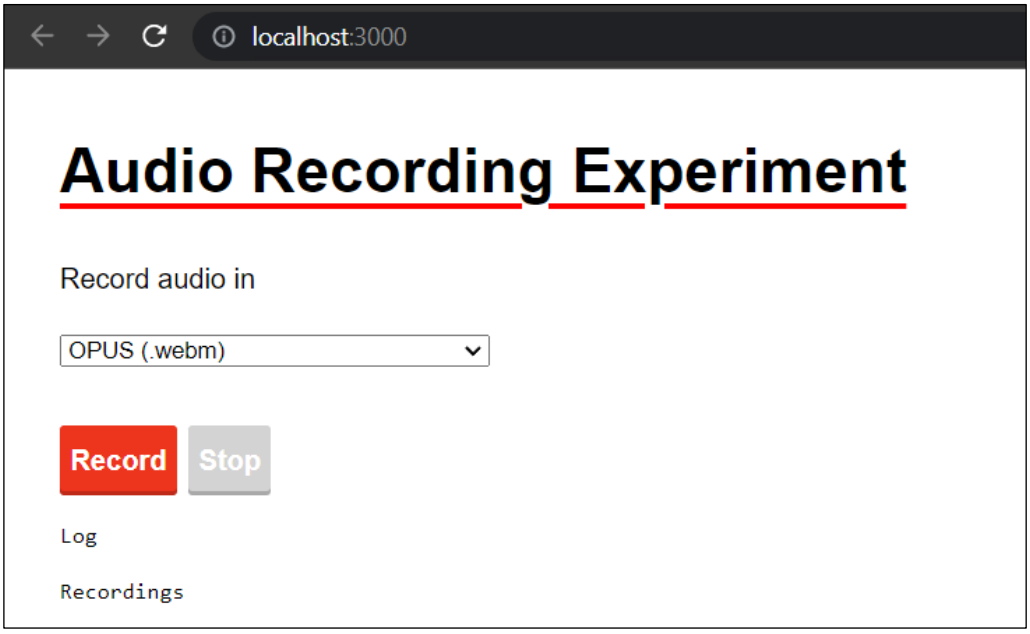

Рисунок 4.2 – Інтерфейс клієнтської частини застосування для тестування звукозапису у різних форматах[Рисунок виконаний самостійно*]

Серверна частина тестового застосування має два контролери, перший з яких слугує для встановлення першого підключення користувача до сервера, а

другий, під назвою RecordController, безпосередньо для прийому повідомлень та їх розповсюдження усім приєднаним клієнтам. Фрагмент коду, який відповідає за прийом повідомлень, наведено нижче:

```
private final SimpMessagingTemplate template;
private final RecordService recordService;

@MessageMapping("/record")
public void writeAudio(@Payload String recordEncoded,
      @Header(name = "bitsPerSecond") Integer bitsPerSecond,
      @Header(name = "audioFormat") String audioFormat) {
      RecordChunkDto dto = new RecordChunkDto();
      dto.setBitsPerSecond(bitsPerSecond);
      dto.setFormat(audioFormat);
      recordService.saveChunk(dto);
      template.convertAndSend("/test/media", recordEncoded);
}
```

Незалежно від формату даних, кожен блок, що записувався, мав однакові бітрейт, частоту дискретизації та два потоки (стерео).

У результаті проведення тестування були отримані наступні результати представлені у таблиці 4.1.

Таблиця 4.1 – Результати запису звуку в різних форматах[Таблиця виконана самостійно]

	OPUS	Vorbis	MPEG-1
Середній розмір файлу, кілобайт	77.5	73	97
Час від завершення запису до інтерпретації, мілісекунди	51	281	370

Суттєво відрізняється час, який витрачається на кодування, передачу, збереження та інтерпретацію даних. Розмір файлу, у який файли зберігалися на серверній частині, показує, що обсяг даних у цьому випадку не впливає суттєво на цей показник. Як показав аналіз часових відміток, головною причиною такої розбіжності стала різниця між часом завершення обробки об’єктом MediaRecorder

та WebAudioRecorder, що ставить під сумнів доцільність використання останнього для звукозапису під час проведення репетицій.

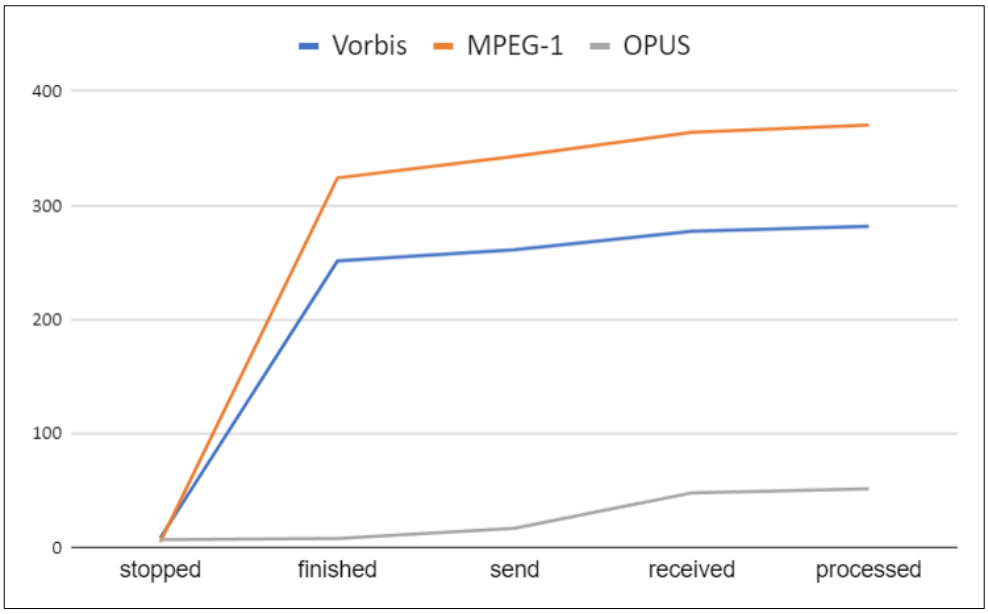

Рисунок 4.3 – Затримка між записом звуку та його відтворенням[Рисунок виконаний самостійно*]

