НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2022 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

на тему: Музикальний сервіс для аналізу аудіо

Виконала : студентка __4__ курсу, групи ІП-84
(шифр групи)

_____Коломієць Євгенія Валеріївна_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____доцент, к.т.н. Волокита А. М._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) професор, д.т.н., Сімоненко В. П._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студенка _____
(підпис)

Київ – 2022 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)

“__” _____2022 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студентки

Коломієць Євгенії Валеріївни

1. Тема проєкту *Музикальний сервіс для аналізу аудіо*
керівник проєкту Волокита Артем Миколайович, доцент, к.т.н.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “ 10 ” __червня_____2022 р.
№ 1033-с

2. Термін здачі студентом закінченого проєкту *11 червня 2022 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Аналіз предметної області.

Розділ 2. Аналіз та вибір засобів розробки.

Розділ 3. Огляд реалізації системи.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна діаграма системи, діаграма модулів, послідовність дій програмного забезпечення.

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Сімоненко В. П.		

7. Дата видачі завдання «30» серпня 2021 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2021-20.12.2021*	
2.	*Вивчення та аналіз завдання*	*10.01.2022-15.02.2022*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2022-26.03.2022*	
4.	*Розробка структур окремих підсистем*	*01.04.2022-10.04.2022*	
5.	*Програмна реалізація системи*	*15.04.2022-18.05.2022*	
6.	*Оформлення пояснювальної записки*	*20.05.2022-10.06.2022*	
7.	*Захист програмного продукту*	*01.06.2022*	
8.	*Передзахист*	*11.06.2022*	
9.	*Захист*	*25.06.2022*	

Студент-дипломник _____ Євгенія КОЛОМІЄЦЬ
(підпис)

Керівник проєкту _____ Артем ВОЛОКИТА
(підпис)

АНОТАЦІЯ

Дана робота розглядає та використовує алгоритми аналізу аудіо даних що складають основу розроблюваної підсистеми сервісу аналізу музичних творів. У ході роботи обрано технології обчислень характеристик музики, що несуть найбільше практичне значення та проаналізовано їх ефективність. У результаті розроблено модулі для обробки аудіо даних, що здійснюють обчислення двух типів спектрального представлення звуку, та знаходження ритму, пульсу та тональності музичного твору. Обчислені результати зберігаються для можливості проведення наступного етапу аналізу – прогнозування акордів аудіо. Розроблений сервіс дозволяє здійснити попередній аналіз музичного твору та виявити його ключові характеристики. Програмне забезпечення реалізовано мовою Python.

Ключові слова: аудіо дані, аудіо аналіз, музичний аналіз, Python, аудіо обробка.

ANNOTATION

This project researches and makes use of audio data analysis algorythms that will make a core of a system that implements a music piece analysis service. As a part of research the music parametrs calculation technologies were considered and there practical value and efficiency were evaluated. The end result includes developing multiple modules for processing audio that implement calculating two different types of spectral form of the sound and finding the rhytm, beat and key of the musical piece. The results of the mentioned operations are stored for further analysis step – identifying the chords of a song. The service implemented in this project allows to perform the previous analysis of an audio and to find its key parametres. The software is implemented in Python programming language.

Key words: audio data, audio analysis, music analysis, Python, audio processing.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Музикальний сервіс	3		
			для аналізу аудіо			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Музикальний сервіс	77		
			для аналізу аудіо			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Музикальний сервіс	1		
			для аналізу аудіо			
			Структурна діаграма			
			системи			
	А4	***ІАЛЦ.4672008.005 Д2***	Музикальний сервіс	1		
			для аналізу аудіо			
			Діаграма модулів			
	А4	***ІАЛЦ.467200.006 Д3***	Музикальний сервіс	1		
			для аналізу аудіо			
			Послідовність дій			
			програмного			
			забезпечення			
	А4	***ІАЛЦ.467200.007 Д4***	Музикальний сервіс	29		
			для аналізу аудіо			
			Текст програмного коду			

Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	

ІАЛЦ.467200.001 ОА

Розроб		Коломієць Є.В.			*Музикальний сервіс для аналізу аудіо* Опис альбому
Перев		Волокита А.М.			

Літ.	Аркуш	Аркушів
	1	1

НТУУ "КПІ" ім. Ігоря Сікорського ***ФІОТ***
ІП-84

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Музикальний сервіс для аналізу аудіо*»

Київ – 2022

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку сервісу аналізу аудіо даних та системи для якої розроблені алгоритми в подальшому стануть функціональною частиною.

Областю застосування цієї системи є загальний аналіз аудіо, попередня обробка аудіо, проектування систем статистичного аналізу сучасних музичних творів, проектування системи розпізнавання комплексних музичних даних (системи розпізнавання акордів), оптимізація сервісів стримінгу аудіо та сервісів що зберігають та візуалізують аудіо.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка системи аналізу аудіо, алгоритми якої стануть основою комплексного аналізу аудіо.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проекту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Зручний та зрозумілий інтерфейс системи.
- Надати можливість обробки аудіо даних та визначення їх параметрів для подальшого використання.
- Надати можливість користувачам візуалізувати аудіо дані.
- Надати можливість користувача обирати вид перетворення аудіо даних.
- Надати вичерпну та зрозумілу документацію для розробленого продукту.

5.2. Вимоги до програмного забезпечення

- ОС Windows, Mac чи Linux.
- Мова програмування Python 3.
- Сервер Redis.

5.3. Вимоги до апаратної частини

- ЦП не менше ніж Intel® Core (TM) i3-2100T.
- ROM не менше ніж 32 ГБ для Windows, 64 ГБ для інших ОС
- RAM не менше ніж 3 ГБ.

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	10.12.2021-20.12.2021
Вивчення та аналіз завдання	21.12.2021-30.01.2022
Розробка архітектури та загальної структури системи	30.01.2022-25.04.2022
Розробка структур окремих частин системи	25.04.2022-1.04.2022
Програмна реалізація системи	1.04.2022-20.04.2022
Виправлення помилок	20.04.2022-25.05.2022
Оформлення пояснювальної записки	25.05.2022

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Музикальний сервіс для аналізу аудіо*»

Київ – 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

МТ	Музична теорія/ Теорія музики
BPM	(Beats per Minute) Удари на хвилину
MIR	(Music Information Retrieval) Дослідження музичної інформації
MFCC	(Mel Frequency Cepstral Coefficients) Мел спектрограма кепстральних коефіцієнтів
SR	(Sample Rate) Частота читання сигналу
STFT	(Short Time Fourier Transform) Короткочасна трансформація Фур’є
CLI	(Command Line Interface) Інтерфейс командного рядку
RAM	(Random Access Memory) Оперативна пам’ять
БД	База даних
КШ	(Алгоритм) Крюмхансля Шмукхлера

ВСТУП

Індустрія музики наразі є однією з найбільших медіа сфер і знаходить своє втілення у багатьох практичних компонентах: сервісах стримінгу музичних творів, програмних забезпеченнях для редагування та створення музики та у інтеграції аудіо з іншими видами медіа – зображеннями та відео.

Внаслідок великого обсягу попиту на програмне забезпечення пов’язане з музикою, для провідних сервісів постає потреба високої якості надання послуг зберігання, відтворення, створення та ідентифікації аудіо. Для ефективної реалізації цих завдань використовується аудіо аналіз.

Аналіз аудіо не є напрямком що використовується виключно для задач пов’язаних з музичними творами, а натомість знаходить застосування у засобах розпізнавання мови, поведінкового аналізу, аналізу аудіо фону для галузі безпеки, монітурування стану довкілля та навіть у галузі охорони здоров’я.

При аналізі музичних творів виокремлюють такі параметри що мають найбільше практичне значення:

- музичний жанр;
- тональність;
- ритм та темп;
- тональне представлення;
- гармонія та мелодія;
- ліричність, енергійність, настрій і т.д.

Дана робота досліджує галузь аналізу музичних творів з метою обчислення та зберігання їх окремих характеристик. Розроблена система стане частиною сервісу для ідентифікації акордів (характеристики що відповідає гармонії) музичних творів та втілює модулі попередньої обробки аудіо даних, такі як:

- візуалізація частотних та тональних представлень аудіо;

- визначення тональності твору;
- визначення пульсу (бітів) твору та його темпу;
- обчислення тональних представлень для кожного з ритмічних фреймів твору

з метою зберігання та надання результуючих даних для частини сервісу що на їх основі ідентифікує та проаналізує акорди що їх утворюють.

У даній роботі розглянуто різні сервіси аналізу даних та їх застосування у сфері стримінгу музики. Сервіс що розроблено має на меті використовувати алгоритми обчислення аудіо параметрів для приведення сирих аудіо даних до вигляду що є оптимальним для подальшої роботи над ними.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основи практичних аспектів музичної теорії

Для аналізу будь-якого формату даних важливим та необхідним кроком є встановлення основних компонентів що утворюють даний формат та грають роль у його класифікації. Музикальні елементи (єдиний такт, пісня, п’єса, концертний твір тощо) мають у своїй основі структурні та мелодійні особливості, що разом становлять теорію музики.

Метою МТ є аналіз та встановлення таких аспектів як: слухове сприйняття музики, естетичне сприйняття та візуальне представлення звуку. Візуальне сприйняття має найбільшу практичну цінність так як знаходить своє застосування у письмовому а в подальшому і у цифровому форматі зберігання аудіо. [1]

На початку важливо визначити природу та характеристики одиниці що утворює музику – звуку. Вважається що всі об’єкти світу це енергія що коливається на певній частоті. Коли щось коливається, воно утворює хвилю. З точки зору фізики хвилі це коливання які передають енергію, і загалом ми можемо спостерігати два типи хвиль – механічні та електромагнітні. Електромагнітні хвилі не потребують фізичного середовища для розповсюдження (води, повітря тощо) в той час як механічні завжди його потребують.

