МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроніки
Центр післядипломної освіти
Кафедра Програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження методів запису та аналізу звуку на смартфоні

Виконав:
студент 2 курсу, групи ІПЗздм-20-1
Черняхов А.В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-наукова

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

Допускається до захисту

Зав. кафедри _____ Дудар З.В.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет	Центр післядипломної освіти
Кафедра	Програмної інженерії
Рівень вищої освіти	другий (магістерський)
Спеціальність	121– Інженерія програмного забезпечення
	(код і повна назва)
Тип програми	освітньо-наукова програма
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«___»_____ 2022 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Черняхову Андрію Вікторовичу

(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження методів запису та аналізу звуку на смартфоні» затверджена наказом університету від «__»____ 2022р. №___

2. Термін подання студентом роботи до екзаменаційної комісії «__»____ 2022р.

3. Вихідні дані до роботи: технічне завдання, календарний план, методичні вказівки.

4. Перелік питань, що потрібно опрацювати в роботі: мета роботи, аналіз предметної галузі і постановка задачі, дослідження методів запису та аналізу звуку, вивчення можливості їх використання у мобільному застосунку .

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі	27.01.2022	виконано
2	Постановка задачі	31.01.2022	виконано
3	Формування вимог	05.02.2022	виконано
4	Дослідження	17.02.2022	виконано
5	Проектування	27.02.2022	виконано
6	Розробка ПЗ	15.04.2022	виконано
7	Підготовка пояснювальної записки	01.05.2022	виконано
8	Перевірка на академічний плагіат	11.05.2022	виконано
9	Нормоконтроль		
10	Рецензування		
11	Занесення диплому в електронний архів		
12	Попередній захист		
13	Допуск до захисту у зав. кафедри		

Дата видачі завдання 24 січня 2022р.

Студент _____
(підпис)

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

РЕФЕРАТ / ABSTRACT

Кваліфікаційна робота магістра містить: 58 с., 39 рис., 12 джерел.

ЗАПИС, ЗВУК, АНАЛІЗ, ДОСЛІДЖЕННЯ, АНАЛІЗ МЕТОДІВ, МЕТОДИ ЗАПИСУ, МЕТОДИ АНАЛІЗУ

Об’єктом дослідження є методи запису та аналізу звуку на смартфоні.

Метою роботи є дослідження методів запису та аналізу звуку на смартфоні.

Результатом роботи є проведення дослідження запису та аналізу звуку на смартфоні.

RECORDING, SOUND, ANALYSIS, RESEARCH, ANALYSIS OF METHODS, RECORDING METHODS, ANALYSIS METHODS

The object of research is the methods of recording and analyzing the sound on a smartphone.

The aim of the research is to study the methods of recording and analyzing sound on a smartphone.

The result of the research is a study of recording and analysis of sound on a smartphone.

Умови публікації пояснювальної записки

Я, Черняхов Андрій Вікторович, студент групи ІПЗздм-20-1, здобувач вищої освіти на другому (магістерському) рівні, кафедра програмної інженерії, заявляю: моя кваліфікаційна робота на тему «Дослідження методів запису та аналізу звуку на смартфоні», що буде представлена до ЕК для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений 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

На даний момент високий рівень людської безпеки досягається з допомогою надійного програмного та апаратного забезпечення. Останніми роками почастішали випадки використання алгоритмів розпізнавання, обробки та систем локалізації виявлення злочинів, що сприяє миттєвому реагуванню на них правоохоронними органами. Ці технології є доступними у зв'язку з легкими конфігураціями мікрофонів, що дозволяє визначити розташування джерела звуку з високою точністю.

Однак наразі не спостерігається прогресу в медичній сфері, особливо у відношенні досліджень, що спрямовані на звичайних людей. Прикладом може слугувати поверхнева оцінка дихання особи, що в свою чергу могло б допомогти у виявленні різних захворювань. Уявімо ситуацію, що кожна людина в змозі провести описаний раніше аналіз свого дихання маючи лише смартфон! Це дозволить вжити превентивні міри, запобігти розвитку тяжких ускладнень та навіть врятувати життя.

Існуючі алгоритми та методи запису та аналізу звуків є доволі просунутими та сучасними, але, на жаль, не досконалими. Усі вони стикаються з достатньо розповсюдженими загальними проблемами, які до цих пір не є вирішеними.

Отримані результати досліджень за даною проблематикою можна буде інтегрувати в можливість попередньої поверхневої медичної діагностики дихання, що є надзвичайно важливим внеском у повсякденне життя людей, котрі мають проблеми з легенями і яким конче необхідно слідкувати за своїм станом здоров'я. Також одним із варіантів застосування підсумків роботи може слугувати можливість удосконалення методів дистанційного навчання [1]. Це дозволило б оптимізувати комунікацію між сторонами навчального процесу. По-перше, надійний метод запису звуку дозволить покращити якість лекційних аудіо матеріалів і допоможе в їх подальшому аналізу. У той же час студент матиме змогу прослухати лекції у вигляді подкастів, що сприятиме кращому запам'ятовуванню матеріалу. Другою варіацією імплементації результатів роботи є можливість

проводити зріз знань у студентів, адже вони матимуть змогу оптимізовано записувати власне аудіо та відправляти педагогу на перевірку. Це одночасно дозволяє провести перевірку знань та підтвердити особу студенту по його голосу. Виходячи з усього вищесказаного, можна дійти висновку, що дана робота є доволі актуальною і має широкі перспективи у майбутньому.

У даній роботі взято за мету дослідження існуючих систем та методів запису та аналізу звуку за допомогою смартфону та створення нового чи покращення функціональних можливостей існуючих методів на базі програмного підходу, що дозволить проводити запис та аналіз більш чітко навіть при досить гучних фонових шумах на невисокоякісних мікрофонах. Для досягнення мети роботи необхідно:

- розглянути та проаналізувати існуючі системи та їх методи запису та аналізу звукових даних;

- на основі проведеного аналізу виявити проблеми і недоліки розглянутих систем та методів.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Загальний аналіз звуку

Загально можна сказати, що звук є коливальним рухом певних маленьких частинок, що розповсюджуються у хвилеподібному вигляді у середовищах рідин, твердих тіл чи газів, а також можуть сприйматися слуховим апаратом. Під звуком можуть мати на увазі й коливання, які збуджують сенсорно-слухову систему тварин та людей [2]. У розглянутому випадку говориться про так зване збурення, що з відповідною частотою може поширюватися в одному із трьох раніше наведених середовищ (рідини, гази, тверді тіла). Наш слуховий апарат в змозі розпізнавати коливання лише у відносно маленькому частотному діапазоні. Якщо говорити про переважну більшість тварин, то варто відмітити що їхній апарат здатен сприймати коливання звуку в значно більшому діапазоні частот. Загалом цей науковий термін також має значення процесу поширення вібрації у відповідних середовищах з різними фізичними властивостями, в яких сила пружності прагне відновити початкове положення частинок, які були збуджені в початковому стані спокою. Хвильові збудження, що в принципі і є тим самим звуком, об'єктивно реальні і існують незалежно від того, чи сприймаються вони кимось чи ні. Вивчення законів сприйняття, утворення і поширення звуку в різних середовищах — це галузь наукового знання, яка називається акустикою.

Майже всі існуючі явища природи супроводжуються відповідними звуками, які сприймаються слуховими апаратами тварин і людей, а також служать для забезпечення спілкування та спрямування в просторі [3]. Існує певна особливість у сприйнятті звукових коливальних рухів слухом людини, що поділяє різноманіття звуків на гармонійні (приємні) звуки та дратівливі чи небажані. До першого типу можна віднести, наприклад, спів пташок, звуки ліри та інші музичні гармонічні звуки. Дратівливі звуки мають специфічне наповнення у певному спектрі, і зазвичай розпізнаються, як шум.

Шум або акустичний шум – це вібрації частинок з навколишнього середовища, які людське вухо сприймає як небажані сигнали. У акустиці: шум

розуміється, як нестабільні або випадкові звукові коливання, що характеризуються випадковими змінами амплітуди та частоти [4].

Шуми класифікують відповідно до джерела:

– аеродинамічні шуми виникають у газовому середовищі;
– гідродинамічні виникають у рідинному середовищі;
– електромагнітні виникають в результаті впливу змінних магнітних сил, що викликають небажані коливання електромеханічних елементів у пристроях;
– механічні виникають в результаті вібрацій поверхні обладнання чи машин, або конструкцій.

Також їх можливо класифікувати за діапазоном частот:

- низькочастотні до 400 Герц;
- средньочастотні від 400 до 1000 Герц;
- високочастотні понад 1000 Герц.

У деяких вузькоспеціалізованих галузях таких, як електроніка та акустика існують поняття про кольори шумів, за яким певний колір присвоюється шумовому сигналу в залежності від його властивостей. В акустиці прийнято загалом поділяти спектри шумів за наступними кольорами: білий шум, рожевий шум, червоний шум та сірий шум. Інколи виділяють й інші різновиди [5].

Білий шум — постійний шум, спектральні складові якого рівномірно розподілені по всьому діапазону частот (рисунок 1.1). однак білим шумом можна вважати будь-який шум, спектральна щільність якого однакова (або майже однакова) у заданому діапазоні частот. Свою назву він отримав від білого світла, до якого входять електромагнітні хвилі частот усього видимого діапазону електромагнітного випромінювання [6].

Рожевий шум – це шум, спектральна щільність якого змінюється з частотою f за законом 1/f (рисунок 1.2). Рожевим шумом іноді називають будь-який шум, спектральна щільність якого зменшується зі зменшенням частоти.

Червоний шум (броунівський шум) – шум також відомий як броунівський шум, оскільки його зміна звуку від моменту до моменту є випадковою. Спектральна щільність червоного шуму пропорційна 1/f2, де f – частота (рисунок 1.3) [7].

Сірий шум – це шумовий сигнал, що відповідає акустичній кривій постійної гучності на всіх частотах, тобто для людського вуха він взагалі має однакову гучність (рисунок 1.4)[8].