Таким чином, проведене тестування доводить дві важливі тези:

– використання аудіоформатів, відмінних від OPUS, заради зменшення обсягів даних, що передаються та зберігаються, не є доцільним при створенні системи для проведення дистанційних репетицій;
– звукозапис у браузерах адаптований для використання аудіоформату OPUS, завдяки чому суттєво зменшується час на обробку даних, що записуються.

4.2 Реалізація прототипу системи для проведення репетицій

Для перевірки можливості застосування звукових ефектів та експериментальної перевірки ефективності методів шумоочищення була використана частина системи для проведення музичних репетицій, розроблена автором цієї роботи у 2021 році в ході роботи над кваліфікаційною роботою бакалавра.

4.2.1 Клієнтська частина програмного прототипу

Клієнтська частина системи реалізована за допомогою фреймворка Angular та мови програмування TypeScript. Принцип звукозапису реалізовано майже так само, як було описано у пункті 4.1 – він відбувається за допомогою класу MediaRecorder і передається на серверну частину за допомогою веб сокетів. Після завершення запису відбувається спроба завантажити записаний файл з сервера за допомогою REST API.

Під час проведення експерименту був використаний екран репетиції (див. рис. 4.4), який містить елементи управління репетицією та результати обробки даних.

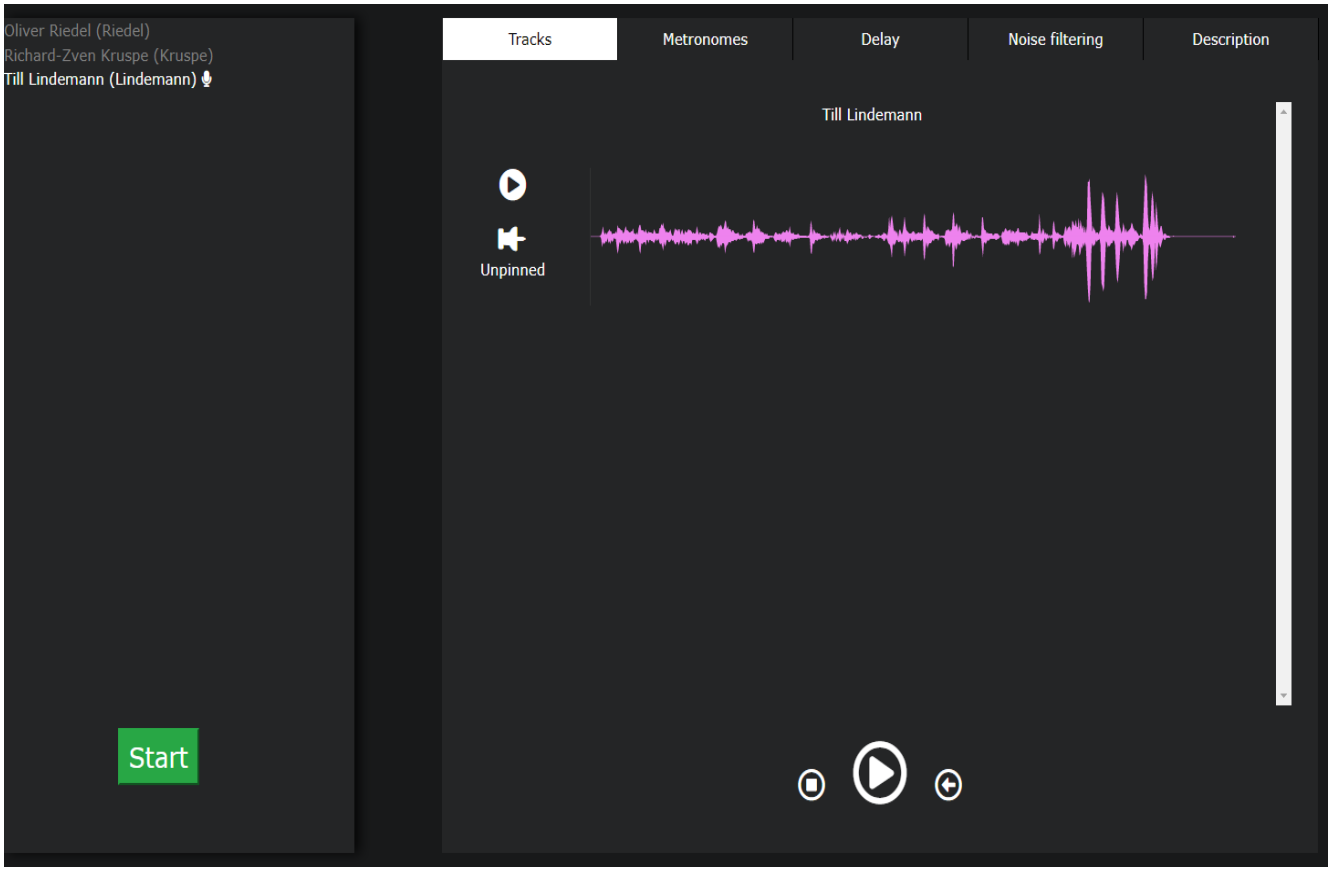

Рисунок 4.4 – Фрагмент екрану репетицій[Рисунок виконаний самостійно*]