Таким чином звук – це механічні хвилі що розповсюджуються у фізичному середовищі. Отже, звук характеризується частотою, швидкістю, амплітудою та довжиною. Для аналізу та МТ найважливішою величиною є частота.

Частота є кількістю хвиль виміряних за якийсь проміжок часу. Одиницею вимірювання частоти є Герц (Hz), при чому 1 Герц означає що хвиля

повторюється раз на одну секунду. Чим вища частота (більше Герц), тим більше хвиль повторюється за секунду і навпаки. [2]

Фізичне представлення частоти знаходить своє застосування в такому понятті як висота звуку. Висота звуку це величина яка характеризує частоту на якій лунає звук. Для МТ висота визначає фактичне мелодійне значення ноти, тобто звук може бути вищим або нижчим за інші. Вивчення гармонії в музиці таким чином полягає у вивченні зв'язків та закономірностей між звуками різної висоти. Людський слух в середньому має змогу сприймати та відрізняти звуки частоти від 20 Гц до 20 000 Гц. З віком цей діапазон може зменшуватись.

Наступним фундаментальним поняттям МТ є нота. Нота це звук певної висоти з певною довжиною та тоном. Ноти мають назву що відповідають їх висоті – наприклад, нота До (інша назва – C) першої октави відповідає частоті 261. 63 Гц (табл. 1.1). Є декілька конвенцій щодо назв нот, які більш детально розглянуті у наступному підпункті.

Таблиця 1.1 – Залежність між назвою та частотою ноти [8]

Назва ноти	Назва ноти	Частота
E	Мі першої октави	329.63 Гц
D	Ре першої октави	293.66 Гц
C#	До-дієз першої октави	277.18 Гц
C	До першої октави	261.63 Гц
B	Сі малої октави	246.96 Гц
C	До малої октави	130.82 Гц
B	Сі великої октави	123.48 Гц
E	Мі четвертої октави	2637.00 Гц
D#	Ре-дієз четвертої октави	2489.00 Гц

Декілька звуків можуть мати різну довжину та можуть звучати по різному (на різних інструментах, голосі) і при цьому відповідати одній ноті. Якщо два звуки однієї ноти не звучать однаково, це означає що вони мають різний тембр. Тембр це тон, або забарвлення звуку. Нота До на струнному інструменті відрізняється від ноти До людського голосу через тембр.

Останнім базовим поняттям МТ є тон. [3] Він може, як і попередня характеристика, тембр, бути ознакою забарвлення або відчуття звуку (наприклад, тон може бути приємним та неприємним, схожим у різних інструментів тощо). Також тон може бути синонімом до ноти – назви звуку за його висотою та бути назвою для базового інтервалу (детальніше у підпункті).

Музичним тоном також називають одиничний сталий звук з певною частотою. На практиці звук однієї чистої частоти можна зустріти лише у ізольованому середовищі без сторонніх вібрацій, згенерований цифровими засобами. Звуки які створюють музичні інструменти, аудіо засоби тощо мають велику суміш частот через резонанс та неідеальне акустичне середовище. Ноти які ми чуємо (і внаслідок того, музика) є насправді цілим спектром різних частот, навіть якщо ми в результаті сприймаємо тільки основну частоту.

1.1.1 Поняття та представлення нот

Усі сучасні письмові музичні дані мають у своїй основі ноти. Вони використовуються для нотатного запису та читання творів, композитори використовують ноти для створення та зберігання музики. Ноти це фундаментальне поняття МТ.

У західній музиці існує 12 нот, що історично з’явились у європейській музиці та наразі є основною та найбільш уживаною системою нотної грамоти. Існують також інші системи нотування (Арабська, Китайська, Індійська тощо), але вони використовуються переважно для традиційної музики відповідних регіонів та не є загально уживаними.

Західна нотатна система (інша назва – американська або наукова система позначення) є найбільш поширеною та використовує як назви нот букви латинського алфавіту (табл. 1.2). Існує також латинська система та локалізовані назви для різних країн. Системи що мають в основі латинські назви також називають сольмізаційними, а системи що мають в основі букви алфавіту, називають алфавітними. [3]

Таблиця 1.2 – Відповідність назв нот деяких нотатних систем

Система назв нот								
Латинська	Do	Re	Mi	Fa	Sol	La	Si	Do
Західна	C	D	E	F	G	A	B	C
Українська	До	Ре	Мі	Фа	Соль	Ля	Сі	До

В подальшому у даній роботі будуть використовуватись назви західної системи. Вони мають назви: A, A#/Bb, B, C, C#/Db, D, D#/Eb, E, F, F#/Gb, G, G#/ Ab.

Для кращого аналітичного розуміння ці ноти проілюстровано на стандартній клавіатурі піаніно (рис. 1.1). Є декілька правил для розуміння принципу розташування нот.

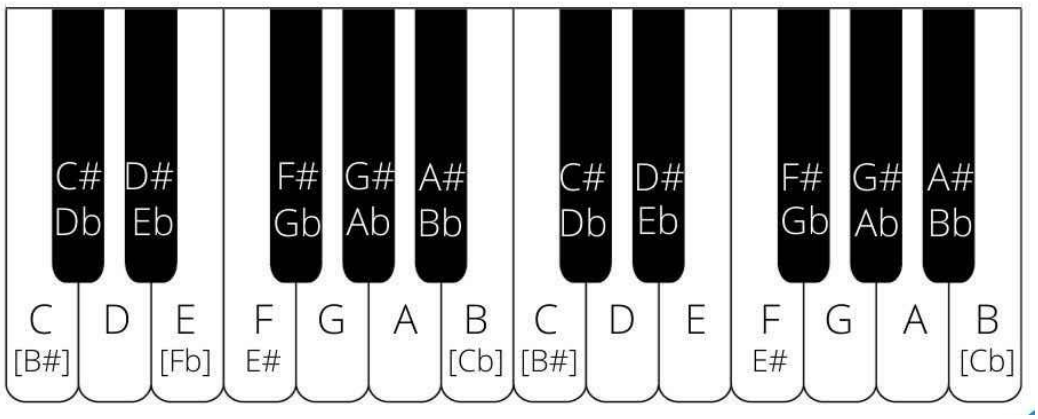


Рисунок 1.1 – Відповідність нот західної нотації клавішам на піаніно [16]

По-перше, 7 нот згідно з західною нотацією мають назви літер від A до G. Ноти між ними мають назви A#, C#, D#, F#, G# де # («дієз») означає що тон

ноти підвищений або Bb, Db, Eb, Gb, Ab де b («бемоль») означає що тон ноти понижений.

Ці два позначення означають одну ноту і називаються енгармонічно рівними. Тобто C# та Db, A# та Bb мають однаковий тон.

Між нотами B та C, E та F немає ні підвищень, ні понижень. Це є одним з прийнятих фундаментальних принципів організації сучасної музики.

Ноти які не підвищені та не понижені («білі клавіші»), називаються натуральними нотами. «Чорні клавіші» є завжди нотами з позначкою дієз або бемоль.

Відстань між будь-якими з цих 12 нот, що знаходяться по сусідству, називається напівтоном, і кожен півтон (наприклад між F та F#, Gb та G) репрезентує однакову відстань. Відстань що охоплює два напівтони – наприклад між C та D, E та F# називається тоном. [3]

1.1.2 Нотне коло

Важливим поняттям для розуміння організації сучасної музики є нотне коло. У нотному (рис. 1.2) колі розміщені всі 12 нот західної музики.

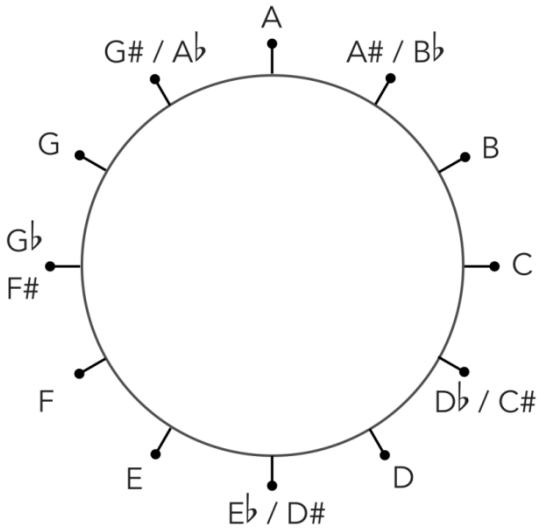


Рисунок 1.2 – Нотне коло [3]

Якщо рухатись за часовою стрілкою у колі, ноти стають вищими. У такому випадку потрібно використовувати позначку «дієз» - наприклад замість Ab потрібно використати G# . І навпаки, при русі вниз варто використовувати позначку «бемоль».

1.1.3 Компоненти музики

Після визначення основних характеристик звуку та його представлення в музикальній теорії є корисним вивчення поняття музики та її складових що її утворюють задля виділення головних компонентів корисних для її аналізу та класифікації.

На даний момент, оминаючи філософські та етнічні аспекти, є декілька тверджень що прийнято вважати правильними щодо визначення музики: музика складається зі звуку та музику утворюють і звук і його відсутність (тиша).