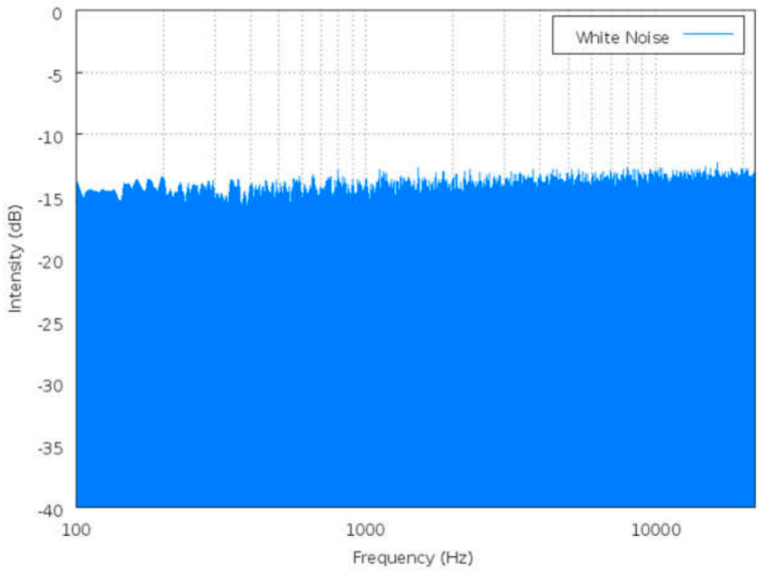

Рисунок 1.1 – Спектр білого шуму

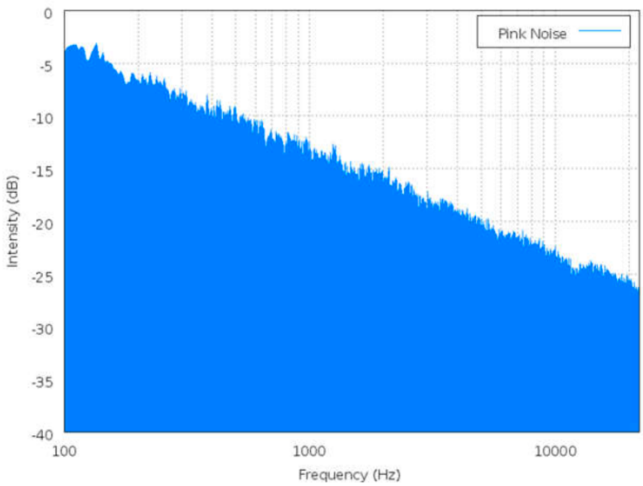

Рисунок 1.2 – Спектр рожевого шуму

На розпізнавання звукових даних також впливають такі явища, як:

– реверберація (зазвичай утворюється у великих приміщеннях і характеризується багаторазовим відбиттям звуків від стін);

– дифракція (викликана накладанням звукових хвиль одна на одну при обході перешкод);
– заломлення та інші явища, які менше впливають на розпізнавання звуків у полі розробленого методу через умови та середовище, в яких вони виникають.

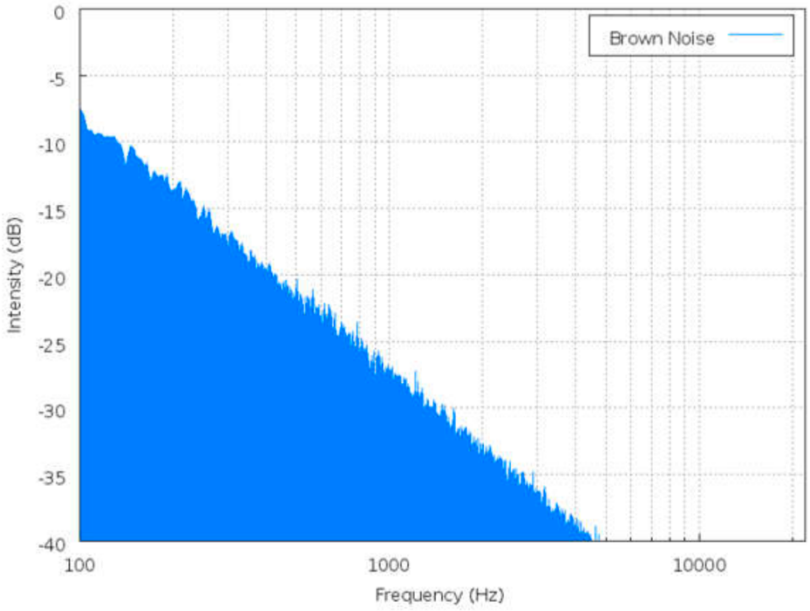

Рисунок 1.3 – Спектр червоного шуму

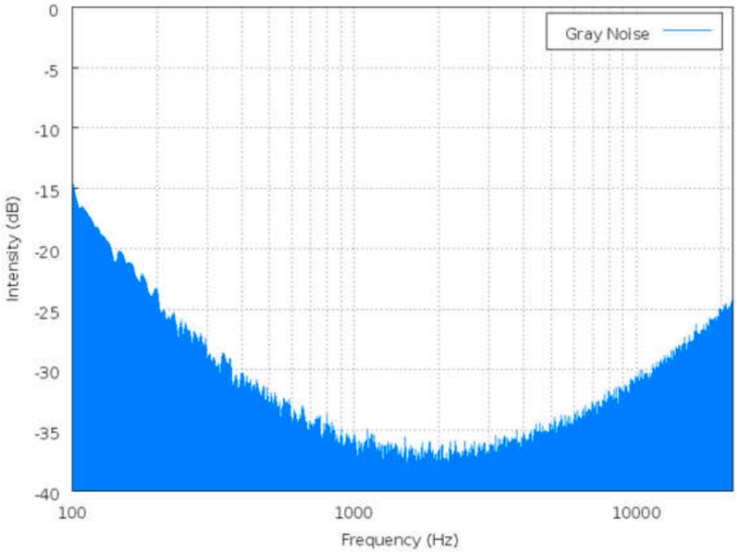

Рисунок 1.4 – Спектр сірого шуму

Хоча запис і аналіз різноманітних звуків і людської мови все більше інтегрується в наше повсякденне життя, і все розвивається, що значно спрощує: запам'ятовування інформації, аналіз різноманітних природних явищ, пошук і запис

різноманітних музичних творів спілкування за допомогою голосового введення, пошук інформації про зразки звуку, керування величезною кількістю різних пристроїв і навіть для розваги. На жаль, точність і якість запису може бути досить нетривіальним завданням, оскільки висока якість вимагає багато пам'яті. Звичайно, існують алгоритми стиснення і навіть стиснення без втрат, але, на мій погляд, є ще багато можливостей для розвитку. Тим не менш, кінцеві користувачі системи запису даних також очікують високу ефективність та оптимальні витрати ресурсів.

1.2 Постановка задачі

Для оптимального дослідження варіантів запису та аналізу звуку на смартфоні необхідно створити мобільний додаток для детального вивчення можливостей мобільних операційних систем. Рішення повинно бути легкодоступним, швидким та мобільним тому створення саме мобільного застосунку є найбільш оптимальним рішенням.

Для початку роботи над застосунком необхідно визначити вимоги до нього. Під вимогами мають на увазі сукупність властивостей, які необхідно буде реалізувати у мобільному додатку. Метою роботи є дослідження методів запису та аналізу звуку на смартфоні. Результатом роботи повинен стати мобільний додаток де у користувача буде можливість з допомогою вбудованого мікрофону записати звук та провести поверхневий його аналіз. Це допоможе дослідити і знайти найоптимальніший шлях запису та аналізу звукових даних.

Метою проекту є дослідження методів запису та аналізу звуку на смартфоні, реалізація мобільного додатку, набуття корисних навичок в галузі проектування, планування, дослідження, вивчення, розробки, тестування та підтримки мобільних додатків.

Для проекту було поставлено наступні задачі:

– провести аналіз предметної галузі;
– дослідити метода запису звуку;
– дослідити методи аналізу звуку;

– спроектувати архітектуру;
– описати прийняті рішення;
– розробити інтерфейс мобільного додатку;
– розробити мобільний додаток на основі поставлених вимог;
– провести тестування.

Інтерфейс мобільного додатку повинен бути розроблений з урахуванням останніх сучасних трендів та технологій користувацького досвіду та користувацького інтерфейсу. Інтуїтивна зрозумілість для користувача повинна бути невід’ємною частиною мобільного додатку.

Основною задачею, що постає перед системою є саме дослідження методів запису та аналізу звуку на смартфоні.

Мобільний застосунок повинен мати добре продуману та прораховану архітектуру, щоб додавання окремого модуля зберігало можливість інтеграції і не руйнувало вже існуючі компоненти додатку.

Програмна система обов’язково повинна бути стійкою до будь-якого зовнішніх втручання. Всі записані аудіо-файли повинні зберігатися лише на смартфоні, нікуди не передаватися та додатково не піддаватися аналізу від сторонніх застосунків.

2 ДОСЛІДЖЕННЯ МЕТОДІВ І АЛГОРИТМІВ РОБОТИ ЗІ ЗВУКОМ

2.1 Дослідження форматів запису звуку

При вирішенні задачі запису чи аналізу звуку сам процес має відбуватися в режимі реального часу. Це передбачає багаторазове повторення трьох операцій:

– запис порції звукових коливань;
– обробка отриманих значень;
– виведення результату.

Для зниження часу аналізу звукових даних необхідний аудіо-формат, який дозволить максимально швидко обробляти дані. Формат представлення звукових даних в цифровому вигляді залежить від способу квантування аналогово-цифровим перетворювачем.

Формати аудіо-файлів – це цифрові стандарти для зберігання звукової інформації. Необроблені дані в потоці аудіо з аналого-цифрового перетворювача у аудіо-інтерфейсі кодуються за допомогою техніки, яка називається PCM або імпульсно-кодовою модуляцією[9]. Аудіо PCM потрібно організувати у файл, щоб була можливість працювати з ним або відтворювати його в системі. Різні формати аудіо-файлів використовують різні контейнери та різні методи стиснення даних для організації потоку PCM. Кожен формат представляє ту саму інформацію в різних розмірах зберігання або рівнях якості. На додаток до цього, деякі формати аудіо-файлів містять метадані, які надають інформацію про файл або його вміст.

Існує два основних типи аудіо-файлів — формати файлів без втрат і формати файлів із втратами. Різниця між ними пов’язана зі стисненням даних. Стиснення даних означає, що файли займають менше місця на жорсткому диску. Це не те саме, що стиснення динамічного діапазону, яке використовується у виробництві музики.

Деякі методи стиснення даних зменшують файл, але зберігають 100% інформації в необробленому звуковому потоці. Вони відомі як стислі формати без втрат.

Інші типи стиснення працюють, усуваючи дані в аудіо, які не роблять великого впливу на звук. За допомогою цього методу деяка інформація викидається, тому вони відомі як стиснені формати з втратами.

Існують інші формати аудіо-файлів, де стиснення даних не використовується. Це так звані нестиснені аудіо-формати. Ці типи файлів діють як контейнер для необроблених аудіоданих, жодним чином не зменшуючи їх розмір чи якість. Це найбільші файли для роботи, але вони забезпечують найвищий рівень деталізації аудіо-інформації.

Нестиснені аудіо-файли – це тип, який найчастіше використовується для запису та мікшування музики в DAW. Незважаючи на це, нестиснені аудіо-файли також мають різні рівні якості. Вони засновані на точності та точності, з якою аналоговий звуковий сигнал був перетворений в цифровий. Чим вища частота дискретизації та розрядність, що використовується, тим більше інформації фіксується в процесі перетворення. Розрядна глибина представляє точність АЦП/ЦА перетворювача для вимірювання амплітуди або рівня гучності сигналу. Можливо уявити це як кількість галочки на лінійці — чим ближче вони розташовані, тим рідше вимірювання потраплятиме між двома мітками. Частота дискретизації означає, скільки разів за секунду виконується вимірювання. Вища частота вибірки означає більше індивідуальних вимірювань.

Стиснуті аудіо-файли з втратами можуть бути закодовані на різних рівнях якості. Якість цього формату файлу визначається бітрейтом або кількістю даних, закодованих за секунду. При нижчих налаштуваннях бітрейту стислі файли будуть набагато меншими, але можуть звучати гірше.