Після натискання кнопки «Start» та завершення зворотнього відліку починається запис. На клієнтському боці за нього відповідає клас RecordService, фрагмент якого наведено на рисунку 4.5.

```
async startRecording(muted : boolean = false): Promise<void> {
  if (!this.mediaRecorder || this.mediaRecorder.state !== 'recording') {
    const audioConstraints = (this.filteringMethod === 'standard')
        ? {audio: {noiseSuppression: true}} : {audio: true};
    const stream = await navigator.mediaDevices.getUserMedia(audioConstraints);
    this.mediaRecorder = new MediaRecorder(stream, options: {
      audioBitsPerSecond: RecordService.BITS_PER_SECOND,
      mimeType: 'audio/webm'
    });
    this.mediaRecorder.ondataavailable = ((event: BlobEvent) => {
        this.sendChunk(event.data);
    });
    if (muted) {
      this.muteRecording();
    }
    this.mediaRecorder.start( timeslice: 500);
  }
}

private sendChunk(chunk: Blob): void {
  const reader = this.encoderService.encodeBase64(chunk);
  reader.onloadend = () => {
    const encodedSound = reader.result.toString();
    this.socketConnector.sendAudio(encodedSound, metadata: {
      firstRecord: this.firstRecord,
      bitsPerSecond: this.mediaRecorder.audioBitsPerSecond,
      lastRecord: this.finished,
      delayTime: this.delay,
      filterNoise: this.filteringMethod === 'frequency'
    });
```

Рисунок 4.5 – Фрагмент класу RecordService, який відповідає за звукозапис[Рисунок виконаний самостійно*]

Постобробка та передача блоку даних відбувається у методі sendChunk, який викликається одразу після того, як цей блок буде готовий. Разом із цим блоком надсилається додаткова інформація про нього – чи є він першим чи останнім, його бітрейт, чи потрібно встановлювати затримку чи проводити шумоочищення.

Для порівняння зручності такого підходу можна навести приклад звукозапису за допомогою Java Sound API, наведений у класі SoundRecorder (див. додаток Д). Цей підхід передбачає повну специфікацію усіх параметрів звуку в об’єкті класу AudioFormat, вибір інструменту для запису (Mixer) та більш низькорівневе управління байтовим потоком. Навіть без специфікації додаткових

параметрів та, власне, подальшої роботи з отриманими даними, такий спосіб є менш прозорим та значно багатослівнішим.

4.2.2 Серверна частина програмного прототипу

Серверна частина прототипу системи для проведення репетицій, яку було реалізовано із застосуванням фреймворка Spring Boot, надає два програмних інтерфейси. WebSocket інтерфейс використовується для приєднання клієнтів до репетиції, передачі подій щодо статусу репетиції та для передачі аудіоданих. Для отримання аудіофайлів з сервера використовується REST API, дані передаються за HTTP протоколом.

Після початку звукозапису серверна частина зберігає аудіофайли у файловій системі, а посилання на них – у базі даних PostgreSQL. Для зручності тестування база даних розміщується не безпосередньо на тому самому пристрої, як це було реалізовано раніше, а запускається у Docker контейнері. Архітектурно весь процес, від отримання повідомлення до завершення постобробки, є комплексним, і потребує реалізації кількох шарів – від контролера до репозиторіїв. Відповідна діаграма компонентів серверної частини наведена на рисунку 4.5.

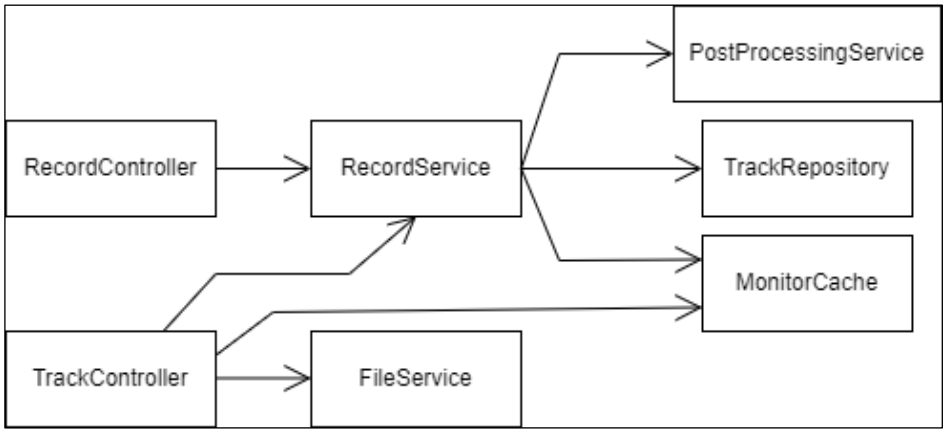

Рисунок 4.5 – Діаграма об’єктів для серверної частини[Рисунок виконаний самостійно*]

На жаль, обробка аудіо за допомогою Java Sound API не передбачає можливості використання даних, закодованих за допомогою формату OPUS. Через це після завершення запису виконується наступний алгоритм:

```
new File(filePath).renameTo(new File(filePath + ".tmp"));
    ProcessBuilder processBuilder = new ProcessBuilder("ffmpeg", "i",
"\"" + filePath + ".tmp\"", "\"" + filePath + "\"");
    Process process = processBuilder.start();
    process.waitFor();
    new File(filePath + ".tmp").delete();
```

Таким чином, за допомогою утиліти «ffmpeg», дані, закодовані у форматі OPUS, перекодовуються у формат WAV, придатний до обробки засобами Java Sound API. І, хоча WAV є нестиснутим форматом, що не найкращим чином впливає на обсяг та швидкість передачі даних, що ставить під сумнів доцільність його використання, він все ж таки містить амплітудно-частотну характеристику, яка може використовуватись для подальшого шумоочищення.