Маючи за основу ці твердження, можна виділити практичні характеристики музики, а саме: музика це твір людини, що складається зі звуку і тиші, має за основу тон (що поділяється на мелодію та акомпанемент або гармонію), ритм, динаміку, та текстуру і тембр. [3]

І гармонія і мелодія описують відношення між нотами (хоча різними способами) без урахування їх тривалості, а ритм описує відношення між нотами та їх тривалістю без урахування висоти їх звучання.

Гармонія утворюється коли ми сполучаємо декілька нот разом. Якщо додати ноту до іншої та відтворити їх разом або по черзі, вважається що утворена гармонія до базової ноти.

Гармонія вважається вертикальним відношенням між нотами. Це структура, або навіть мережа. Якщо визначити гармонічне відношення між двома або більше нотами, то можна трактувати їх як одночасне звучання, навіть якщо вони звучать окремо одна за одною.

Мелодія, як і гармонія, описує відношення між нотами, але у горизонтальному напрямку. Хоча вона все ще зберігає структурні відношення, вона трактує ноти враховуючи порядок, так що деяка кількість однакових нот, але відтворена у різному порядку, буде мати різне мелодійне значення, навіть якщо разом ці ноти мають однакове гармонійне значення. [4]

Мелодію можна вважати частиною гармонії зосередженої на тому як ноти звучать одна за одною. Зазвичай у сучасній музиці гармонія додається до існуючої мелодійної лінії для того щоб розширити її та зробити звучання більш повним та насиченим. Або навпаки, мелодія додається до існуючих акордів (гармонії).

Третій елемент, ритм, описує відношення між нотами чи звуками та часом, незалежно від їх тонального значення. Ритм описує яким чином звук «пульсує» або залишається сталим, його швидкість та повторюваність. Ритмічні структури описують рух твору залежно від визначених сталих проміжків часу і є одними з найважливіших характеристик музики. [3]

1.1.4 Мелодія

Як було зазначено, мелодія (рис. 1.4) є однією з головних компонентів музичного твору і обов’язково набором декількох нот (зазвичай більше ніж 3-х).

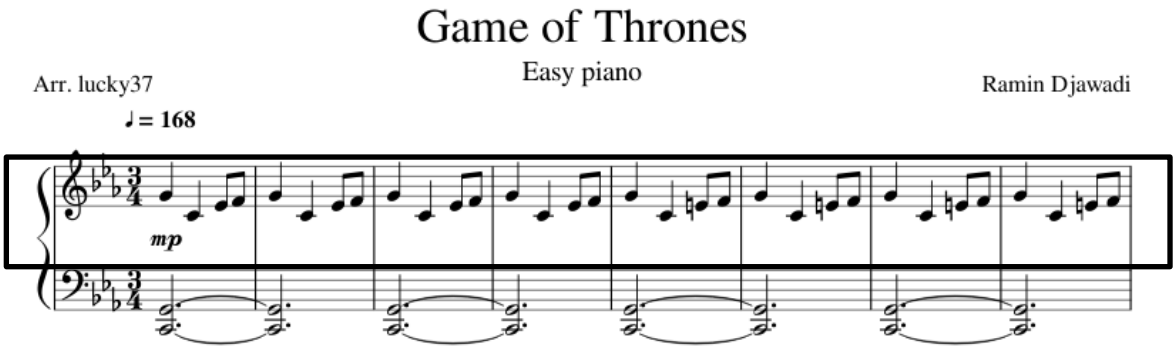


Рисунок 1.4 – Приклад популярної мелодії [20]

Більшість сучасних мелодій мають більше ніж 3 ноти – зазвичай вони складаються з 20 та більше нот у певній послідовності.

Більшість сучасних композицій в свою чергу складаються з декількох мелодій що складаються в одну хоча відворюються різними інструментами.

Мелодійна лінія може звучати одна без гармонії і мати назву монофонічної. В такому випадку вона може відтворюватись без змін одним або одночасно декількома інструментами. У сучасній музиці такий різновид використовується щоб емоційно виділити частину твору (рис. 1.5).

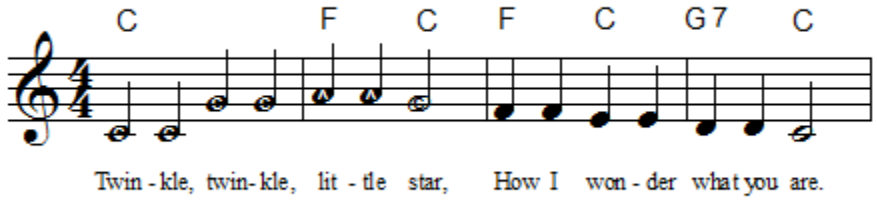


Рисунок 1.5 – Види мелодії – монофонічна [4]

Також її можуть супроводжувати одна або декілька гармонійна (зазвичай в такому випадку вона буде знаходитись тонально найвище) мелодія але зберігаючи той самий ритм. Така мелодія має назву гомофонічної – приклад даного виду мелодій наведено на рис. 1.6.

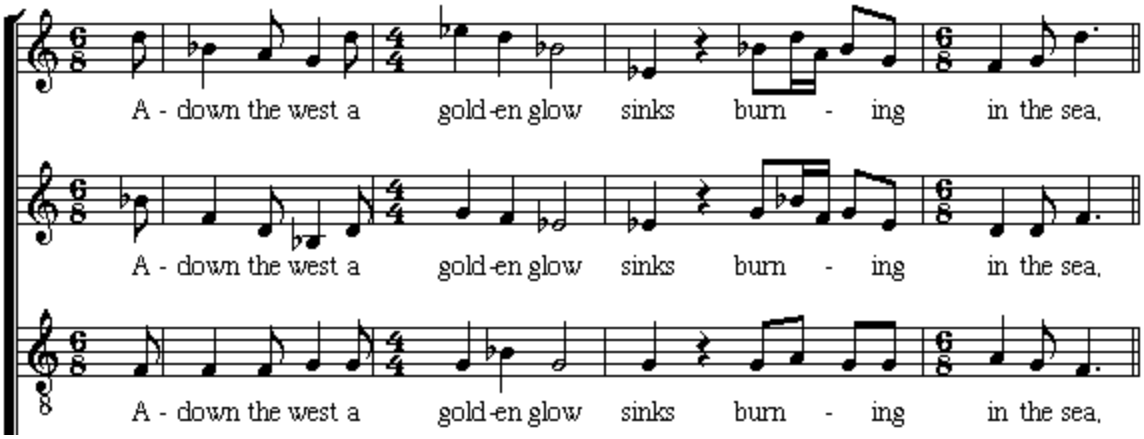


Рисунок 1.6 – Види мелодії – гомофонічна [4]

Гомофонічна мелодія зустрічається у класичнх творах та рідко знаходить застосування у сучасних творах.

Як було вказано вище, мелодія може співзвучати разом з іншими та утворювати поліфонію (рис. 1.7). Для цього зазвичай зберігаються такі принципи: компліментарна мелодія має звучати у окремій октаві та у моментах коли основна мелодія не має ритмічної інтенсивності, інші мають бути ритмічно посиленими. [4]

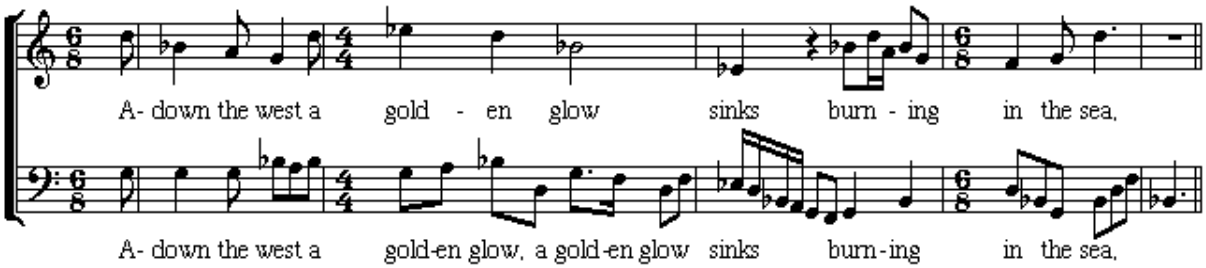


Рисунок 1.7. – Види мелодії – поліфонічна [4]

На практиці у сучасній західній музиці майже завжди використовується гомо фонічна мелодія.

1.1.5 Акорди та гармонія

У сучасні музиці акорди переважно утворюються за допомогою акордів. Акорди відтворені у відповідний мелодії час утворюють акомпанемент.

Акорд це музикальна одиниця що складається з декількох нот що звучать одночасно. Акорди походять з тональності, а саме з її певних нот. [3]

Акорд як і тональність визначається набором інтервалів відносно базової ноти. Таким чином акорди отримують свою назву – назва акорду визначає які ноти в нього входять, а саме: базову ноту (це може бути будь яка нота з 12 відомих), і з яких інтервалів він складається. Інтервальну структуру акорду - спосіб у який ноти розміщуються відносно базової, можна назвати сигнатурою акорду, отже якщо відома назва акорду, завдяки сигнатурі, можна визначити які ноти входять в нього. [5]

Гармонія створюється коли дві або більше нот звучать одночасно, хоча гармонія може бути виявлена навіть коли звуки звучать по черзі а не одночасно (наприклад, у арпеджіо). Дві ноти що звучать одночасно,

утворюють діаду. Три або більше нот що звучать одночасно, утворюють акорд, хоча зазвичай цей термін використовують для позначення певного визначеного набору звуків, а не будь-яких звуків.