Раніше це було необхідно, оскільки носії даних мали обмежену ємність, а мережі не могли легко передавати великі файли. Але сховище та пропускна здатність не є такою проблемою в сучасному цифровому світі.

Ось чому більший файл із вищим бітрейтом майже завжди є найкращим вибором, коли потрібно використовувати формат із втратами.

Високим стандартом якості для стиснення MP3 є 320 кбіт/с[10]. За таких налаштувань дуже важко відрізнити стиснений звук від нестисненого в випадкових тестах прослуховування.

Для швидкісного і точного аналізу формати із стисненням і втратами непридатні. Наявність втрат неприйнятна, оскільки знижується точність одержуваних результатів, а необхідність розпакування даних збільшує час їх обробки. Отже, необхідно використовувати формат без компресії.

2.2 Дослідження AIFF формату

AIFF – це формат файлу аудіо-інтерфейсу, формат аудіо-файлів, розроблений компанією Apple у 1988 році. Він заснований на форматі IFF (Format Interchange File). Зазвичай люди більше знайомі з форматом MP3, але у порівнянні з AIFF, він має свої нюанси. AIFF – нестиснений аудіо-файл, який зберігає вищу якість порівняно з MP3. Але нестиснені аудіо-файли означають, що вони будуть займати більше місця на носії даних, до 10 МБ на хвилину аудіо-запису. Це може бути невелика ціна, якщо необхідний високоякісний аудіо вихід.

AIFF працює в основному на пристроях Mac і iOS, він також може відтворюватися на Windows за допомогою певних медіа-програвачів, таких як VLC. Windows додає розширення .aif до таких файлів перед їх читанням, тоді як Mac додає до них .aiff. Це означає, що користувачі Windows бачитимуть .aif, а користувачі Mac — розширення .aiff.

Однак цей формат аудіо-файлу все ще можна стиснути. Існує варіант, який може використовувати стиснення, щоб отримати файл, що займає менше місця на диску, і називається AIFF-C, або просто AIFFC, що означає формат файлу для обміну стиснутими звуками. Як і очікується, стиснені версії AIFF матимуть розширення .aifc.

Можливість відкривати та відтворювати медіа-файли AIFF є у більшості популярних програвачів, на Apple iTunes, VLC, Media Player Classic, Windows Media Player та більшості інших мультиформатних медіаплеєрів.

Файл Audio-IFF – це набір різних типів фрагментів. Існує обов’язковий Common Chunk, який містить важливі параметри, що описують форму сигналу, такі як його довжина та частота дискретизації. Фрагмент звукових даних, який містить фактичні дані форми сигналу, також необхідний, якщо дані форми сигналу мають довжину більше 0 (тобто у FORM насправді є дані форми сигналу). Усі інші частини є необов'язковими. Серед інших необов'язкових блоків є ті, які визначають маркери, перераховують параметри інструменту, зберігають інформацію, специфічну для програми, тощо.

Усі програми, які використовують FORM AIFF, повинні мати можливість читати 2 обов’язкові фрагменти та вибірково ігнорувати додаткові фрагменти. Програма, яка копіює FORM AIFF, повинна копіювати всі фрагменти в FORM AIFF, навіть ті, які вона вирішила не інтерпретувати. Також немає обмежень щодо порядку фрагментів у FORM AIFF. На рисунку 2.1 можна побачити приклад мінімального файлу AIFF. Він складається з однієї FORM AIFF, що містить 2 необхідні фрагменти, що повинні бути у кожному такому файлі, загальний фрагмент і блок даних звукових даних.

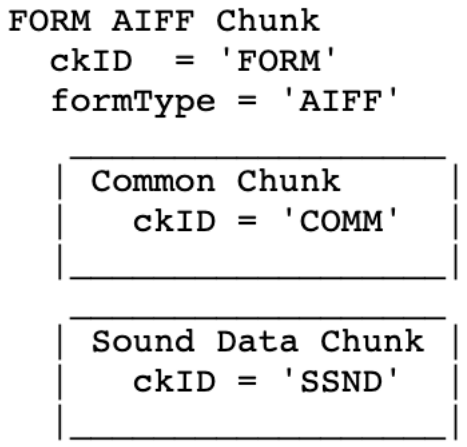

Рисунок 2.1 – Схема AIFF файлу

Значна частина інтерпретації файлів AIFF обертається навколо двох концепцій точок вибірки та кадрів вибірки.

Точка вибірки — це значення, що представляє зразок звуку в даний момент часу. Кожна точка вибірки зберігається як лінійне значення з доповненням до 2, яке може мати ширину від 1 до 32 біт (як визначено полем sampleSize в Common Chunk). Наприклад, кожна точка вибірки 8-бітового сигналу буде 8-бітовим байтом (тобто символом зі знаком).

Оскільки більшість операцій читання та запису мають справу з 8-бітовими байтами, було вирішено, що точка вибірки повинна бути округлена до розміру, кратного 8 при зберіганні в AIFF. Це полегшує читання AIFF у пам’яті. Якщо використовувати точку вибірки шириною від 1 до 8 біт, точку вибірки слід зберегти в AIFF як 8-бітовий байт (тобто, знаковий символ). Але якщо використовувати точку вибірки шириною від 9 до 16 біт, точку вибірки слід зберегти в AIFF як 16-розрядне слово (тобто коротко зі знаком).

Крім того, біти даних повинні бути вирівняні по лівому краю, а всі біти, що залишилися (тобто, заповнення), обнулені. Наприклад, розглянемо випадок 12-бітової точки вибірки. Він має 12 біт, тому точка вибірки повинна бути збережена як 16-бітове слово. Ці 12 бітів слід вирівняти по лівому краю, щоб вони стали бітами з 4 по 15 включно, а біти з 0 по 3 повинні бути встановлені на нуль. Нижче показано, як 12-розрядна точка вибірки зі значенням двійкового 101000010111 зберігається з вирівнюванням по лівому краю як два байти (тобто 16-бітове слово).

2.3 Дослідження FLAC формату

Файл FLAC — це аудіо-файл, стиснутий у форматі Free Lossless Audio Codec (FLAC), який є форматом стиснення звуку без втрат з відкритим вихідним кодом. Він схожий на файл .MP3, але стискається без втрати якості або втрати будь-яких оригінальних аудіоданих.

FLAC зменшує розмір цифрового аудіо приблизно на 60 відсотків і є широко підтримуваним форматом на найпопулярніших платформах. Цей формат з відкритим вихідним кодом і розроблений Xiph.Org Foundation.

Основна структура потоку FLAC:

– 4-байтовий рядок "fLaC";
– блок метаданих STREAMINFO;
– 0 або більше інших блоків метаданих;
– 1 або кілька звукових кадрів.

Перші чотири байти призначені для ідентифікації потоку FLAC. Наведені нижче метадані містять всю інформацію про потік, крім самих аудіоданих. Після метаданих надходять закодовані аудіо-дані.

FLAC визначає кілька типів блоків. Блоки метаданих можуть мати будь-яку довжину, і можна визначити нові. Декодеру дозволено пропускати будь-які типи метаданих, які він не розуміє.

Обов’язковим є лише один: блок STREAMINFO. Цей блок містить таку інформацію, як частота дискретизації, кількість каналів тощо, а також дані, які можуть допомогти декодеру керувати своїми буферами, як-от мінімальна та максимальна швидкість передачі даних, мінімальний та максимальний розмір блоку. Також до блоку STREAMINFO входить підпис MD5 незакодованих аудіоданих. Це корисно для перевірки всього потоку на наявність помилок передачі.

Інші блоки дозволяють додавати доповнення, таблиці пошуку, теги, таблиці ключів і дані, що стосуються програми. Є параметри flac для додавання блоків PADDING або визначення точок пошуку. FLAC жорстко не вимагає наявності точок пошуку для пошуку, але вони можуть прискорити пошук або використовуватися для підказки в програмах для редагування.

Крім того, є можливість визначити свій власний блок метаданих та записати його ідентифікатор. Тоді є можливість зарезервувати блок PADDING правильного розміру під час кодування та перезаписати блок заповнення потрібним блоком APPLICATION після кодування. Отриманий потік буде сумісним з FLAC; декодери, які знають вказані метадані, можуть використовувати їх, а інші безпечно ігнорують їх.

Після метаданих надходять закодовані аудіо-дані. Аудіо-дані та метадані не чергуються. Як і більшість аудіо-кодеків, FLAC розбиває незакодовані аудіо-дані

на блоки і кодує кожен блок окремо. Закодований блок упаковується у фрейм і додається до потоку. Еталонний кодер використовує один розмір блоку для всього потоку, але формат FLAC цього не вимагає.

Розмір блоку є важливим параметром для кодування. Якщо він занадто малий, верхня частина рамки зменшить стиснення. Якщо він занадто великий, етап моделювання компресора не зможе створити ефективну модель. Розуміння моделювання FLAC допоможе покращити стиснення деяких видів введення, змінюючи розмір блоку.

У найбільш загальному випадку, використовуючи лінійне передбачення для аудіо 44,1 кГц, оптимальний розмір блоку буде від 2 до 6 ксемплів. У цьому випадку flac за замовчуванням має розмір блоку 4096. Використовуючи швидкі фіксовані предиктори, менший розмір блоку зазвичай є кращим через менший заголовок кадру.

2.4 Дослідження WAVE формату

Аналіз показав, що вся необхідна інформація в зручній для обробки та запису формі містить формат WAVE (wav). Аудіо-файл складається з двох основних блоків. Перший блок – це інформаційний заголовок файлу, який містить таку інформацію:

- розмір файлу;
- кількість каналів;
- частота дискретизації;
- глибина звучання (швидкість).

Другий блок складається з даних, що зберігають цифровий сигнал – набір значень амплітуд. Звук складається з коливань, які при оцифруванні набувають ступінчастого вигляду. Це обумовлено тим, що комп'ютер може відтворювати в будь-який короткий проміжок часу звук певної амплітуди (гучності) і цей короткий момент не нескінченно короткий. Тривалість цього проміжку і визначає частота дискретизації. Сукупність амплітуди і короткого проміжку часу носить назву

семпл. Амплітуда виражається числом, яке може займати в файлі 8, 16, 24, 32 біт (або більше).

Таким чином, чим більше місця в пам’яті зарезервовано для числової амплітудної характеристики, тим більший діапазон значень можна записати. При розробці програмної реалізації слід враховувати важливий момент, що в одноканальному звуковому файлі значення амплітуди вже є. Наприклад, стереозвук спочатку має значення всіх амплітуд для лівого каналу, потім для правого.

Як показано в таблиці 2.1, аудіо-формат wave дозволяє записувати сигнал на кількох каналах і з різною частотою дискретизації. Людське вухо сприймає звуки з частотою дискретизації не більше 22 кГц, тому робоча частота повинна бути в тих же межах.

Крім того, не доцільно орієнтувати підсистему на багатоканальну обробку сигналів, оскільки більшість обчислювальних пристроїв, які ми маємо сьогодні, із вбудованими можливостями запису використовують одноканальні мікрофони із середньою якістю запису, а використання кількох каналів збільшить час обробки.