4.3 Програмне застосування звукових ефектів

Екран репетиції містить компонент «Delay», який дозволяє виставити затримку для аудіодоріжки (див. рис. 4.5).

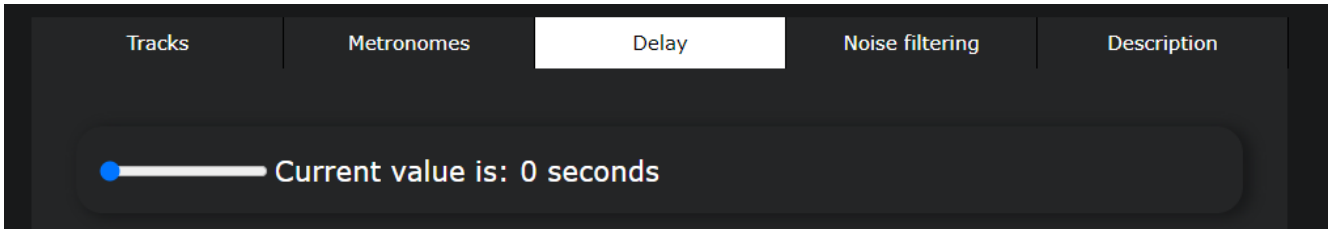

Рисунок 4.5 – Встановлення затримки[Рисунок виконаний самостійно*]

Встановлений у значення більше нуля, цей індикатор ініціює процес обробки даних на серверній частині системи після завершення звукозапису. Тривалість затримки встановлюється від 0 до 5 секунд з інтервалом у 0.25 секунди, хоча найчастіше вона не триває довше секунди.

Ефект досягається наступним чином: оригінальна доріжка копіюється, після чого за допомогою засобів пакету Java IO вона доповнюється на початку тишею з файлу затримки. Ці файли, згенеровані заздалегідь, зберігаються у ресурсних файлах сервера. Код генерації цих файлів наведено у додатку Д. Доповнена звукова доріжка зберігається на файловому сервері окремо.

На жаль, стандартні засоби Java Sound API не дозволяють об’єднувати звукові доріжки, тому такий байтовий потік було реалізовано самостійно. Сенс цього об’єднання полягає у простому складанні байтів, але формат вхідних даних має бути однаковим. Так, наприклад, не можна об’єднувати потоки даних, коли перший перший закодовано у signed форматі (який дозволяє кодування негативними числами), а другий – у unsigned. Після об'єднання новий байтовий потік зберігається в окремому файлі, який насамперед намагається знайти користувач після завершення запису. Код створення файлів для застосування ефекту «delay» наведено на рисунку 4.6.

```
File file = new File(filePath);
aist1 = AudioSystem.getAudioInputStream(file.toURI().toURL());
File file2 = new File(filePath);
aist2 = AudioSystem.getAudioInputStream(file2.toURI().toURL());
String silenceFilename = String.format("Silence-%d-%d.wav", (dto.getDelayTime()) / 4,
        (dto.getDelayTime() % 4) * 25);
File silenceFile = new File( pathname: "src/main/resources/media/silence/" + silenceFilename);
silence = AudioSystem.getAudioInputStream(silenceFile.toURI().toURL());

diplicatedStream = new AudioInputStream(
        new SequenceInputStream(silence, aist2), audioFormat,
         length: silence.getFrameLength() + aist2.getFrameLength());
File edited = new File( pathname: filePath.split( regex: "\\.wav")[0] + "-canon.wav");
AudioSystem.write(diplicatedStream, AudioFileFormat.Type.WAVE, edited);

diplicatedStream = AudioSystem.getAudioInputStream(edited.toURI().toURL());
List<AudioInputStream> aists = Arrays.asList(diplicatedStream, aist1);

mixer = new MixingAudioInputStream(audioFormat, aists);
File canonFile = new File( pathname: filePath.split( regex: "\\.wav")[0] + "-mixed.wav");
AudioSystem.write(mixer, AudioFileFormat.Type.WAVE, canonFile);
```

Рисунок 4.6 – Код створення файлів, у яких застосовано ефект «delay» [Рисунок виконаний самостійно*]

Отриманий результат гарно візуалізує спектрограма, на якій зображено оригінальну, доповнену, та об’єднані доріжки (див. рис. 4.7). На ній можна чітко помітити момент, у який оригінальна доріжка посувається вперед, а на об’єднаній доріжці гарно видно пикові значення амплітуди, коли дві ноти збігаються в часі.