Консонанс (співзвучність) можна визначити як декілька гармоній тони яких доповнюють резонанс один одного, дисонанс в свою чергу є назвою для гармоній що суперечать одна одній та ускладнюють сприйняття звуків.

Гармонія у західній сучасній музиці заснована на тріадах. Тріади це прості акорди що складаються з трьох нот між якими існують інтервали довжиною три.

В свою чергу, тріади поділяються на такі що мають базову ноту у основі та інвертовані. У першому тип тріад (рис.1.8) базова нота акорду (за якою акорд отримав назву), знаходиться найнижче. Друга нота знаходиться на інтервал довжиною 3 вище, а третя нота знаходиться на інтервал довжиною 5 вище (що є вище за другу ноту на інтервал довжиною 3).

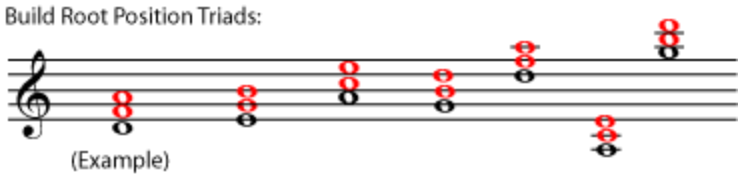


Рисунок 1.8 – Приклад неінвертованого акорду [5]

Другий тип акорду складається з тих самих нот що і перший тип, але ці ноти розташовані у іншому порядку. Вони також мають таку ж назву. Не є важливим, як далеко розташовуються друга або третя нота, важливим є яка нота є найнижчою (рис. 1.9). Інверсією першого типу називають акорд де найнижчою нота є друга нота базового акорду, інверсією другого типу – акорд де найнижче розташована третя нота базового акорду. [5]

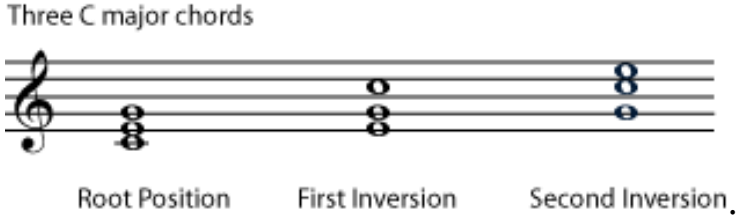


Рисунок 1.9 – Приклад інвертованого акорду [5]

На практиці для визначення акорду не важливий порядок нот у ньому-важливе тональне представлення.

1.1.6 Ритм

Щоб характеризувати та аналізувати ритм музики, потрібно ознайомитися з величинами що утворюють його: час, біт, такт та темп.

Час у музикальній теорії описує пульсацію музики і її ритмічний розмір (детальніше у натупному розділі). Біт у музиці є фундаментальною одиницею часу. На практиці біт відчувається аналітично – люди танцюють саме відповідно до розпізнаного біту музичного твору. [3]

Такт є одиницею вимірювання що визначає за який час біти музики утворюють повний цикл. Такт визначається розміром твору.

Темп описує швидкість, з якою відтворюються біти, або пульс музики. Темп зазвичай вимірюється в BPM, або бітах (ударах) за хвилину. Чим більше BPM, тим як правило є енергійнішою пісня (табл. 1.3).

Таблиця 1.3 – Порівняння темпу пісень різних категорій [9]

Назва твору	BPM	Тип твору
Teenage Dream	120	Танцювальна
Empire State Of Mind	174	Танцювальна (Хіп-хоп)
See You Again	100	Лірична
If We Ever	173	Стимулююча (Драм-енд-бейс)
Love Like This	101	Лірична
Lights	120	Енергійна

Іноді ліричні пісні мають BPM вищий за 100 та навіть 120, але зазвичай танцювальні та стимулюючі твори мають BPM від 150 до 200. Іноді BPM для зручності вимірють в half-time або double-time.

1.2 Огляд існуючих систем аналізу аудіо

На сьогодні існують безліч систем та інструментів для аналізу та збереження інформації про музичні твори та аудіо загалом. Вони використовують головні характеристики творів – ритм, елементи гармонії для класифікації та систематизації музичних творів. Розглянемо популярні та менш популярні системи та визначимо основні елементи що роблять їх корисними разом з елементами що потребують покращення або імплементації.

1.2.1 Spotify

Платформа Spotify є однією з найбільших платформ для стримінгу аудіо. Користувачі мають онлайн доступ до бази даних пісень, що налічує понад 30 млн екземплярів. [6]

Інтерфейс платформи зображено на рис. 1.10. і демонструє яку інформацію показано користувачу стосовно окремих творів та опції відтворення аудіо – за показником популярності або рекомендовані.

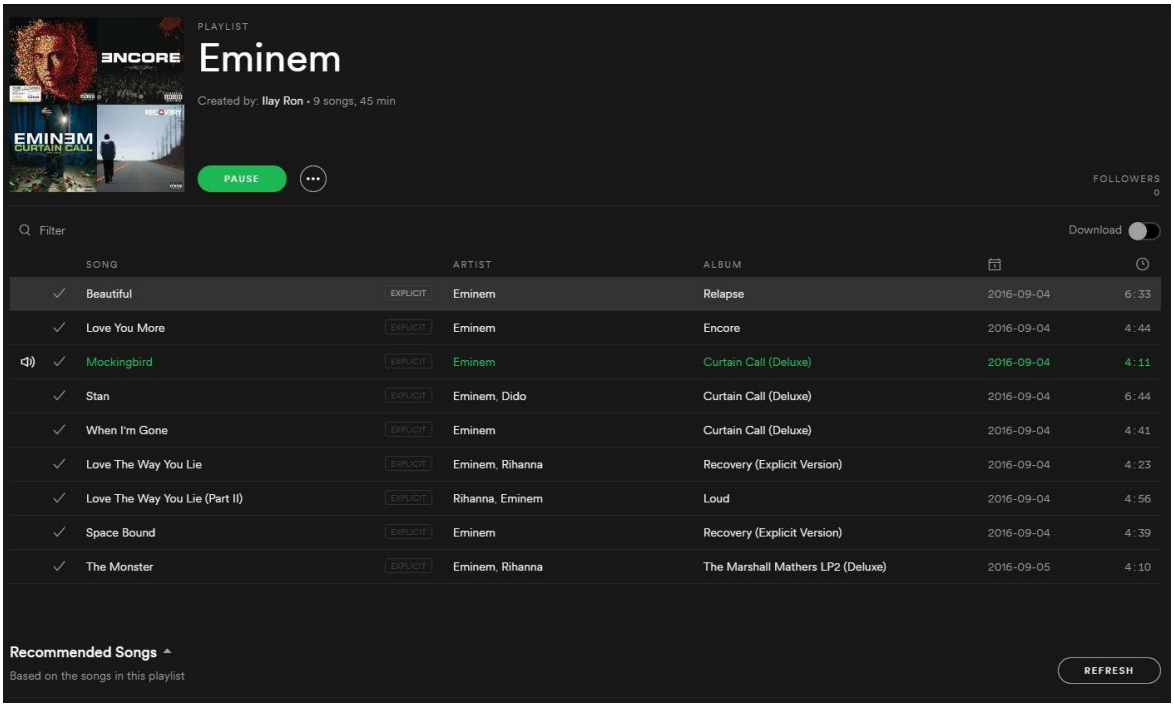


Рисунок 1.10. – Користувацький інтерфейс платформи Spotify

Пошук в даній платформі працює за допомогою збереженої інформації про артистів, їх твори, альбоми, музичні лейбли та плейлісти у яких вони знаходяться – як користувацькі так і офіційно складені самою платформою.

Розглянемо більш детально які алгоритми та дані роблять можливими ці функції.

Spotify Web API надає бібліотеку інструментів для застосування та аналізу аудіо, а саме для читання заздалегідь обчисленої інформації про аудіо різних категорій (табл.. 1.4).

Таблиця 1.4 – Мета інформація про аудіо що надає Spotify API

Категорія	Дані
Настрій	Танцювальність, Валенція, Енергійність, Темп
Властивості	Гучність, Інструментальність, Словесність
Контекст	Енергійність, Акустичність
Інше	Сегменти, Такти, Біти, Тони, Тембр

Є декілька ендпоінтів що є корисними для систематизації та аналітичної роботи з даними та обширний набір команд для отримання необроблених даних які зберігає платформа – творів, збережених творів певного користувача, рекомендацій певного користувача, даних щодо пошуку, жанрів тощо (рис. 1.11).

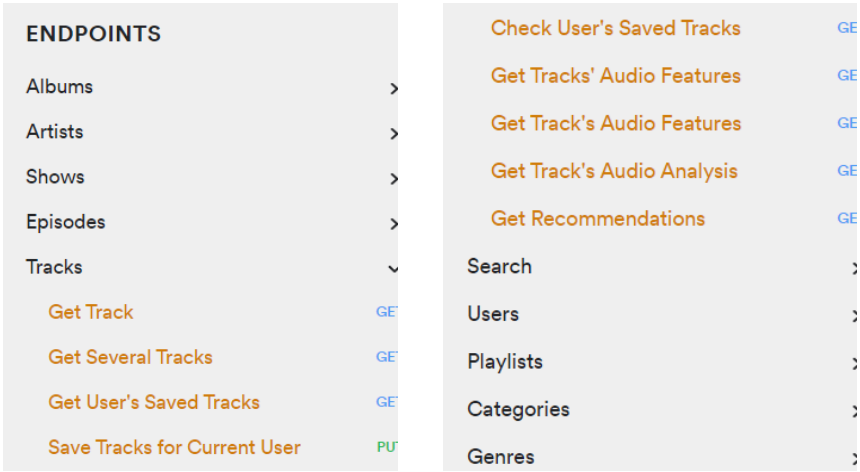


Рисунок 1.11 - Ендпоінти Spotify API

Засобами що надають корисну інформацію про аудіо є методи Get Tracks' Audio Features та Get Track's Audio Analysis.