Таблиця 2.1 – Структура wave файлу

Номер байту	Поле	Опис
0..3 (4 байти)	chunkId	Містить символи “RIFF” в ASCII кодуванні
4..7 (4 байти)	chunkSize	Розмір файлу
8..11 (4 байти)	format	Містить символи “WAVE”
12..15 (4 байти)	subchunk1Id	Містить символи “fmt”
16..19 (4 байти)	subchunk1Size	16 для формату для імпульсно-кодової модуляції

Кінець таблиці 2.1

20..21 (2 байти)	audioFormat	Аудіо формат
22..23 (2 байти)	numChannels	Кількість каналів
24..27 (4 байти)	sampleRate	Частота дискретизації аудіо
28..31 (4 байти)	byteRate	Кількість байт, переданих за секунду відтворення
32..33 (2 байти)	blockAlign	Кількість байт для одного семплу, включаючи всі канали
34..35 (2 байти)	bitsPerSample	Глибина звучання аудіо-треку
36..39 (4 байти)	subchunk2Id	Містить символи “data”
40..43 (4 байти)	subchunk2Size	Кількість байт в області даних
44..	Data	Аудіо-дані у форматі WAV

2.4 Дослідження перетворення Фур'є

Дискретне перетворення Фур'є - це один з найважливіших інструментів цифрової обробки сигналів. Є три найбільш поширених варіанти використання ДПФ. Перший з них - це обчислення частотного спектра сигналу. Обчислення спектру безпосередньо дозволяє розкрити інформацію, закладену в частоті, фазі і амплітуді гармонійних компонентів, на які розкладається сигнал. Прикладом такого представлення інформації і системи, що її розпізнає, можуть служити мова і слух людини. Другий варіант використання ДПФ пов'язаний з розрахунком частотної характеристики системи по її імпульсній характеристиці або навпаки. Перехід до частотної характеристики дозволяє вести аналіз і опис систем в

частотній області, що є альтернативою опису в часовій області і застосування згортки. А третій варіант використовує ДПФ як проміжний етап більш складних перетворень сигналу. Класичним прикладом цього є реалізація згортки в частотній області - швидка згортка, на виконання якої витрачається в сотні разів менше часу, ніж на традиційну.

Дискретне перетворення Фур'є (ДПФ) є перетворенням Фур'є, яке може бути обчислено на цифровому комп'ютері. Формулу перетворення можна побачити на рисунку 2.2.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn}$$
$$= \sum_{n=0}^{N-1} x_n \cdot \left[\cos\left(\frac{2\pi}{N} kn \right) - i \cdot \sin\left(\frac{2\pi}{N} kn \right) \right],$$

Рисунок 2.2 – Формула ДПФ

Реалізація ДПФ вимагає багато комплексних множень і додавань. Кількість комплексних множень і додавань може бути значно зменшена за рахунок ефективного рекурсивного N-точкового ДПФ, коли N є степінь 2. Ці алгоритми ДПФ відомі як алгоритми швидкого перетворення Фур'є (FFT) (існує кілька ефективних кодів, які були майже еквівалентні FFT). Алгоритм ШПФ був представлений для обробки сигналів в 1965 році в статті авторів Кулі і Тьюкі (Cooley–Tukey). Однак сама ідея виникла більш ніж за сто років до цього у німецького математика К.Ф. Гауса в 1805 році. Але його роботи з даної теми виявилися забутими через відсутність цифрових комп'ютерів, в яких цей підхід можна було б застосувати на практиці.

Для графічного відображення спектру сигналу на основі FFT потрібний кадр (блок) даних. Якщо припустимо, що частота дискретизації Fs = 48 кГц (період вибірки Ts = 1 / 48000 = 20.83 мкс) і розмір кадру 2048 вибірок, то спектр може не бути оновлений не швидше, ніж кожних 2048 * 20.83 мкс = 42.66 мс, що

еквівалентно 23,44 оновлень дисплею в секунду. За телевізійним стандартом зображення оновлюються 25 або 30 кадрів в секунду, а фільми в кінотеатрах зазвичай оновлюються з частотою 24 кадри в секунду, так що 23,44 оновлень може бути задовільним. Зверніть увагу на те, що в цьому прикладі ми тепер маємо 42.66 мс або більше 42660000 тактових циклів процесора (для частоти 1ГГц) для обробки даних. Якщо ми подвоюємо розмір кадру, то отримаємо в два рази більше часу для обробки даних, однак частота оновлення спектру зменшиться вдвічі, що не може бути прийнятними для користувача.

Розмір кадру є одним з багатьох нюансів інженерного проектування, який має бути врахований при проектуванні системи. Якщо вихідні блоки посилаються на ЦАП, то вони будуть перетворені в аналоговий сигнал вибірка за вибіркою на заданій частоті вибірки Fs. Для правильної роботи наступний блок сигналу для виведення повинен бути готовий на той час, коли перетвориться остання вибірка поточного блоку. Наприклад, якщо ми виконуємо обробку звуку, це гарантує, що з точки зору слухача не буде ніякого «розриву» в музиці, і на виході вона не буде звучати інакше, ніж при використанні безблокової обробки. Звичайно, є часовий лаг або затримка, яка рівна періоду блоку, але вона непомітна для слухача.

Швидке перетворення Фур’є (ШПФ) – це алгоритм, який обчислює дискретне перетворення Фур’є (ДПФ) послідовності або її обернене (IDFT). Аналіз Фур’є перетворює сигнал з його вихідної області (часто часу або простору) у представлення в частотній області і навпаки. Приклад структури алгоритму можна побачити на рисунку 2.3.

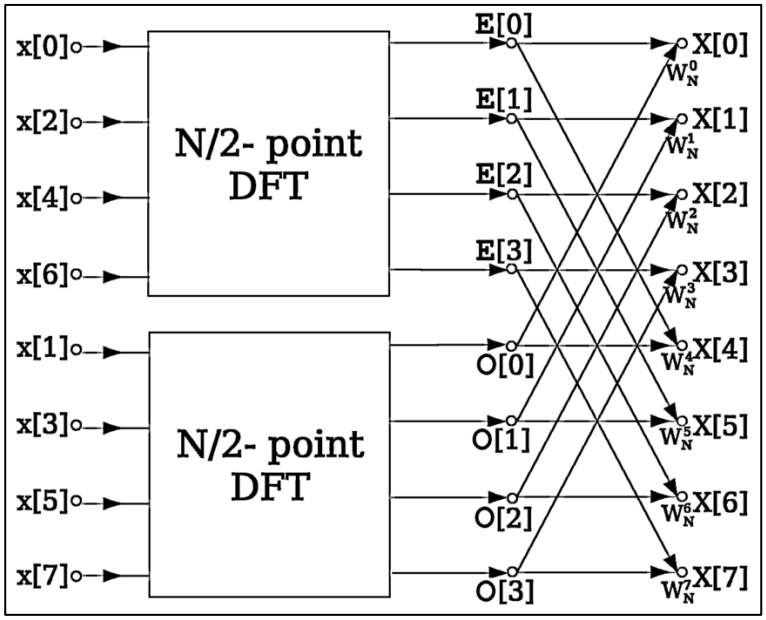

Рисунок 2.3 – Структура ШПФ алгоритму

DFT отримують шляхом розкладання послідовності значень на компоненти різних частот. Ця операція корисна в багатьох областях, але обчислення її безпосередньо з визначення часто занадто повільне, щоб бути практичним. ШПФ швидко обчислює такі перетворення, розкладаючи матрицю DFT на добуток розріджених (переважно нульових) факторів. В результаті йому вдається зменшити складність обчислення DFT, що виникає якщо просто застосувати визначення DFT. Різниця в швидкості може бути величезною, особливо для довгих наборів даних, де N може бути тисячами або мільйонами. За наявності помилки округлення багато алгоритмів ШПФ є набагато точнішими, ніж прямо чи опосередковано оцінювання визначення DFT. Існує багато різних алгоритмів ШПФ, заснованих на широкому спектрі опублікованих теорій, від простої арифметики комплексних чисел до теорії груп і теорії чисел.

2.5 Дослідження вейвлет-перетворення

Зазвичай перетворення Фур’є використовується для вилучення частот із сигналу. Перетворення Фур’є використовує серію синусоїд з різними частотами для аналізу сигналу. Але у нього є великий недолік. Це вибір правильного розміру вікна. Відповідно до принципу невизначеності Гейзенберга:

– вузьке вікно локалізує сигнал у часі, але буде значна невизначеність частоти;
– якщо вікно досить широке, то тимчасова невизначеність збільшується.

Це компроміс між часовою та частотною роздільною здатністю.

Одним із способів уникнути цієї проблеми є Multiresolution Analysis (MRA). Прикладом MRA є саме вейвлет-перетворення. У MRA сигнал аналізується на різних рівнях роздільної здатності що дозволяє досягти найбільш точних та репрезентативних результатів обчислень.

Вейвлет – це хвилеподібне коливання, локалізоване в часі. Вейвлети мають 2 основні властивості: масштаб і розташування. Масштаб визначає, наскільки «розтягнутий» або «здавлений» вейвлет. Розташування – це положення в часі. На рисунку 2.4 наведено формулу вейвлет-перетворення.

$$T(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t)\, \psi^* \frac{(t-b)}{a} dt$$

Рисунок 2.4 – Формула вейвлет-перетворення

Вейвлет-перетворення може змінити параметр «масштабування», щоб знайти різні частоти в сигналі разом із їх розташуванням. Менший масштаб означає, що вейвлет здавлюється. Таким чином, він може захоплювати більш високі частоти. З іншого боку, більший масштаб може захопити нижчі частоти. На нижченаведеному рисунку 2.5 можна побачити приклад стиснутого вейвлета.

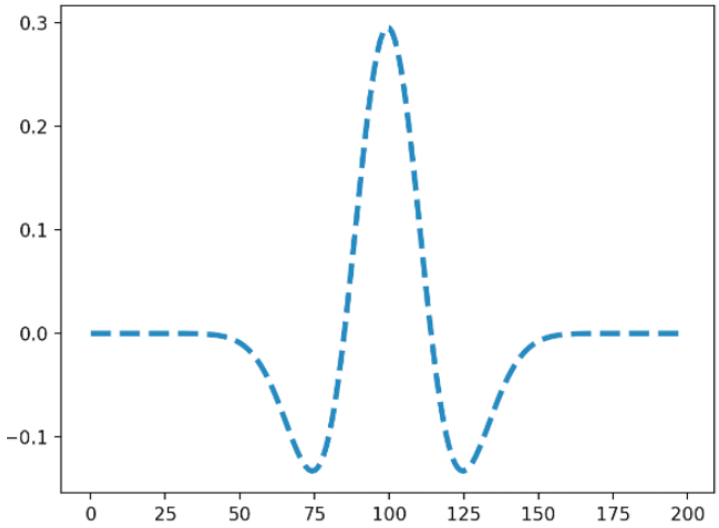

Рисунок 2.5 – Стиснутий вейвлет

В свою чергу, на рисунку 2.6 зображено приклад розтягнутого вейвлету.