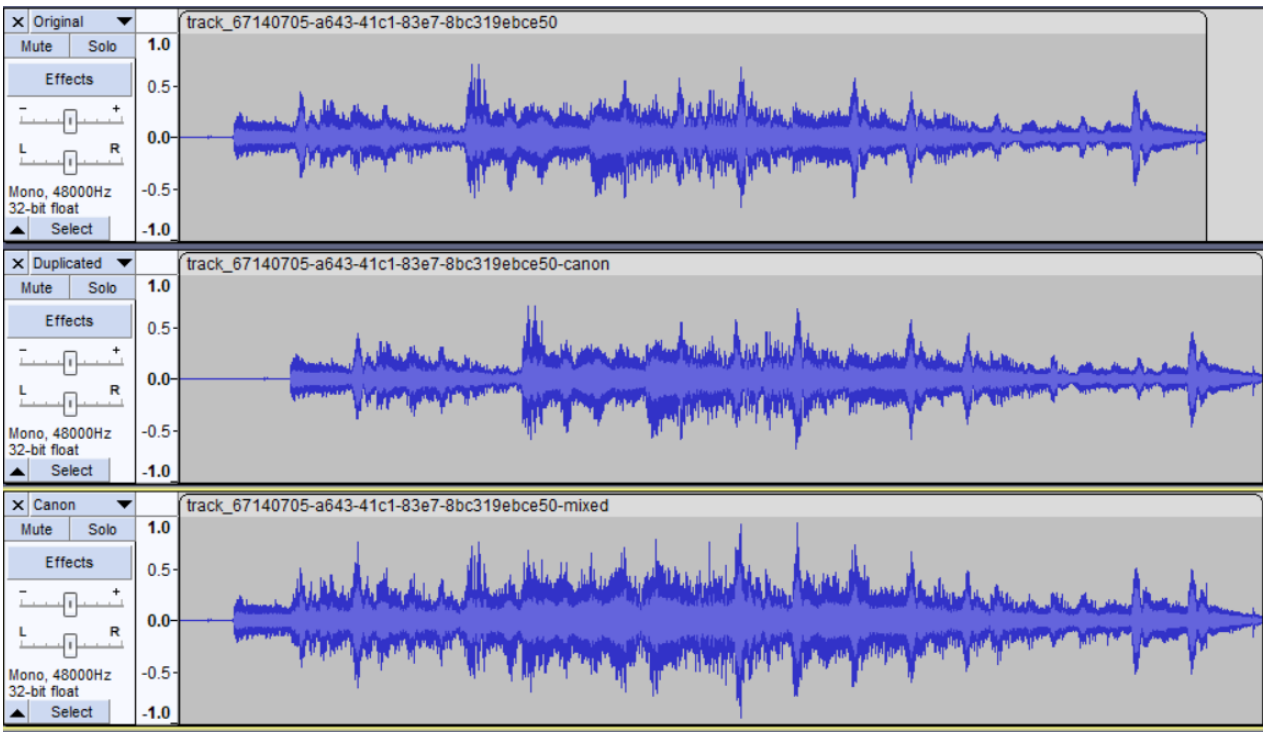

Рисунок 4.7 – Спектрограма, що візуалізує результат виконання обробки[Рисунок виконаний самостійно*]

Після завершення обробки користувач бажає якнайскоріше отримати результат. Найскорішим способом зробити це є ті самі веб сокети, про які вже йшлось раніше, проте організувати розсилку на всіх учасників після завершення обробки одного треку недоцільно. Більш того, якщо обсяг файлу буде достатньо великим, щоб перебільшити сконфігуровані ліміти для розміру повідомлення, то станеться помилка. Через це потік обробки даних і потік їх отримання були синхронізовані за допомогою стандартних методів Java. Алгоритмічно це виглядає наступним чином:

- перший потік (потік обробки) отримує метадані, у яких вказано те, що фрагмент даних, що записуються, є останнім;
- перший потік створює у пулі моніторів об'єкт, прив'язаний до назви файлу, який обробляється, і синхронізується на цьому об'єкті (моніторі);
- перший потік починає обробку;
- другий потік створюється після HTTP запиту на отримання файлу;

– другий потік отримує з пулу моніторів об'єкт, створений раніше першим потоком, і очікує його звільнення;
– перший потік завершує обробку файлу та звільняє монітор;
– другий потік синхронізується на моніторі та читає файл, обробку якого на цей час гарантовано завершено.

За допомогою бібліотеки «WaveSurfer» фінальна аудіодоріжка відображається на боці клієнта.

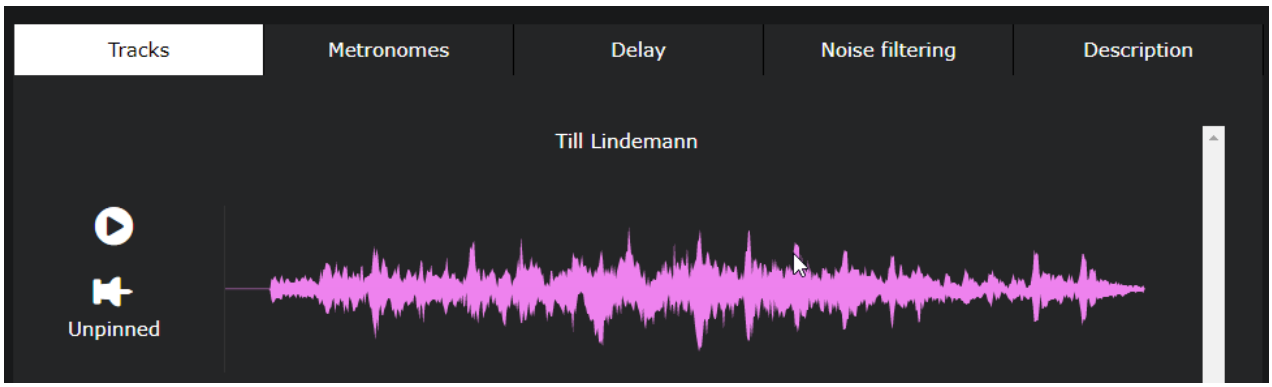

Рисунок 4.8 – Спектрограма, що візуалізує результат виконання обробки[Рисунок виконаний самостійно*]

4.4 Програмне застосування методів шумоочищення

Екран репетиції містить компонент «Noise filtering», який дозволяє виставити спосіб шумоочищення для звукозапису (див. рис. 4.9).