Get Tracks' Audio Features приймає єдиним парметром ідентифікатор або масив ідентифікаторів творів у базі даних Spotify та у відповіді повертає такі дані як акустичність, енергійність, темп – дані зазначені вище. Також він повертає URL який можна використати як параметр наступного методу.

Get Track's Audio Analysis приймає зазначений вище URL та повертає низько рівневі (метадані) треку, такі як кількість семплів (детальніше у наступному розділі), тривалість, темп, гучність, музичний розмір, тональність, біти – всі дані необхідні для глибокого аналізу. [6]

1.2.2 Tunebat

Ще одним сервісом для аналізу та зберігання інформації про аудіо є веб сервіс Tunebat. Він зберігає таку інформацію як аудіо файл, його тональність, альтернативну тональність (що може бути корисним для роботи діджеїв - міксингу), темп та тривалість.

Окрім цього, більша частина інформації що надається Spotify API є присутньою, хоча іноді є нульовою що на практиці є помилковим твердженням (рис. 1.12).

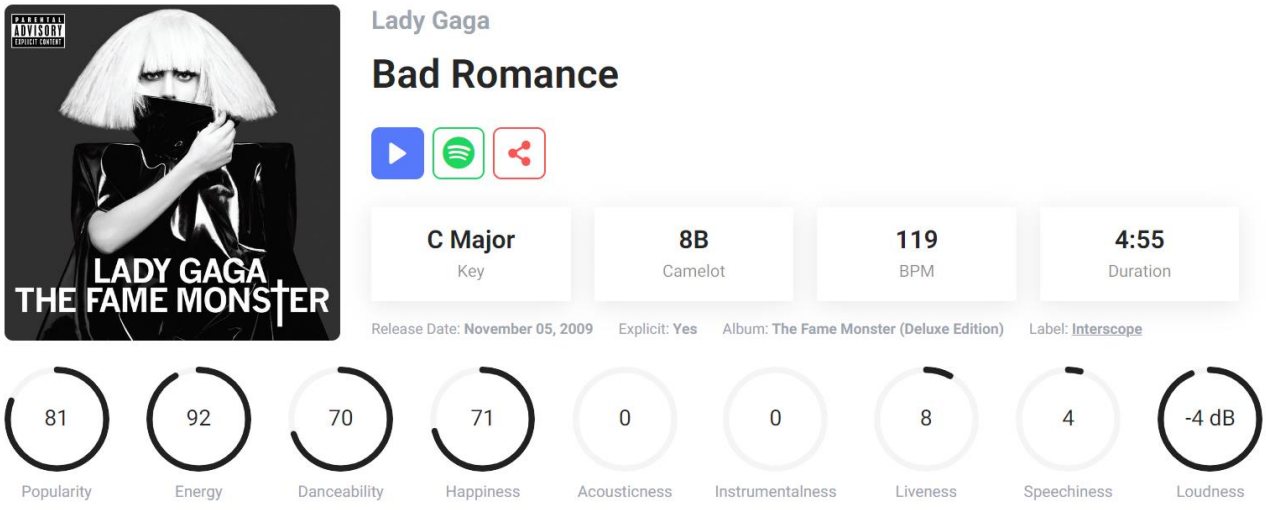


Рисунок 1.12 – Користувацький інтерфейс Tunebat

Щодо відтворення, даний сервіс використовує Spotify API, зберігаючи посилання на твір у за стосунку Spotify. Якщо користувач хоче відтворити трек, його юуде перенаправлено на іншу систему.

Також є декілька функцій притаманних саме цьому сервісу, такі як засіб для ізоляції вокалу (Vocal Remover) та обчислення тональності та темпу твору. Сервіс підтримує завантаження власних файлів у різних форматах та повертає характеристики зазначені вище.

1.2.3 Moises

Мойзес (Moises) є популярним веб сервісом що надає змогу розділяти різні компоненти аудіо композицій різних форматів за допомогою машинного навчання (рис. 1.13). Система використовує сучасний алгоритм поділу джерел, розробленого Deezer. Мойзес підтримує завантаження користувацького аудіо файлу з пристрою та завантаження треку за допомогою посиланням на YouTube. [10]

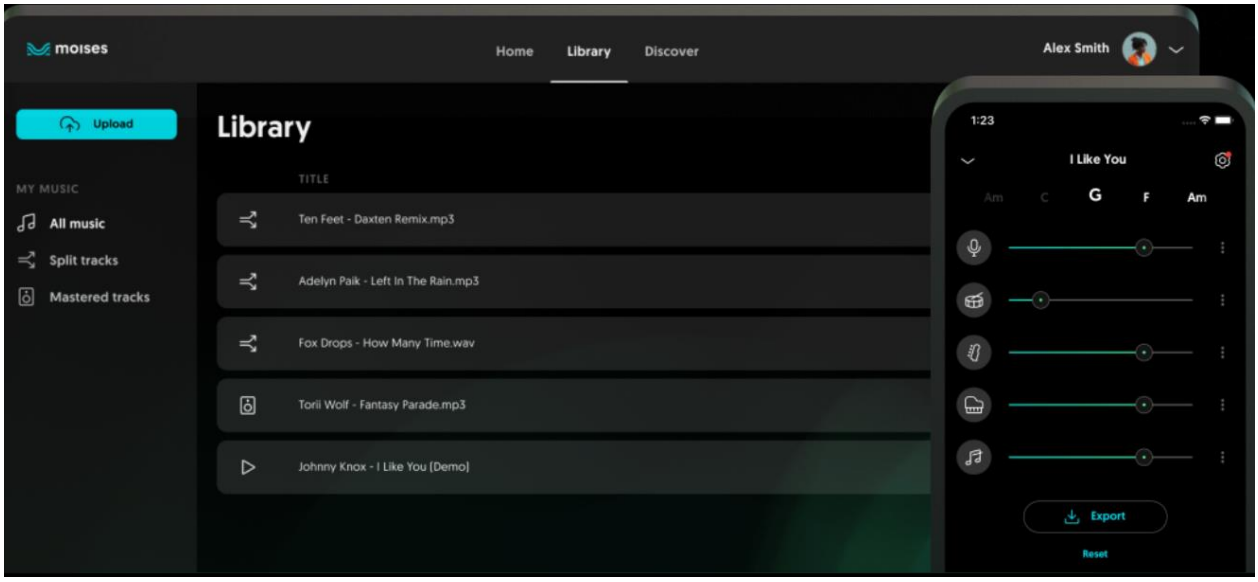


Рисунок 1.13 – Користувацький інтерфейс Moises

Головний функціонал сервісу це AI Audio Separation – розділення аудіо на такі компоненти як вокал, бас, гітара та перкусія та регулювання гучності їх відтворення у реальному часі за допомогою штучного інтелекту.

Також сервіс дозволяє визначити тональність та темп аудіо, і на даний момент доступна бета функція розпізнавання акордів (гармонії) у композиції.

1.2.4 Порівняння існуючих сервісів

Таким чином досліджено та виокремлено основні характеристики та функціонал популярних сервісів аналізу та зберігання інформації про аудіо композиції. Більш наочне порівняння складено у табл. 1.5.

Таблиця 1.5. – Порівняння існуючих сервісів аналізу аудіо

Функціонал	Spotify API	Tunebat	Moises
Отримання інформації: тональність	+	+	+
Отримання інформації: біти/ BPM	+	+	-
Завантаження користувацького аудіо	-	+	+
Розпізнавання акордів (гармонії)	-	-	+
Відокремлення частин аудіо композицій	-	+	+
Доступне API	+	-	-

Можна зробити висновок що кожна система містить функціонал що є корисним але жодна з них не задовольняє всіх вимог системи для попередньої обробки аудіо.

ВИСНОВОК ДО РОЗДІЛУ 1

У першому розділі було досліджено теоретичні засади аналізу музичних творів та аудіо загалом. Для цього було визначено природу звуку та методи його аналітичного сприйняття та нотування.

Важливим етапом дослідження було виявлення основних елементів музичних творів – мелодії, гармонії та ритму. Для даної роботи можна зробити висновок, що найважливішим є встановлення компонентів ритму – темпу та бітів та встановлення фундаментальної одиниці гармонії – тональності для подальшого аналізу та виявлення акордів.

Також було проаналізовано існуючі системи аналізу аудіо та порівняно компоненти що пов’язані саме з низькорівневими даними про музичні твори. Один з аналогів має повний набір методів для отримання даних про твори, але є обмеженим – можна отримати лише попередньо обчислену інформацію про аудіо що присутнє в базі даних сервісу.

Другий аналог підтримує завантаження користувацьких даних але не повертає дані у потрібному форматі – лише результат обчислень та є критично неточним щодо окремих величин.

Було виявлено, що розглянуті сервіси аудіо аналізу є основою популярних сервісів стримінгу та отримання інформації про музичні твори та внаслідок є важливим напрямком розробки.

Таким чином можна зробити висновок що розроблювана система є актуальною та має практичну цінність для галузі аналізу аудіо, так як має втілити основні компоненти попередньої обробки музичних даних.