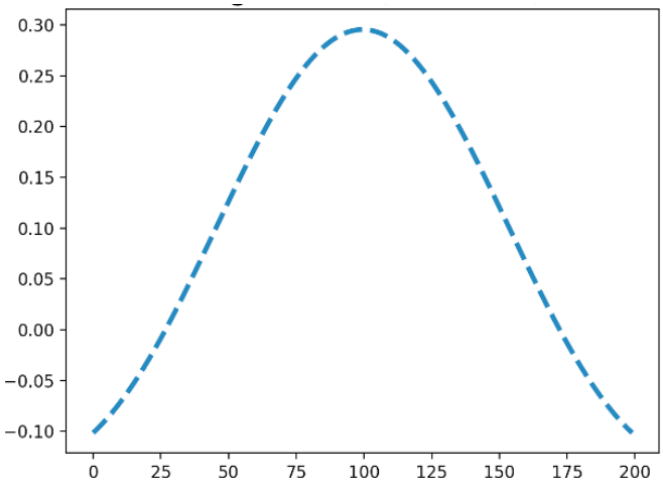

Рисунок 2.6 – Розтягнутий вейвлет

Це перевага вейвлет-перетворення перед перетворенням Фур’є. Він може фіксувати спектральну та тимчасову інформацію одночасно. Таким чином, в основному, сигнал згортається з набором вейвлетів у різних масштабах і положеннях. Початковий вейвлет, який масштабується і зміщується, називається «материнським вейвлетом». На сьогоднішній день існує доволі широкий вибір різних вейвлетів, які використовуються для різних випадків.

Для прикладу, поглянемо детальніше на один із вейвлетів. А саме на вейвлет Морле (або вейвлет Габора), нижче на рисунку 2.7 наведено його формулу. Він використовується як для звукових, так і для візуальних даних, оскільки він тісно пов’язаний із людським сприйняттям. Він складається з комплексної експоненти, помноженої на гаусове вікно.

$$\psi(t) = exp(-i\omega_0 t)exp(-t^2/2)$$

Рисунок 2.7 – Формула вейвлета Морле

На рисунку 2.8 наведено приклад того, як саме виглядає цей вейвлет.

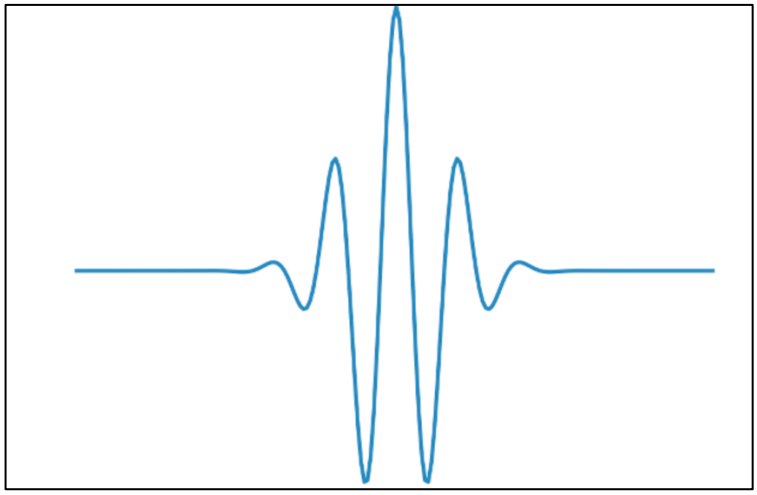

Рисунок 2.8 – Вигляд вейвлета Морле

3 ПРОЕКТУВАННЯ СИСТЕМИ

3.1 Проектування архітектури програмного забезпечення

Основою мобільного додатку є архітектурний патерн MVVM (model-view-viewmodel) та принципи Clean Architecture. MVVM відокремлює ваше представлення (тобто активність та фрагменти) від вашої бізнес-логіки. Цей патерн може використовуватись для проектів з різноманітними розмірами кодової бази. Але коли кількість коду стає величезною, ViewModels починають роздуватися, що може завдати шкоди як розробнику так і продукту в цілому.

MVVM в поєднанні з чистою архітектурою працюють стабільно та зрозуміло у різноманітних умовах. Це ще один крок у розділенні обов'язків кодової бази. Це поєднання чітко абстрагує логіку дій, які можна виконати у мобільному додатку.

Модель повинна представляти поточний стан нашої view, який може бути станом завантаження, успіхом або невдачею. І наше представлення має відображати інтерфейс користувача відповідно до поточного стану (рисунок 5.1).

Рисунок 3.1 – Схема взаємодії MVVM

MVVM в основному має 3 шари:

- Model;
- ViewModel;
- View.

Модель представляє дані та бізнес-логіку програми. Однією з рекомендованих стратегій реалізації цього шару є надання його даних за допомогою патерну observer, щоб повністю відокремити їх від ViewModel або будь-якого іншого спостерігача/споживача.

ViewModel взаємодіє з моделлю, а також готує спостережувані елементи, які можна спостерігати за допомогою View. ViewModel може додатково надавати шляхи для view, щоб передавати події в модель. Однією з важливих стратегій реалізації цього рівня є від'єднання його від View, тобто ViewModel не повинна знати про View, з яким взаємодіє.

Нарешті, роль View в цьому шаблоні полягає в тому, щоб спостерігати (або підписатися) на спостережувану ViewModel, щоб отримувати дані, і відповідно оновлювати елементи інтерфейсу користувача.

ViewModel як компонент нещодавно приєднался до стандартних представлених компонентів архітектури. Компоненти архітектури забезпечують новий клас під назвою ViewModel, який відповідає за підготовку даних для UI/View. Стандартна реалізація ViewModel дає хороший базовий клас для шару ViewModel, оскільки розширювані класи ViewModel (і його дочірні AndroidViewModel) автоматично зберігають свої дані під час змін конфігурації. Це означає, що після зміни конфігурації ці дані, що зберігаються в ViewModel, відразу ж доступні для наступної активності або фрагменту.

У будь-якого підходу є свої переваги та недоліки. Тож розглянемо переваги використання чистої архітектури у поєднанні з MVVM:

- код легше перевірити, ніж із звичайним MVVM;
- код додатково відокремлений (найбільша перевага);
- у структурі пакетів легше орієнтуватися;
- команда може додавати нові функції ще швидше;
- проект ще легше підтримувати.

Але у даного підходу існують свої недоліки, їх не багато, але все ж необхідно враховувати при розробці програмного продукту. До недоліки чистої архітектури можна віднести наступне:

- досить складний вхідний поріг. Щоб зрозуміти, як усі шари працюють разом, може знадобитися деякий час;

– додавання великої кількості додаткових класів, тому не ідеально підходить для проектів низької складності.

Схематичне зображення чистої архітектури та взаємодії її компонентів зображено на рисунку 3.2.

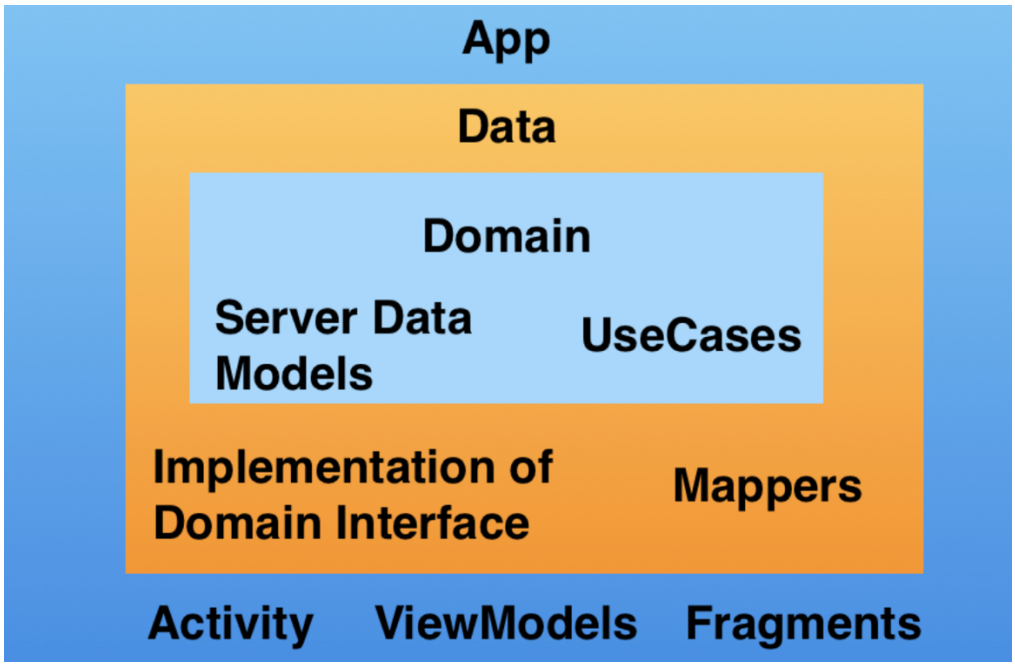

Рисунок 3.2 – Схема чистої архітектури

Важливо пам’ятати, що чиста архітектура не є засобом захисту, але її можна застосувати до будь-якої платформи. На основі проекту повинні визначити, чи відповідає вона вашим вимогам.

Якщо проект великий і складний, з великою кількістю бізнес-логіки, наприклад, чиста архітектура має багато переваг. Однак для менших і простіших завдань переваги можуть не варті того – ви будете писати більше коду і збільшувати складність з усіма рівнями, а також вкладатимете більше часу.

3.2 Вибір методів обчислення у фоні

Всі знають закон Мура про подвоєння швидкості обчислень кожні два роки, але зараз він підходить до кінця. Для значного збільшення часу обробки залишилося використовувати архітектуру, що складається з кількох обчислювальних блоків.

Kotlin розробляє корутини, які допомагають нам писати асинхронний код синхронно. Android – це однопотокова платформа. За замовчуванням все виконується в основному потоці (потоці інтерфейсу користувача), тому, коли настав час для виконання операцій, не пов'язаних з інтерфейсом користувача (наприклад, мережевий виклик, операція БД, операції введення-виводу файлів або будь-яке завдання), ми делегуємо ці завдання іншим потокам, а після завершення, якщо потрібно, передайте результат назад до потоку інтерфейсу користувача[11].

Android має власні механізми для виконання завдання в іншому потоці, наприклад AsyncTask схему роботи якого зображено на рисунку 3.3, Handler, Services тощо.