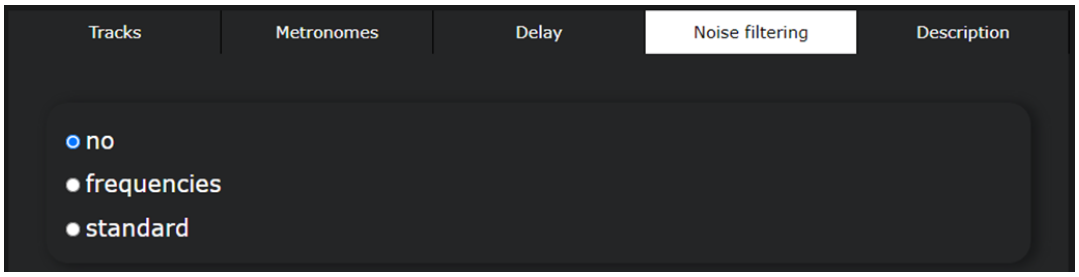

Рисунок 4.9 – Встановлення способу шумоочищення[Рисунок виконаний самостійно*]

Перша опція, обрана за замовчуванням, не передбачає застосування шумоочищення. Друга опція дозволяє увімкнути шумоочищення на боці сервера. Відповідний алгоритм починає діяти після завершення запису. Третя опція

дозволяє увімкнути шумоочищення на боці клієнта, яке застосовується безпосередньо при формуванні блоку даних для відправки.

Технологія WebRTC передбачає можливість застосування шумоочищення, який вмикається параметром конфігурації при налаштуванні об’єкта MediaRecorder (див. рис. 4.5), і саме це відбувається при виборі опції «standard». Для перевірки ефективності цього способу були записані дві схожі аудіодоріжки, одна з яких записувалася без шумоочищення, а друга із його застосуванням. Для запису використовувався стаціонарний шум телевізійних перешкод, а у якості корисного сигналу програвалася однакова гітарна композиція. Після запису звук вручну був обрізаний до 12 секунд та була нормалізована амплітуда обох доріжок. Результат цих операцій візуально наведено на рисунку 9.10. Візуально помітно, що у першому випадку кількість шумів значно вища протягом усього часу.

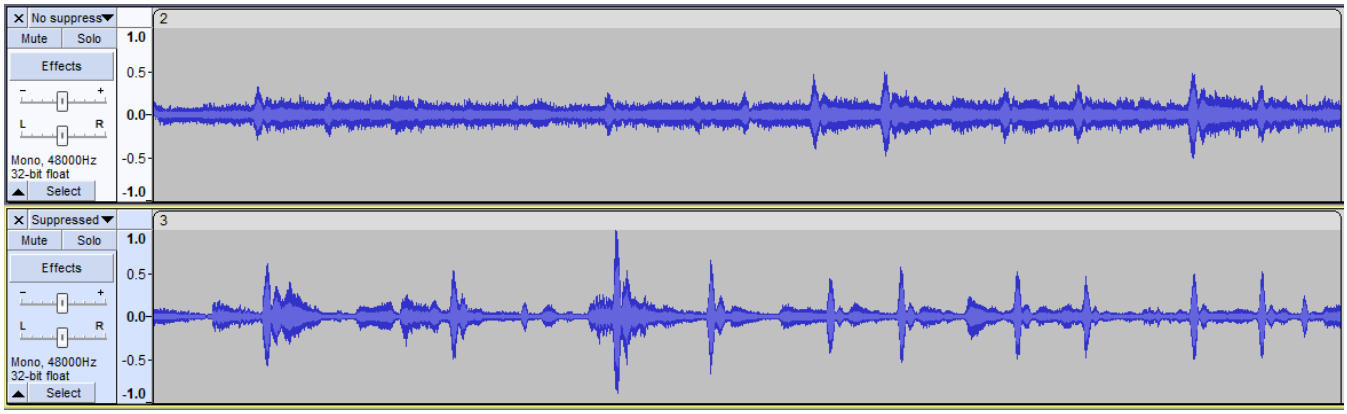

Рисунок 9.10 – Спектрограма, що візуалізує записані доріжки з шумом[Рисунок виконаний самостійно*]

Для того, щоб оцінити вміст шуму у сигналі, було реалізовано алгоритм, що базується на середніх значеннях амплітуди у певний період часу. Приблизний час, який займають піки амплітуди під час гри на гітарі, дорівнює приблизно 0.15 секунди. На проміжки, відповідні цьому часу, було поділено байтові потоки зазначених доріжок, оскільки ці байти саме зберігають амплітудно-частотну характеристику звуку.

Далі підраховується середнє значення амплітуди у кожному з періодів. Таких значень небагато, і це зумовлює поділ доріжки на кілька частин.

Сукупність тих частин, середня амплітуда яких зустрічалася найчастіше, можна вважати не корисною, оскільки стаціонарний шум заповнював їх повністю. Усі інші частини можна вважати корисними. Тоді відношення корисної частки до частки шуму буде показником якості. Код наведеного алгоритму наведено на рисунку 9.10.

```
//calculate average values - block characteristic
final byte[] buf = new byte[blockSize];
Integer[] averageValues = new Integer[bytes.length / blockSize];
for (int i = 0; i < bytes.length / blockSize; ++i) {
    audioStream.read(buf,  off: 0, buf.length);

    int sum = 0;
    for (byte b : buf) {
        sum += Math.abs(b);
    }
    int average = sum / buf.length;
    averageValues[i] = average;

    System.out.printf("Second %.2f - %.2f. Average: %d%n", i * 0.15, (i + 1) * 0.15, average);
}

//frequencies of amplitudes
Map<Integer, Integer> timesRepeat = new HashMap<>();
for (Integer average : averageValues) {
    if (timesRepeat.containsKey(average)) {
        timesRepeat.put(average, timesRepeat.get(average) + 1);
    } else {
        timesRepeat.put(average, 1);
    }
}

//define the most noising part
Integer maxRepeated = timesRepeat.entrySet().stream()
        .max(Comparator.comparingInt(Map.Entry::getValue))
        .get().getKey();

//obtain the result
double result = (averageValues.length - timesRepeat.get(maxRepeated)) / (double) timesRepeat.get(maxRepeated);
```

Рисунок 9.10 – Обчислення частки корисного сигналу по відношенню до шуму[Рисунок виконаний самостійно*]

Після виконання описаного алгоритму на вказаних доріжках, були отримані результати 1.05 для першої доріжки і 2.33 для другої, що показує зростання частки корисного сигналу на 122%.