РОЗДІЛ 2
АНАЛІЗ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

2.1. Вибір технології розробки алгоритму

Для розробки алгоритму аналізу аудіо потрібно окреслити основні вимоги до системи. Базовими вимогами є попередня обробка аудіо, в результаті якої буде визначено деякі характеристики твору (тональність, темп та біти) та перетворення вхідних аудіо даних на нормалізований масив оброблених проаналізованих даних.

Розглянемо декілька мов програмування що можуть бути придатними для роботи з аудіо.

2.1.1 Python

Python є популярною високорівневою мовою загального призначення. Ця мова використовується для розробки API, Штучного Інтелекту, веб розробки тощо.

Однією з причин популярності даної мови є її часте використання у науках про дані та аналізу даних. Існує 3 етапи роботи з даними на яких Python вважається високоефективним інструментом.

Збирання даних (Data Mining) є початковим етапом роботи з даними. Інженери даних використовують такі бібліотеки, як наприклад Scrapy та BeautifulSoup, для отримання великої кількості даних на. За допомогою Scrapy можна створювати програми та алгоритми, що збирають структуровані дані з Інтернету. Ця бібліотека також широко використовується для збору даних з API. BeautifulSoup, в свою чергу, використовується коли дані не можна отримати з API: він отримує та впорядковує дані їх у бажаному форматі. [7]

Найважливішим етапом є обробка та моделювання даних. Для цього використовуються такі бібліотеки як NumPy та Pandas. Вони створені для роботи з великим масивами даних та підтримують великий набір математичних операцій, що є ключовою частиною обробки аудіо. NumPy надає змогу виконувати велику кількість операцій за відносно короткий час, що є критично важливим, адже аудіо є великим масивом значень.

Також важливо відзначити що Python має бібліотеки для візуалізації даних (наприклад, Matplotlib) з багатьма опціями щодо відображення даних. Це є важливим для побудови графічних представлень аудіо (спектрограм, діаграм частот), для аналітичної оцінки роботи алгоритмів обробки аудіо.

2.1.2 R

R є мовою програмування що використовується для статистичного аналізу або обчислень. Ця мова містить набір інструментів для лінійного та нелінійного моделювання, статистичних тестів тощо.

R є популярною технологією для роботи з даними через те що є спеціалізованою мовою для різних етапів роботи з даними: попередньої обробки даних, візуалізації даних, тренування та оцінки роботи моделей штучного інтелекту та машинного навчання. [7]

Візуалізація даних в даній мові є фундаментальним елементом аналізу даних, і не потребує додаткових бібліотек. R надає можливості для різних способів демонстрації даних і має бібліотеку ggplot2 для побудови комплексних графіків.

Проте варто зазначити що етап збирання даних та етап обчислень є менш розвиненим – щодо збирання даних R підтримує невелику кількість форматів вхідних даних та має лише базові інструменти (такі як бібліотека Rvest) для здобуття інформації з веб ресурсів. [7]

Також варто зазначити що дана мова оптимізована для роботи з великими дата сетами, і є менш ефективною для роботи з масивами даних ніж бібліотеки зазначені для Python.

Таким чином можна підсумувати окремі характеристики досліджених мов (табл. 2.1) та зробити висновок що доцільно обрати мову програмування Python, так як вона має кращі показники у важливих категоріях.

Таблиця 2.1 – Порівняння застосувань мов Python та R на різних етапах роботи з даними

Категорія використання	Python	R
Збирання даних	+	
Математичні операції над великими ми масивами даних	+	
Візуалізація даних		+
Методи для комплексних статистичних обчислень		+

2.1.3 Потенційні недоліки використання мови Python

Хоча вибрана мова задовольняє основні потреби проєктуємої системи, важливо оцінити аспекти у яких вона має недоліки щоб впевнитись у доцільності її використання – визначити чи ці недоліки є критичними.

Відомо, що Python є повільнішим під час виконання порівняно з такими мовами як Java, C++, PHP, Javascript, Swift тощо. На перший погляд це є важливим критерієм для розробки.

Python є високорівневою мовою програмування. Виконання коду відбувається не за допомогою компілятору, а інтерпретатором. Інтерпритатор виконує рядки коду по черзі, що сповільнює систему.

Python також є динамічно типізованою мовою програмування. Це означає що більшість компонентів програми, що у випадку роботи зі статичною типізацією виконуються на етапі компіляції, виконуються під час виконання.

У динамічно типізованих мовах не потрібно вказувати тип змінної при присвоєнні значень – тип присвоюється під час виконання. Це означає що кожного разу при зверненні до змінної (читанні, посиланні чи зміні), її тип перевіряється та відповідно цьому призначається (алокується) пам’ять.

Це є однією з причин чому статично типізовані мови як C та C++ виконуються швидше. Інша причина полягає в тому що у Python присутній Global Interpreter Lock – алгоритм що дозволяє одночасно виконуватись лише одному потоку, навіть при багатопоточній архітектурі та наявності більше одного ядра процесора. У результаті багатопоточні програми можуть виконуватись повільніше ніж за однопоточну.

Іншим основним недоліком використання Python є відсутність набору інструментів та технологій для розробки мобільних додатків. Розробка для платформ Android та iOS є практично недоступною на даний момент.

Попри те наявні декілька бібліотек (такі як Kivy та Beeware), що розроблені для втілення мети розробки мобільних додатків, вони не мають такого ж широкого застосування та функціоналу як Java, Kotlin або Swift.

Ці недоліки є суттєвими для мови програмування загального призначення, проте у контексті даної роботи втрачають свою критичність. Розробка мобільного додатку не є частиною поставленої задачі, натомість головною задачею є розробка серверної частини.

Основною частиною розроблюваного алгоритму є виконання значної кількості обчислень над аудіо даними, тому швидкість обчислень є критично важливим показником. Як зазначено вище, Python є повільнішим за статично типізовані мови програмування та за декілька динамічно типізованих мов.

Хоча Python не є оптимальним для обчислень, широкого використання набула бібліотека Numpy, розроблена у 2005-му році з метою створення інструменту для швидких арифметичних обчислень. Наразі дана бібліотека

використовується для різних галузей роботи з даними, зокрема для тренування моделей машинного навчання.

Є декілька причин високих показників швидкості Numpy:

- Масив Numpy є набором однакових типів даних що тісно розміщені у пам'яті. Як було зазначено вище, стандартний масив Python може містити дані різних типів що обмежує та сповільнює процес обчислень.
- Numpy має змогу розділяти завдання на декілька під завдань та обчислювати їх паралельно.
- Функції Numpy імплементовані на мові програмування C. Дана мова є більш низькорівневою і в результаті масиви Numpy є швидшими за стандартні масиви мови Python. [11]

Для порівняння швидкості виконання стандартних операцій за допомогою Python та Numpy доцільно дослідити час виконання цих операцій на даних різного розміру.

Під час тестування швидкості операції додавання створено масиви різних розмірів та виконано операцію додавання скаляру до кожного елементу. При розмірі масивів у десять тисяч елементів Numpy виконує дане обчислення у 5 разів швидше. Рис. 2.1 ілюструє у скільки разів швидше виконується обчислення даного типу інструментами Numpy за стандартний Python залежно від розміру вхідних даних.

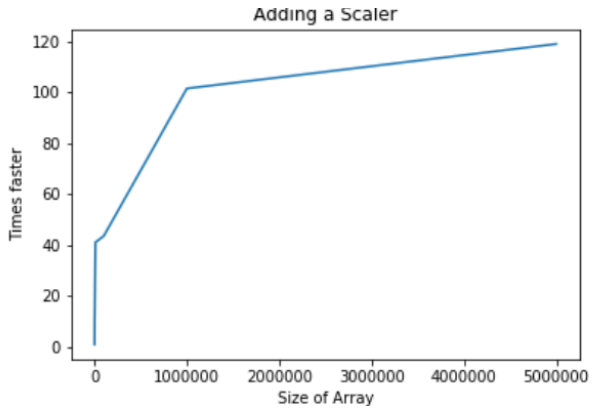


Рисунок 2.1 – Порівняння часу виконання операції додавання засобами Numpy та Python

З графіку можна визначити що при наближенні розміру масиву до 5,000,000 Numpy виконує обчислення у 120 разів швидше. Це пояснюється тим що зі збільшенням розміру масиву Numpy може виконувати більшу кількість операцій паралельно.

Для виконання операції множення твердження щодо переваги Numpy зберігається. З графіку (рис. 2.2) можна зробити висновок що на великих розмірах даних Numpy виконує операції у 140 разів швидше, що є суттєвим показником.

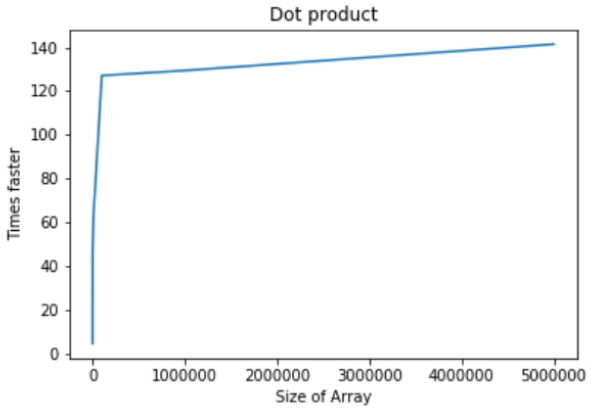


Рисунок 2.2 – Порівняння операції множення засобами Numpy та Python

Існують також операції що мають оптимальний час виконання стандартними засобами Python. Прикладом однією з таких операцій є конкатенація двух масивів. На графіку (рис. 2.3) можна аналітично визначити що час виконання конкатенації є майже однаковим.