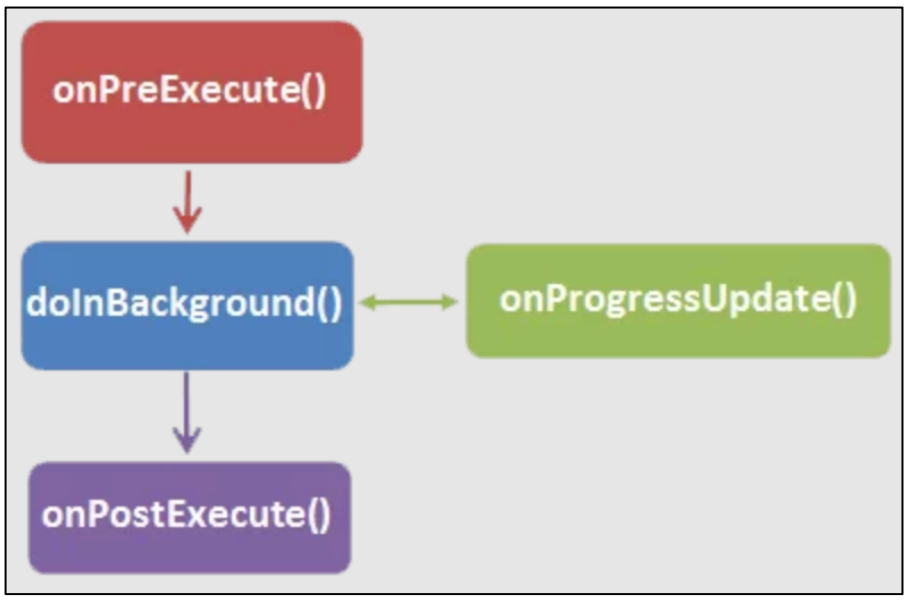

Рисунок 3.3 – Схема роботи AsyncTask

Усі вони гарні та працюють досить стабільно, якщо їх використовувати ретельно. Ці механізми включають зворотні виклики, методи публікації для передачі результату між потоками, але чи не було б краще, якби ми могли

позбутися всіх цих зворотних викликів і могли б писати асинхронний код так само, як ми пишемо код синхронізації.

Kotlin корутини дозволяють легко, послідовно писати паралельний код. Основна перевага корутини це те, що вони не прив'язані до одного потоку. Вони можуть почати виконання в одному, але потім відновитися в іншому – це слід розглядати як «легкий потік». Схематичне зображення роботи корутин можна подивитися на рисунку 3.4. Ось чому можливо легко запустити 100 тисяч корутин без сильного впливу на продуктивність або, ще гірше, отримати виключення OutOfMemoryException.

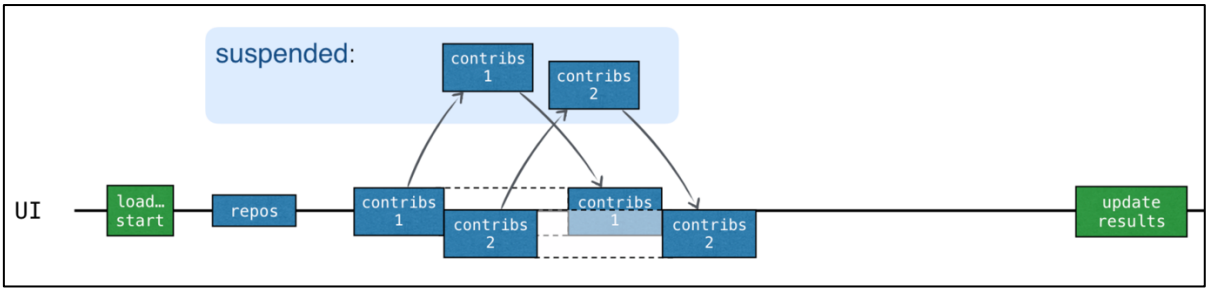

Рисунок 3.4 – Схематичне зображення роботи корутин

У Android корутини допомагають керувати довготривалими завданнями, які в іншому випадку можуть заблокувати основний потік і призвести до того, що додаток не відповідає. Понад 50% професійних розробників, які використовують корутини, повідомили про підвищення продуктивності.

Спираючись на вищезазначену інформацію, було вирішено у мобільному додатку використовувати архітектурний патерн MVVM у поєднанні з принципами чистої архітектури. Це дозволить швидко та безболісно розвивати та розширяти додаток у майбутньому. Як засобом важких обчислень у фоні необхідно використовувати корутини, адже вони являються найоптимальнішим рішенням цієї задачі.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Запис даних у файл

Android пропонує кілька варіантів збереження даних програми. Який саме варіант обрати, залежить від конкретних потреб, наприклад, скільки місця потрібно для ваших даних, які дані вам потрібно зберігати, а також від того, чи мають дані бути приватними для програми чи доступними для інших програм і користувачів. Android декларує наступні дозволи, які необхідні для роботи з пам’яттю:

- READ_EXTERNAL_STORAGE;
- WRITE_EXTERNAL_STORAGE;
- MANAGE_EXTERNAL_STORAGE.

У попередніх версіях Android програмам потрібно було оголосити дозвіл READ_EXTERNAL_STORAGE для доступу до будь-якого файлу за межами окремих каталогів програми на зовнішній пам’яті. Крім того, програмам потрібно було оголосити дозвіл WRITE_EXTERNAL_STORAGE для запису в будь-який файл за межами каталогу програми.

Останні версії Android більше покладаються на призначення файлу, ніж на його розташування для визначення можливості програми отримати доступ до певного файлу та записувати в нього. Зокрема, якщо програма націлена на Android 11 (рівень API 30) або вище, дозвіл WRITE_EXTERNAL_STORAGE не впливає на доступ програми до сховища. Ця цільова модель зберігання покращує конфіденційність користувачів, оскільки програмам надається доступ лише до тих областей файлової системи пристрою, які вони фактично використовують.

Android 11 вводить дозвіл MANAGE_EXTERNAL_STORAGE, який надає доступ на запис до файлів за межами каталогу програми та MediaStore.

Існує 4 типи сховища, які надає Android:

- внутрішнє сховище файлів, для збереження приватних файлів програми у файловій системі пристрою;

- зовнішнє сховище файлів, для збереження файлів у спільній зовнішній файловій системі. Зазвичай це стосується спільних файлів користувача, наприклад фотографій;
- preferences, для збереження приватних примітивних даних в парах ключ-значення;
- бази даних, для збереження структурованих даних в приватній базі даних;
- FileProvide, якщо ви необхідно поділитися файлами з іншими програмами.

У даному мобільному додатку ми будемо використовувати внутрішнє сховище для файлів. За замовчуванням файли, збережені у внутрішній пам’яті, є приватними для програми, і інші програми не можуть отримати до них доступ (як і користувач, якщо він не має root-доступу). Це робить внутрішню пам’ять хорошим місцем для внутрішніх даних програми, до яких користувачеві не потрібен прямий доступ. Система надає приватний каталог у файловій системі для кожної програми, де можна впорядковувати будь-які файли, які потрібні програмі.

Приблизну схему розподілення сховищ у системі Android зображено на рисунку 4.1.

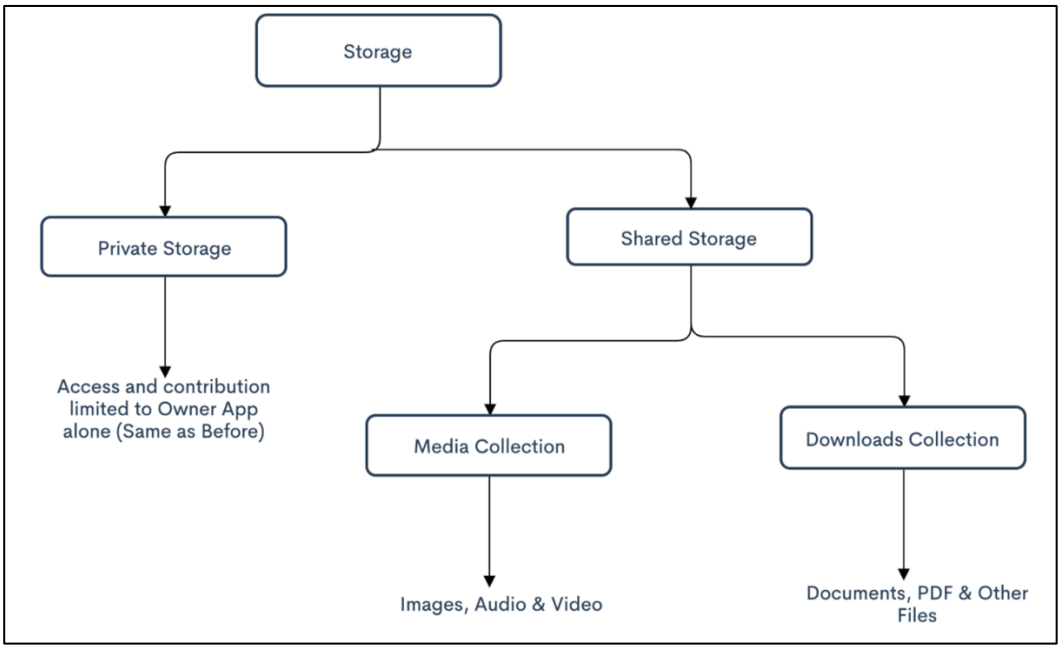

Рисунок 4.1 – Схема зберігання даних в Android

Коли користувач видаляє програму, файли, збережені у внутрішній пам’яті, видаляються. Через таку поведінку не потрібно використовувати внутрішню пам’ять для збереження того, що користувач очікує, щоб зберігатися незалежно від програми.

Зберігаючи файл у внутрішній пам’яті, є можливість отримати відповідний каталог як файл, викликавши один із двох методів:

- getFilesDir, повертає файл, що представляє внутрішній каталог;
- getCacheDir повертає файл, що представляє внутрішній каталог для файлів тимчасового кешу програми.

Щоб створити новий файл в одному з визначених каталогів, необхідно використовувати конструктор File(), передаючи файл або шлях до цієї директорії, наданий одним із вищевказаних методів, який визначає внутрішній каталог пам’яті для зберігання.

4.2 Запис звукових даних до WAVE файлу

Окрім використання наміру запустити диктофон та використання MediaRecorder, Android пропонує третій метод запису звуку, використовуючи клас AudioRecord. AudioRecord є найбільш гнучким із трьох методів, оскільки він дає нам доступ до вихідного аудіо-потоку, але має найменшу кількість вбудованих можливостей, наприклад, він не може стискати аудіо автоматично. Основи використання AudioRecord прості. Нам просто потрібно побудувати об’єкт типу AudioRecord, передаючи різні параметри конфігурації.

Клас AudioRecord керує аудіо-ресурсами для запису аудіо з апаратного аудіо входу платформи. Це досягається шляхом «витягування» (читання) даних з об’єкта AudioRecord. Розроблюваний додаток відповідає за вчасне опитування об'єкта AudioRecord за допомогою одного з наступних трьох методів:

– read(byte[], int, int);
– read(short[], int, int);
– read(java.nio.ByteBuffer, int).

Вибір того, який метод використовувати, буде залежати від формату зберігання аудіо-даних, який є найбільш зручним для користувача AudioRecord.

Після створення об’єкт AudioRecord ініціалізує пов’язаний з ним аудіо-буфер, який він заповнить новими даними. Розмір цього буфера, зазначений під час побудови, визначає, скільки часу може записувати AudioRecord до «перезавантаження» даних, які ще не були прочитані. Дані слід зчитувати з аудіоапаратури фрагментами розмірів, менших за загальний розмір буфера запису.