Другий спосіб шумоочищення, який був використаний – це фільтрація верхніх і нижніх частот. Для отримання частотної характеристики з амплітудно-часової використовується дискретне перетворення Фур’є, яке розкладає байтовий

потік на складові комплексних чисел. Далі визначаються магнітуди – об'єднана частотно-амплітудна характеристика звуку. Магнітуди дозволяють визначити середній рівень шуму у доріжці, після чого залишається прибрати ці частоти з доріжки. Відповідний код отримання магнітуд наведено далі:

```
//Read file content
AudioFormat audioFormat = new AudioFormat( sampleRate: 44100.0f, sampleSizeInBits: 16,
        channels: 1, signed: true, bigEndian: false);
FileInputStream fis = new FileInputStream( name: "src/main/resources/no-supp.wav");
byte[] bytes = IOUtils.readAllBytes(fis);
ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
AudioInputStream audioStream = new AudioInputStream(bais, audioFormat,
        length: bytes.length / audioFormat.getFrameSize());

byte[] buf = new byte[4096];
int numberOfSamples = buf.length / audioFormat.getFrameSize();
Transform fft = FFTFactory.getInstance().create(numberOfSamples);
for (int i = 0; i < bytes.length / 4096; ++i) {

    audioStream.read(buf, off: 0, buf.length);

    float[] samples = decode(buf, audioFormat);
    float[][] transformed = fft.transform(samples);
    float[] realPart = transformed[0];
    float[] imaginaryPart = transformed[1];
    double[] magnitudes = toMagnitudes(realPart, imaginaryPart);
}
```

Рисунок 9.11 – Визначення магнітуд[Рисунок виконаний самостійно*]

У результаті виконання алгоритму було отримано нову доріжку (див. рис. 9.12), у якій відношення частки корисного сигналу до частки шуму склало 2.48, що показує зростання частки корисного сигналу на 136%. Тим не менш, при цьому було втрачено частину корисних даних –амплітуда для непікових значень сильно зменшилася, що каже про те, що доцільно використовувати такий фільтр для обробки мовної інформації, а не музикальних композицій.

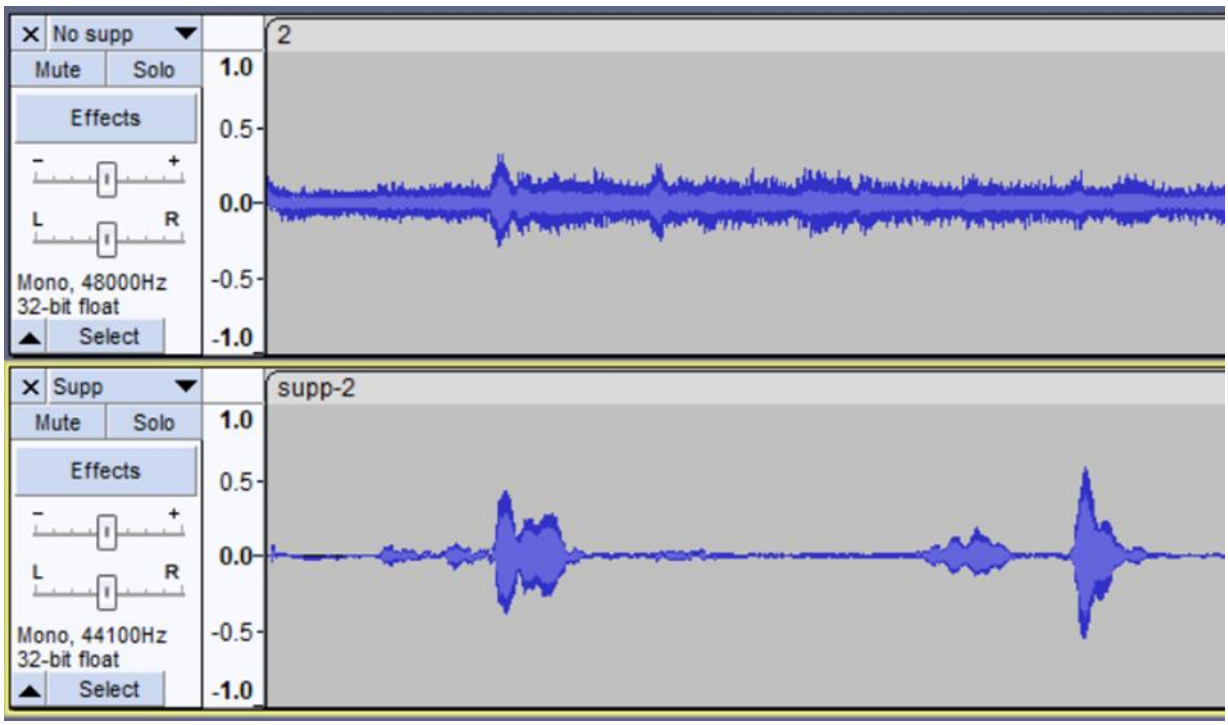

Рисунок 9.12 – Результат обробки звуку шляхом фільтрації нижніх частот[Рисунок виконаний самостійно*]

Таким чином, можна зробити висновок про те, що стандартні засоби шумоочищення, які застосовується у технології WebRTC, надають кращі можливості для застосування під час розробки системи для проведення музикальних композицій, ніж самостійно розроблений метод фільтрації частот.