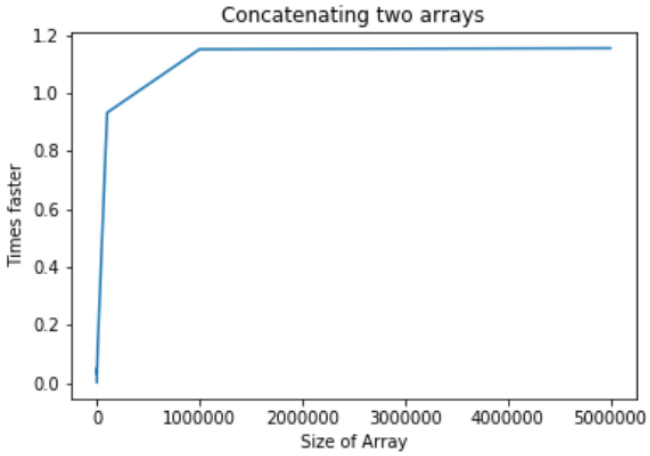


Рисунок 2.3 – Порівняння часу виконання операції множення засобами Numpy та Python

Варто зазначити що під час виконання даної операції Numpy створює третій, результуючий масив в той час як Python дозволяє розширити існуючий.

2.2 Огляд технологій та бібліотек

Для розробки системи важливо дослідити технології що стануть основою розроблених алгоритмів. Так як основною задачею системи є аналіз та попередня обробка аудіо, варто розглянути бібліотеки та інструменти що допоможуть як аналізувати дані так і зберігати їх у належному нормалізованому форматі.

2.2.1 Librosa

Область дослідження музичної інформації широко охоплює галузі музикознавства, цифрої обробки сигналів, машинного навчання та пошуку інформації. На початку розвитку цієї галузі переважна частина досліджень MIR була проведена на замовлення інструментів та сценаріїв для різних музичних платформ, та розроблена за допомогою таких мов програмування як MATLAB або C++. При цьому стабільність, масштабованість і простота використання цих інструментів часто не є досконалою. [12]

Популярною альтернативою серед дослідників MIR наразі є Python. Основними факторами що цьому сприяють є наявність якісних бібліотек для роботи з даними та машинного навчання разом з інструментами для роботи з веб даними та текстом. З появою потреби у централізованій бібліотеці що надає базовий набір рутин для MIR, було розроблено librosa, Python пакет для обробки аудіо сигналів.

Однією з конвенцій librosa є відкритість усіх важливих параметрів функцій користувачу. Це надає змогу гнучко використовувати дані функції для імплементації комплексних алгоритмів, хоча ця конвенція ускладнює

взаємодію з інтерфейсом для користувачів що потребують бібліотеку для простих найпоширеніших методів аналізу аудіо.

Аудіо сигнал у librosa представлений одновимірним масивом numpy. Зазвичай сигнал має характеристику SR що означає частоту (виміряну у Гц), з якою виміряно даний сигнал.

Таким чином тривалість сигналу у librosa може бути виміряна, поділивши довжину масиву що репрезентує сигнал, на SR (рис. 2.4).

```
>>> duration_seconds = float(len(y)) / sr
```

Рисунок 2.4 – Приклад роботи з представленням аудіо сигналу у librosa

За замовчуванням, при завантаженні стерео аудіо файлів, функція *librosa.load()* конвертує дані у моно сигнал шляхом знаходження середнього значення між лівим та правим каналами та встановлює SR у стандартне значення 22050 Гц.

Так як більшість методів аналізу аудіо проводять обчислення не на окремих значеннях сигналу, а на фреймах («вікнах»), що вичисляються за певним кроком. За замовчуванням один фрейм складають 2048 значення, а крок що визначає зміщення становлять 512 значень сигналу. Якщо представити це значення у одиницях вимірювання часу, ми отримаємо 93 та 23 мілісекунди відповідно. На практиці фрейм з індексом t відповідає масиву значень сигналу, визначеному за індексами за принципом ілюстрованим на рис. 2.5.

```
y[(t * hop_length - frame_length / 2):
  (t * hop_length + frame_length / 2)],
```

Рисунок 2.5 – Приклад роботи з представленням аудіо сигналу у librosa

Більшість функцій аналізу аудіо, імплементованих у librosa, повертають двохвимірний масив типу *numpy.ndarray*. Також за замовчуванням типи аналізу що працюють з назвами нот, працюють з розділеним на 12 рівних частин спектром, що відповідно репрезентує усі 12 значень нот, і налаштовані

таким чином що базова нота A відповідає частоті 440.0 Гц. Результати частотного аналізу з хроматичними значеннями налаштовані таким чином що нульовий проміжок спектру відповідає ноті C.

Основний функціонал бібліотеки знаходиться у підмодулі *librosa.core*. Загалом функціонал даного під модуля можна розділити на чотири категорії: операції над аудіо та часовим представленням аудіо, обчислення спектрограм, конвертація часу та частот, і операції над висотою звуку. Для зручності усі операції у під модулі взаємозамінні на верхньому рівні ієрархії пакетів, тобто *librosa.core.load* відповідає функції *librosa.load*.

Перша категорія функціоналу включає такі функції як: читання аудіо з диску за допомогою пакету *audioread*, зміна величини SR за допомогою *core.resample,* конвертація зі стерео у моно сигнал за допомогою *core.to_mono*, обчислення автокореляції за допомогою *core.autocorrelate* тощо.

Операції, пов’язані зі спектрограмами, включають STFT, зворотню STFT та миттєву спектрограму частот (ifgram), що разом забезпечують основний функціонал для аналізу аудіо частот. Також ефективна трансформація константи Q імплементована таким чином, що повертає логарифмічно розміщені представлення частот, що є придатними для подальшого тонального аналізу сигналу.

Також *logamplitude* є гнучкою та надійною імплементацією відношення тональності до логарифмовано нормалізованих частот, що може бути корисним для запобігання недостатньої кількості даних сигналу та додавання шумового фону для роботи з лінійною амплітудою.

Через те що існує декілька варіантів представлення даних як одиниць часу та частоти, librosa надає широкий вибір функцій для зручності конвертації між різними репрезентаціями часу: секунд, фреймів або одиниць сигналу, та представленнями частоти: Герц, базовий індекс константи Q, базовий індекс трансформації Фур’є, базовий індекс Мел спектрограми, номер ноти у MIDI представленні або нота у одній з нотатник систем – американській системі.

Також даний підмодуль забезпечує функціоналом для оцінки переважаючої частоти проміжків STFT за допомогою параболічної інтерполяції (*piptrack*), і оцінка відхилення від основної частоти (виміряна у одиницях - центах) по відношенню до вище зазначеної ноти A. Такий функціонал дозволяє проводити тональний аналіз щоб динамічно застосовувати фільтри для визначення загального відхилення тону аудіо сигналу.

Спектральні представлення – розподіл енергії щодо набору частот – складає основу багатьох технологій аналізу в MIR та обробки цифрових сигналів загалом. Модуль *librosa.feature* імплементує набір спектральних представлень, більшість яких засновані на функції STFT.

Мел спектрограма є часто вживаною для представлення аудіо сигналу, так як вона надає базову модель для симулювання сприйняття частот людським вухом. І звичайна Мел спектрограма *librosa.feature.melspectrogram* і широко вживана MFCC, імплементована як *librosa.feature.mfcc,*становлять частину модулю. За замовчуванням, ці функції налаштовані імплементувати алгоритм Slaney але можуть працювати за алгоритмом моделі Маркова за допомогою встановлення параметру *htk* у значення True.

Так як дане аудіо представлення часто використовується для визначення характеристик тембру музики, вони є прийнятним рішенням для представлення тонального аспекту аудіо.

Альтернативою Мел представленню є хроматичне (далі – хрома) представлення. Воно використовується для визначення характеристик гармонії без врахування тембру, висоти октави та гучності. Доступні дві гнучкі імплементації хрома трансформації:

- *chroma_stft* – використовує STFT аналіз;
- *chroma_cqt* – використовує трансформацію константи Q.

Альтернативною репрезентацію можна отримати використавши функцію *tonnetz*, що обчислює тональну центроїду як координати у шестивимірному просторі використовуючи метод Харта. [18]

Детальніше про особливості різних представлень та їх практичну цінність досліджено у наступному підрозділі.

Також даний під модуль включає функціонал що імплементує часто вживані трансформації масивів часу у MIR. Це включає функцію *delta*, що обчислює похідну часу, *stack_memory*, що конкатенує вхідний маив тональних представлень зі своєю копією віддвленою в часі, що дозволяє змоделювати інший тип представлення, та *sync*, що дозволяє застосувати агрегатну функцію (наприклад, *numpy.mean* або *median*) до окремих колонок.

Модуль display надає прості інтерфейси для візуального представлення аудіо даних через бібліотеку *matplotlib*.

1. Функція *display.waveplot* вимальовує амплітуду аудіо сигналу використовуючи функцію matplotlib's fill_between. При цьому для більшої ефективності сигнал конвертується з меншим значенням SR. Моно сигнали зображені горизонтально, симетрично до осі X, стерео сигнали зображені з амплітудами двох каналів симетрично по різні боки цієї осі.
2. Функція *display.specshow* слугує обгорткою для функції *matplotlib imshow* та слугує для візуалізації спектрограм та динамічно визначає набори кольорів відповідно до типів та обсягу даних. Також дана функція надає вибір акустично змістовних назв осей та параметрів масштабування.