Виражена в Гц, частота дискретизації в екземплярі AudioFormat виражає кількість звукових семплів для кожного каналу за секунду у вмісті, який ви відтворюєте або записуєте. Це не частота вибірки, з якою відтворюється або створюється вміст. Наприклад, звук із частотою дискретизації медіа 8000 Гц можна відтворювати на пристрої, що працює з частотою дискретизації 48 000 Гц; перетворення частоти дискретизації автоматично обробляється платформою, вона не відтворюватиметься на швидкості 6x.

Для лінійного PCM аудіо-фрейм складається з набору вибірок, захоплених одночасно, чия кількість та асоціація каналів задаються маскою каналу, а вміст вибірки визначається кодуванням. Наприклад, стерео 16-розрядний кадр PCM складається з двох 16-розрядних лінійних PCM-семплів з розміром кадру 4 байти. Для стисненого звуку аудіо-фрейм може по черзі посилатися на одиницю доступу зі стислими байтами даних, які логічно згруповані разом для декодування та доступу до бітового потоку, або на один байт стиснених даних, або лінійний кадр PCM є результатом декодування стиснених даних, залежно від контексту, де використовується аудіо-фрейм.

Процес запису аудіо-даних до тимчасового файлу наведено на рисунку 4.2. Тимчасовий файл потрібен лише, як проміжний етап, адже для повноцінного функціювання аудіо-файлу потрібно записати метадані в початку файлу. Отже шукати шляхи уникнення тимчасового файлу не потрібно, адже після завершення обробки його буде видалено.

```
private suspend fun writeAudioDataToStorage() {
    val bufferSize = AudioRecord.getMinBufferSize(
        waveConfig.sampleRate,
        waveConfig.channels,
        waveConfig.audioEncoding
    )
    val data = ByteArray(bufferSize)
    val file = File(filePath)
    file.outputStream().use { outputStream →
        while (isRecording) {
            val operationStatus = audioRecorder.read(data,  offsetInBytes: 0, bufferSize)

            if (AudioRecord.ERROR_INVALID_OPERATION ≠ operationStatus) {
                if (!isPaused) {
                    outputStream.write(data)
                }

                withContext(Dispatchers.Main) {...}
            }
        }
    }
}
```

Рисунок 4.2 – Запис аудіоданих до файлу

Оскільки ми вже з’ясували структуру файлу (інформація про байти, поля, дані), тепер, щоб створити новий файл WAV, нам просто необхідно розмістити всю інформацію відповідно до заданого порядку та типів даних (String, int, short). Це заповнення даних ми можемо здійснити тільки після успішного здійснення запису аудіо-даних, оскільки нам потрібно до заголовків додати дані саме про звук. Наприклад необхідно ввести дані про тривалість записаного звуку, кількість каналів, частоту. В свою чергу і довжину самих метаданих необхідно також додати в заголовок, що не можливо здійснити до моменту запису. Тому всі ці дію здійснюються в останню чергу, при записі wav файлу, а вже згодом, з тимчасового файлу, який описаний вище, звукові дані додаються до вже створеного основного файлу, до якого вже записані всі метадані про записану звукову доріжку. Основні моменти запису та кодування даних про записаний аудіо трек можна побачити на рисунку 4.3.

```kotlin
private fun getWavFileHeaderByteArray(
    totalAudioLen: Long, totalDataLen: Long, longSampleRate: Long, channels: Int, byteRate: Long, bitsPerSample: Int
): ByteArray {
    val header = ByteArray( size: 44)
    header[0] = 'R'.code.toByte()
    header[1] = 'I'.code.toByte()
    header[2] = 'F'.code.toByte()
    header[3] = 'F'.code.toByte()
    header[4] = (totalDataLen and 0xff).toByte()
    header[5] = (totalDataLen shr 8 and 0xff).toByte()
    header[6] = (totalDataLen shr 16 and 0xff).toByte()
    header[7] = (totalDataLen shr 24 and 0xff).toByte()
    header[8] = 'W'.code.toByte()
    header[9] = 'A'.code.toByte()
    header[10] = 'V'.code.toByte()
    header[11] = 'E'.code.toByte()
    header[12] = 'f'.code.toByte()
    header[13] = 'm'.code.toByte()
    header[14] = 't'.code.toByte()
    header[15] = ' '.code.toByte()
    header[16] = 16
    header[17] = 0
    header[18] = 0
    header[19] = 0
    header[20] = 1
    header[21] = 0
    header[22] = channels.toByte()
    header[23] = 0
    header[24] = (longSampleRate and 0xff).toByte()
    header[25] = (longSampleRate shr 8 and 0xff).toByte()
    header[26] = (longSampleRate shr 16 and 0xff).toByte()
    header[27] = (longSampleRate shr 24 and 0xff).toByte()
    header[28] = (byteRate and 0xff).toByte()
    header[29] = (byteRate shr 8 and 0xff).toByte()
    header[30] = (byteRate shr 16 and 0xff).toByte()
    header[31] = (byteRate shr 24 and 0xff).toByte()
    header[32] = (channels * (bitsPerSample / 8)).toByte()
    header[33] = 0
    header[34] = bitsPerSample.toByte()
    header[35] = 0
    header[36] = 'd'.code.toByte()
    header[37] = 'a'.code.toByte()
    header[38] = 't'.code.toByte()
    header[39] = 'a'.code.toByte()
    header[40] = (totalAudioLen and 0xff).toByte()
    header[41] = (totalAudioLen shr 8 and 0xff).toByte()
    header[42] = (totalAudioLen shr 16 and 0xff).toByte()
    header[43] = (totalAudioLen shr 24 and 0xff).toByte()
    return header
}
```

Рисунок 4.3 – Запис метаданих до WAVE файлу

Після запису метаданих можна починати запис саме звукових даних до цього файлу з тимчасового. Адже для успішного подальшого використання звукового файлу потрібна повна підтримка усіх стандартів запису даних. Це допоможе правильно прочитати та програти файл на будь-якому пристрої, що підтримує даний тип аудіо-файлі.

4.3 Аналіз та відображення звукових даних в режимі реального часу

Для оптимального візуального аналізу звукових даних в режимі реального часу було обрано швидке перетворення Фур'є. Для виконання перетворення необхідний список float із заданим розміром вибірки. Розмір вибірки повинен бути коефіцієнтом 2, було обрано 4096. Збільшення цього числа призведе до більш точних даних, але збільшить час обчислення, а також більш рідкісні оновлення (оскільки ми можемо виконувати одне оновлення на кожні n байт звукових даних, де n – розмір вибірки). Якщо дані закодовані у стандартному форматі PCM-16, то одна амплітуда складається з 2 байт. Значення float насправді не мають значення, оскільки вони масштабуються. Результатом буде список плаваючих елементів. Приклад реалізації наведено на рисунку 4.4.

```
private fun processFFT(buffer: ByteBuffer) {
    srcBuffer.put(buffer.array())
    srcBufferPosition += buffer.array().size
    val bytesToProcess = SAMPLE_SIZE * 2
    var currentByte: Byte? = null
    while (srcBufferPosition > audioTrackBufferSize) {
        srcBuffer.position( newPosition: 0)
        srcBuffer.get(tempByteArray,  offset: 0, bytesToProcess)

        tempByteArray.forEachIndexed { index, byte ->
            if (currentByte == null) {
                currentByte = byte
            } else {
                src[index / 2] =
                    (currentByte!!.toFloat() * Byte.MAX_VALUE + byte) / (Byte.MAX_VALUE * Byte.MAX_VALUE)
                dst[index / 2] = 0f
                currentByte = null
            }

        }
        srcBuffer.position(bytesToProcess)
        srcBuffer.compact()
        srcBufferPosition -= bytesToProcess
        srcBuffer.position(srcBufferPosition)
        val fft = noise.fft(src, dst)
        listener(fft)
    }
}
```

Рисунок 4.4 – Реалізація перетворення Фур'є

В свою чергу необхідно коректно відобразити результати швидкого перетворення Фур'є. Для цього було розроблено декілька додаткових віджетів відображення. Перший віджет це контейнер який задає основні розміри нашого віджету, кешує дані які потрібно відобразити та відповідає за їх очистку. Приклад реалізації можна побачити на рисунку 4.5.

```
class Visualizer @JvmOverloads constructor(
    context: Context, attrs: AttributeSet? = null, defStyleAttr: Int = 0
) : FrameLayout(context, attrs, defStyleAttr) {

    private var currentWaveform: FloatArray? = null

    private val bandView = FFTBandView(context, attrs)

    init {
        addView(bandView, LayoutParams(LayoutParams.MATCH_PARENT, LayoutParams.MATCH_PARENT))
    }

    private fun updateProcessorListenerState(enable: Boolean) {
        if (!enable) {
            currentWaveform = null
        }
    }

    override fun onAttachedToWindow() {...}

    override fun onDetachedFromWindow() {...}

    fun onFFTReady(fft: FloatArray) {
        currentWaveform = fft
        bandView.onFFT(fft)
    }

    fun updateMode(isLight: Boolean) {
        bandView.maxConst = if (isLight) 5_000 else Int.MAX_VALUE / 40
    }

    fun clear() {
        currentWaveform = null
        bandView.onFFT(FloatArray( size: 4096))
    }
}
```

Рисунок 4.5 – Реалізація контейнеру

Другим віджетом який було розроблено є саме відображення результату функції швидкого перетворення Фур'є. Для цього необхідно відображати на Canvas вигляд функції після кожного оновлення. Оскільки оновлення надходять надто швидко, необхідно подумати про синхронізацію потоків для коректного

відображення всіх отриманих результатів. Основну частину реалізації даного віджету наведено на рисунку 4.6.

```
var accum = 0f
val nextLimitAtPosition = floor( x: FREQUENCY_BAND_LIMITS[currentFrequencyBandLimitIndex] / 20_000.toFloat() * size).toInt()

synchronized(fft) {
    for (j in 0 until (nextLimitAtPosition - currentFftPosition) step 2) {
        val raw = (fft[currentFftPosition + j].toDouble().pow( x: 2.0) + fft[currentFftPosition + j + 1].toDouble().pow( x: 2.0)).toFloat()
        val m = bands / 2
        val windowed = raw * (0.54f + 0.46f * cos( x: (currentFrequencyBandLimitIndex - m) * Math.PI / (m + 1))).toFloat()
        accum += windowed
    }
}
if (nextLimitAtPosition - currentFftPosition != 0) {
    accum /= (nextLimitAtPosition - currentFftPosition)
} else {
    accum = 0.0f
}
currentFftPosition = nextLimitAtPosition
var smoothedAccum = accum
for (i in 0 until smoothingFactor) {
    smoothedAccum += previousValues[i * bands + currentFrequencyBandLimitIndex]
    if (i != smoothingFactor - 1) {
        previousValues[i * bands + currentFrequencyBandLimitIndex] = previousValues[(i + 1) * bands + currentFrequencyBandLimitIndex]
    } else {
        previousValues[i * bands + currentFrequencyBandLimitIndex] = accum
    }
}
smoothedAccum /= (smoothingFactor + 1)
currentAverage += smoothedAccum / bands
val leftX = width * (currentFrequencyBandLimitIndex / bands.toFloat())
val rightX = leftX + width / bands.toFloat()
val barHeight = (height * (smoothedAccum / maxConst.toDouble()).coerceAtMost( maximumValue: 1.0).toFloat())
val top = height - barHeight
```

Рисунок 4.6 – Реалізація відображення графіку

Візуально поєднання цих двох успішно реалізованих віджетів можна побачити на рисунку 4.7, де зображено приклад швидкого перетворення Фур’є.