ВИСНОВКИ

При виконанні кваліфікаційної роботи магістра було розглянуто декілька методів збереження, шумоочищення та передачі звуку, наведено приклади музикальних ефектів, які можна реалізувати програмно без допомоги додаткових фізичних пристроїв. Були висунуті теоретичні припущення щодо ефективності використання цих методів для розробки систем для проведення музичних репетицій.

Для доведення цих припущень було проведено два експерименти, спроєктовано та розроблено прототип системи для проведення музичних репетицій, який має клієнт-серверну архітектуру і забезпечує можливості проведення звукозапису кількома клієнтами одночасно. За допомогою цього застосування було отримано наступні результати:

- аудіоформат OPUS краще підходить для запису звуку у веб застосуваннях ніж інші популярні формати , таких як Vorbis або MPEG-1, за рахунок кращої адаптації для використання з боку сучасних браузерів;
- класичні методи шумоочищення, які використовуються у засобах WebRTC, діють краще, ніж реалізований у розробленому прототипі системи алгоритм фільтрації нижніх та верхніх частот;
- реалізація специфікації Java Sound API, наявна в бібліотеці класів Java, дозволяє програмно проводити обробку звуку і застосовувати звукові ефекти. Тим не менш, її можливості обмежені рядом форматів та функцій, за рахунок чого при розробці програмного забезпечення доводиться широко використовувати сторонні бібліотеки та системні утиліти.

Результати цієї кваліфікаційної роботи можуть бути використані для подальших досліджень у сфері звукозапису, наведені практичні прийоми та алгоритми можуть допомогти у створенні нових систем та застосунків для проведення дистанційних репетицій у режимі реального часу.

Кваліфікаційна робота магістра має широкі перспективи для проведення подальших досліджень у зазначеній предметній області. Залишається ряд питань, які залишилися нерозглянутими чи неперевіреними:

- дослідження ефективності методів шумоочищення з використанням нейронних мереж та фільтру Калмана;
- порівняння швидкості обробки та передачі даних при здійсненні звукозапису з мобільних чи десктопних клієнтів;
- практична перевірка ефективності методів шумоочищення при застосуванні файлів шаблонів, таких як файли у форматі MIDI (див. п. 3.1.2).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Google Meet vs Zoom in 2022. URL: https://www.talkinfotech.com/computer/google-meet-vs-zoom-in-2022 (date of access: 16.03.2023).

2. JamKazam Review: Collaborate Remotely. producelikeapro.com. URL: https://producelikeapro.com/blog/jamkazam-review (date of access: 16.03.2023).

3. Воронкін О. С. Суб'єктивні та об'єктивні характеристики звуку. Фізико математична освіта. 2017. № 2.

4. Weinzierl S., Lerch A., Wiesener C. Adaptive Noise Reduction for Real-time Applications. Proceedings of the 128th Audio Engineering Society Convention: Міжнар. наук. конф., Лондон, 22 May 2012.

5. Черняхов А., Каук В. Використання дискретного перетворення фур'є на прикладі мобільного застосунку для запису та аналізу звуку. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : Міжнар. наук. конф., 27 квіт. 2022 р. С. 185.

6. Назаров О.С., Яровий М.В. Частотний аналіз в задачах розпізнавання звуку з використанням нейроних мереж. Сучасні аспекти та перспективи розвитку науки. 16 квітня 2021 м. Кропивницький, Україна, с. 48-50.

7. Обзор методов улучшения речи и шумоподавления: от классики к SotA. https://habr.com/ru/company/ru_mts/blog/584308. URL: https://habr.com/ru/company/ru_mts/blog/584308 (дата звернення: 30.11.2022).

8. Сравнение цифровых аудиоформатов. URL: https://ru.wikipedia.org/wiki/Сравнение_цифровых_аудиоформатов (дата звернення: 12.12.2022).

9. Філімончук Т. В. Аналіз актуальності використання шаблону MVC при розробці сучасних веб-застосунків / Т. В. Філімончук, О. С. Настенко // Дев'ята міжнародна науково-технічна конференція «Проблеми інформатизації». Т.2. – Черкаси - Харків - Баку - Бельсько-Бяла. – 18-19 листопада 2021 р. – С. 95.

10. Top JavaScript Frameworks and Technology 2023. medium.com. URL: https://medium.com/javascript-scene/top-javascript-frameworks-and-technology-2023-4e4a06d6be93 (date of access: 27.03.2023).

11. Walls C. Spring in Action, Sixth Edition. Manning, 2022. 520 p.

12. R. G. Baldwin Java Sound, Getting Started, Part 1, Playback. URL: https://www.developer.com/java/java-sound-getting-started-part-1-playback (date of access: 27.03.2023).

13. Java Sound API. URL: https://www.oracle.com/java/technologies/java-sound-api.html (date of access: 27.03.2023).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

1. Черняхов А., Каук В. Використання дискретного перетворення фур'є на прикладі мобільного застосунку для запису та аналізу звуку. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління : Міжнар. наук. конф., 27 квіт. 2022 р. С. 185.

2. Назаров О.С., Яровий М.В. Частотний аналіз в задачах розпізнавання звуку з використанням нейроних мереж. Сучасні аспекти та перспективи розвитку науки. 16 квітня 2021 м. Кропивницький, Україна, с. 48-50.