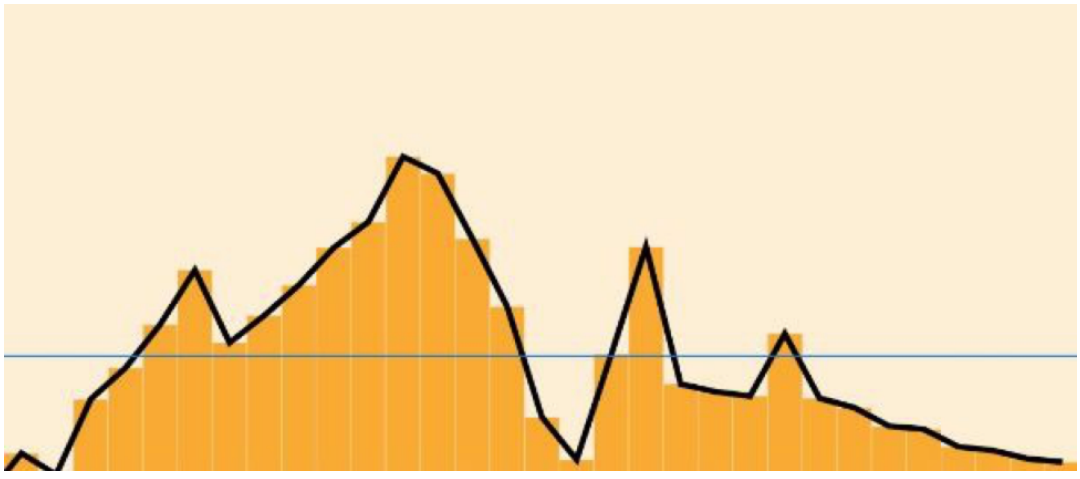

Рисунок 4.7 – Візуалізація результату

Цей графік змінюється в режимі реального часу відповідно до звукових коливань, що надходять до мікрофону смартфону та аналізуються додатком. Для більшої наглядності було проаналізовано середнє значення коливань на заданому проміжку, на що на графіку вказує синя лінія. Вона також змінює своє положення в режимі реального часу, як і весь графік.

4.4 Створення UI/UX дизайну

Спираючись на те, що застосунок створюється для проведення, одне з важливих рішень було це вибір шрифту який буде використаний. Застосунок не містить дуже багато інформації, яку критично необхідна користувачеві. В результаті цього, було обрано стандартний шрифт «Roboto», який використовується за замовчуванням. Він буде зрозумілим та зручним для читання. Приклад шрифту наведено на рисунку 4.8.

Рисунок 4.8 – Приклад шрифту

Головне вікно додатку умовно розділено на 3 основні частини, які необхідні для зручності користування. Ці зони можна назвати так:

– зона графіку проаналізованого звукового сигналу;
– зона керування записом;
– зона записаних аудіо-файлів.

Саме такий порядок зон було обрано для зручності користування та наглядності результату. Приклад розробленого інтерфейсу можна побачити на рисунку 4.9.

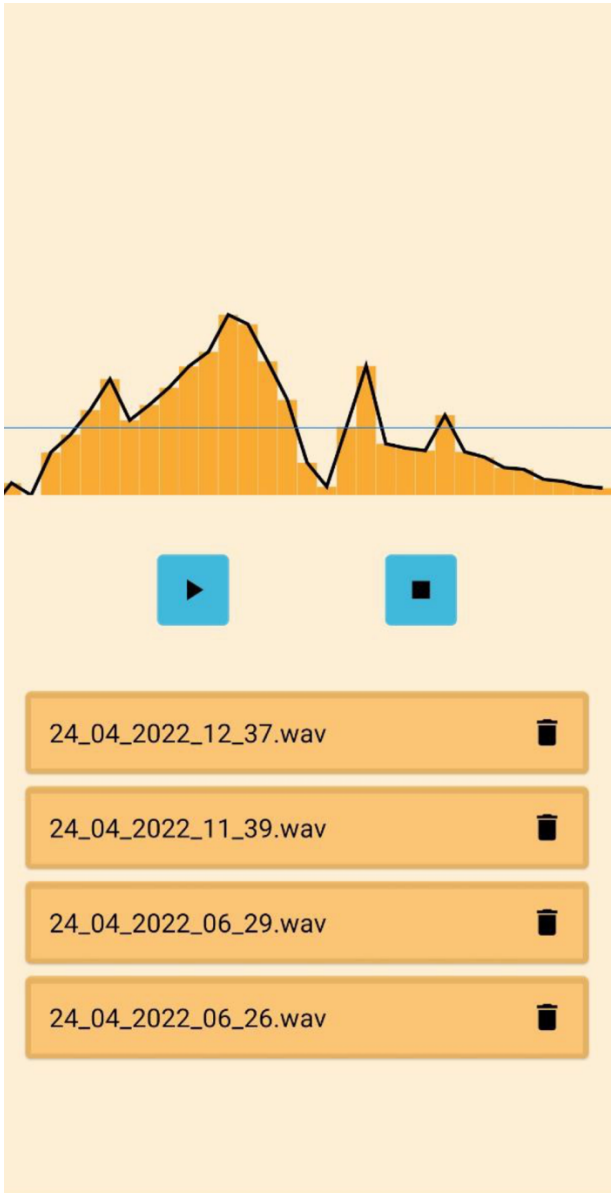

Рисунок 4.9 – Інтерфейс додатку

ВИСНОВКИ

В ході виконання магістерської роботи було виконано дослідження та аналіз методів запису та аналізу звуку з допомогою смартфону відповідно до вхідних даних.

В процесі дослідження методів запису та аналізу звуку на смартфоні було визначено оптимальні методи, які необхідно використовувати в залежності від типу задачі. Досліджено декілька типів файлів, які можуть використовуватися для запису аудіо-даних. Типи файлів, що використовуються для запису звуку зі втратами в даній роботі не розглядалися, адже для чистоти експерименту втрати не прийнятні. Було визначено, що оптимальним форматом для запису та аналізу звуку без втрат слід використовувати формат WAVE(wav).

Однією з найважливіших частин роботи було дослідження методів поверхневого аналізу звуку. В рамках цього було проведено детальне вивчення практичного застосування різних методів аналізу. У роботі було розглянуто дискретне перетворення Фур'є, що відіграє одну з фундаментальних ролей у сучасному світі аналізу звукових коливань. Разом з дискретним перетворенням велику увагу було приділено дослідженню швидкого перетворення Фур'є. Адже з його допомогою можливо здійснювати оптимальні обчислення в режимі реального асу на смартфоні. Також в рамках роботи було розглянуто вейвлет перетворення, як альтернативу перетворенню Фур'є. У них є свої переваги та недоліки. Але у даній роботі прикладом практичного застосування поверхневого аналізу звуку було взято швидке перетворення Фур'є. З його допомогою на головному вікні розробленого мобільного додатку є можливість спостерігати графічне відображення звукових коливань, що надходять до апаратної частини мобільного пристрою та трансформуються у зрозумілий для мобільного пристрою вигляд. На виході ми вже отримуємо значення байтів яке за допомогою швидкого перетворення Фур'є трансформуємо у більш зрозумілий для нас вигляд. Розроблений графічний інтерфейс інтерпретує результати перетворення, та

відображає їх. Додатково було вирахувано середнє значення коливань, що також можна спостерігати на віджеті з графіком.

У користувача є можливість, як записати аудіо на своєму мобільному пристрої так і програти його. Записане аудіо також обробляється за допомогою швидкого перетворення Фур'є, тому користувач має можливість спостерігати звукові коливання та їх середнє значення.

В результаті атестаційної роботи магістра виконано аналіз предметної галузі, поставлено задачу та розглянуто наявні методи запису та аналізу звуку на смартфоні, зроблено дослідження оптимальних форматів та алгоритмів роботи зі звуком. Під час роботи було підготовлено та опубліковано односторінкову публікацію на тему «Використання дискретного перетворення Фур'є на прикладі мобільного застосунку для запису та аналізу звуку»[12]. Розроблено мобільний додаток, що демонструє алгоритми запису та обробки звуку. Всі поставлені задачі в рамках кваліфікаційної роботи магістра були успішно виконані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Каук В.І., Гребенюк В.О., Пуголовок К.М., Водяницький Д.В. Виклики, які надають нові можливості//Екстренне дистанційне навчання в Україні: Монографія/ За ред. В.М.Кухаренка, В.В. Бондаренка. - Харків:. Вид-во КП "Міська друкарня", 2020.- 409 с.

2. Голямина И. П. Звук. Физическая энциклопедия. URL: www.femto.com.ua/articles/part_1/1222.html (дата звернення: 10.02.2022)

3. GetAClass. Физика в опытах и экспериментах. Источники звука. URL: https://youtu.be/nFcmTTT9yiE (дата звернення: 11.02.2022)

4. Шум. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Шум (дата звернення: 30.04.2022)

5. ДСТУ 2325-93. Шум. Терміни та визначення.

6. Білий шум. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Білий_шум (дата звернення: 30.04.2022)

7. What is Gray Noise? URL: https://www.techopedia.com/definition/27898/gray-noise (дата звернення: 23.04.2022)

8. Dynamic Time Warping. URL: https://handwiki.org/wiki/Dynamic_time_warping (дата звернення: 25.04.2022)

9. Suryo Wijoyo, "Speech Recognition Using Linear Predictive Coding and Artificial Neural Network for Controlling Movement of Mobile Robot", International Conference on Information and Electronics Engineering IPCSIT vol.6, IACSIT Press, pp.179-183, Singapore, 2011.

10. Neural network. Wikipedia. URL: https://en.wikipedia.org/w/index.php?title=Neural_network (дата звернення: 05.04.2022)

11. Viterbi algorithm. Wikipedia. URL: https://en.wikipedia.org/wiki/Viterbi_algorithm (дата звернення: 15.04.2022)

12. Черняхов А.В., Каук В.І. Використання дискретного перетворення Фур’є на прикладі мобільного застосунку для запису та аналізу звуку//Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. 2022. №2. С. 5

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

1. Каук В.І., Гребенюк В.О., Пуголовок К.М., Водяницький Д.В. Виклики, які надають нові можливості//Екстренне дистанційне навчання в Україні: Монографія/ За ред. В.М.Кухаренка, В.В. Бондаренка. - Харків:. Вид-во КП "Міська друкарня", 2020.- 409 с.

2. Viktor Kauk, Vyacheslav Grebenyuk, K. Pugolovok, D. Vodyanytskyi. Distance learning techniques and technologies for effective and successful online learning. URL: https://www.researchgate.net/publication/351794521_Distance_learning_techniques_and_technologies_for_effective_and_successful_online_learning (дата звернення: 10.05.2022)