Останніми технологіями що заслуговують особливої уваги є підмодулі onset та beat. Функції описані вище описують інформацію пов'язану з частотами, а часові характеристики аудіо є однією з найважливіших областей MIR. Аналізувати сигнали відносно абсолютного часу не є оптимальним способом роботи з ними, натомість сигнали індексуються відносно бітів або

сильних нот. Дані підмодулі імплементують функції для обчислення різних парметрів часового аспекту музики:

- Модуль onset включає дві функції: *onset_strength* та *onset_detect*. Перша функція обчислює спектральні викиди зі вхідної спектрограми та повертає одновимірний масив що репрезентує збільшення спектральної енергії на кожному фреймі. Це проілюстровано синьою лінією на рис. 2.6. Функція *onset_detect* з іншого боку визначає піки даного масиву. Значення, що повертаються, відповідають червоним крапкам на рис. 2.6.
- Модуль *beat* надає функціонал для оцінки загального темпу (BPM) та позиції бітів за результатами функції *onset_strength*. Точніше, спочатку обчислюється BPM, та на основі нього обираються піки що розташовані на схожій до значення темпу відстані. Визначені біти відповідають зеленій лінії на рис. 2.6.

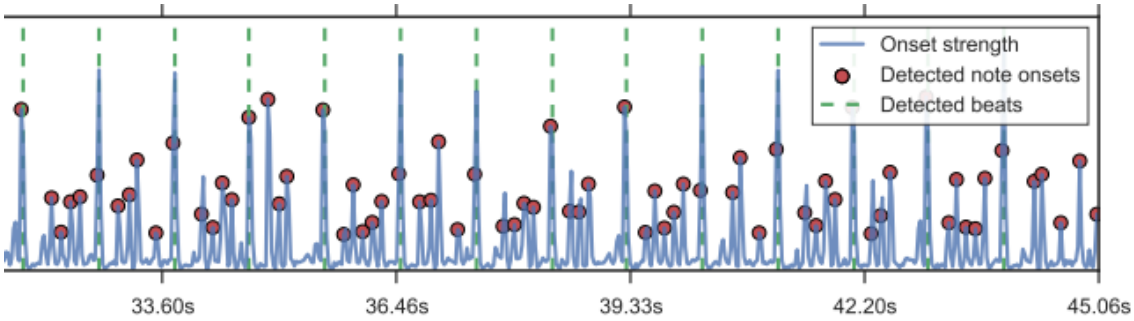


Рисунок 2.6 – Демонстрація принципу роботи підмодулів *librosa onset* та *beat* [12]

Таким чином BPM та позиції бітів можуть бути обчислені за допомогою коду такого зразка (рис. 2.7):

```
>>> oenv = some_other_onset_function(y, sr)
>>> librosa.beat.beat_track(onset_envelope=oenv)
```

Рисунок 2.7 – Приклад роботи з *beat track librosa*

При цьому необхідно передати параметром функції користувацьку функцію визначення спектральних викидів.

Як було зазначено вище, функції цього під модуля повертають позиції не в значеннях абсолютного часу, а у індексах фреймів. Це спрощує взаємодію з іншими функціями бібліотеки.

Окрім розглянутих модулів, дана бібліотека також має засоби для визначення структури аудіо сигналу (куплети приспів і т.д.), та для застосування ефектів над аудіо. Дані функції не несуть великої практичної цінності у контексті даної роботи,тому не грають великої ролі.

2.2.2 PyAudioAnalysis

PyAudioAnalysis є пакетом розробленим для завдань аудіо аналізу [17]. Даний пакет дозволяє:

1. Отримувати декілька видів тональних представлень (таких як MFCC, спектрограми, тональні представлення) у режимі CLI. Підтримується візуалізація та збереження результатів.
2. Тренувати, налаштовувати параметри та оцінювати системи класифікування аудіо сегментів.
3. Класифікувати незнайомі звуки.
4. Визначати та виключати довгі паузи з довгих аудіо записів.
5. Проводити ручну сегментацію аудіо.
6. Проводити автоматичну сегментацію з метою вилучення відбитку аудіо.
7. Тренувати та використовувати моделі аудіо регресії. Дана модель знаходить використання у розпізнаванні таких аспектів музики, як емоції.
8. Візуалізувати дані за допомогою стиснення їх вимірів. [19]

Даний пакет оптимальний для ознайомлення з прогнозуванням аудіо характеристик за допомогою побудованих моделей машинного навчання. Проте функціонал пов’язаний з попередньою обробкою сирих аудіо даних та

обчислення їх тональних та, що важливіше, часових характеристик, обмежений у порівнянні з *Librosa*.

2.3 Огляд БД та інших способів зберігання даних

Для вибору методу зберігання даних потрібно чітко окреслити формат та структуру цих даних для визначення оптимальних вимог до засобів зберігання та доступу до них. Кінцевим результатом розробленої системи є об’єкт що зберігає метаінформацію про певне аудіо, а саме: назву, тональність, BPM, SR, розмір фрейму та обчислене тональне представлення.

У той час як більша частина цих даних може бути збережена у нормалізованому вигляді, зі збереженням принципу атомарності, тональне представлення, в свою чергу, містить великий об’єм даних що зберігаються в двовимірних масивах значної довжини. Порівняння характеристик аудіо та типами даних що представляють їх у системі, наведено у табл. 2.2. Варто зазначити що за стандартний файл сприймається аудіо твір до 5 хвилин.

Таблиця 2.2 – Обчислені параметри аудіо та їх тип даних у реалізації

Параметр	Тип даних	Розмір
Назва	String	До 800 Б
Тональність	String	До 16 Б
BPM	Int32	4 Б
SR	Int32	4 Б
Розмір фрейму	Int32	4 Б
MFCC	List – двовимірний для кожного семплу з вимірами (розмір семплу, 13)	8000 – 10 000 Кб

Кінець таблиці 2.2

Параметр	Тип даних	Розмір
Мел спектрограма	List – двовимірний для кожного семплу з вимірами (розмір семплу, 128)	60 000 – 100 000 Кб

Таким чином, в залежності від обраного формату представлення, файл з даними може мати розмір 10 Мб або навіть 60 – 100 Мб. Постає вибір чи варто зберігати файли такого розміру у реляційній БД.

2.3.1 Зберігання файлів у БД – огляд практики та аналогів

Зберігання файлів у БД вважається неоптимальною практикою. Хоча БД мають на меті структуроване зберігання даних, є декілька суттєвих недоліків, що виділяють щодо зберігання у ній файлів:

- Повільніші запити у базу даних. Зберігання файлів у БД сповільнює загальну швидкість системи через те що більший об’єм даних передається між за стосунком та БД. Також файли використовують RAM для покращення ефективності. Дані до яких часто роблять запит, зберігаються у RAM тому що таким чином їх читання займає суттєво менше часу. Під час запиту до файлів БД доводиться тримати їх у RAM. Сервери зазвичай мають обмежену кількість RAM тому деякі дані будуть мати пріоритет перед іншими – що означає що у процесі доступу до файлів важливі ресурси можуть бути недоступними для інших запитів.
- Підтримка БД стає складнішою з ростом її розміру. Якщо БД має великий розмір, такі операції як створення бекапів, індексування та де фрагментація БД стають суттєво повільнішими та мають вищу ймовірність помилки. Також при роботі з дублікатом БД

можуть виникати проблеми з синхронізацією даних (race conditions).

- Зберігання файлів є складнішим за зберігання простих типів даних, так як вони потребують конвертування. Якщо зберігати файл у текстовому вигляді, можна конвертувати його у формат *base64*. Для цього потрібно імплементувати логіку перетворення файлів у даний формат при зберіганні їх у БД та навпаки – конвертація з *base64* у вихідний формат перед надсиланням файлу клієнту. Це додає шар програмного забезпечення, що є неоптимальним та може стати джерелом помилки.
- Також файли, що зберігаються у форматі *base64*, займають на 33% більше місця ніж їх оригінальний розмір. [22]

Таким чином зберігання файлів розроблюваної системи у БД є неоптимальним. Одним з аналогів є зберігання файлів локально або у хмарі. У обох випадках необхідно зберігати у БД шлях до файлу у файловій системі.

Файлова система є популярним аналогом. Зберігати файли локально недалеко від коду за стосунку є зручною практикою, так як не потрібно звертатись до зовнішнього провайдеру для читання або запису даних.

Python має широкий набір інструментів для роботи з файловою системою та для серіалізації даних (зокрема, пакет json). Тому у даній роботі прийнятно та оптимально зберігати дані у json форматі на диску.

2.3.2 – Redis

Так як обчислення займають певний значний проміжок часу, для оптимізації роботи системи доцільно імплементувати механізм що пришвидшить роботу сервісу, зберігаючи результати даних обчислень для можливого повторного використання.

Redis (Remote Dictionary Server – віддалений сервер-словник) є швидким сховищем даних типу «ключ – значення» що зберігається у пам’яті. Він використовується у якості БД, кеша та черги.

Redis забезпечує час відповіді на рівні долі мілісекунди та дозволяє за стосункам, працюючим у режимі реального часу, здійснювати міліони запитів у секунду. Завдяки високій ефективності Redis широко застосовується для кешування, керування сеансами, аналітики у режимі реального часу і т.д.

Даний сервер має назву словника тому що типи даних у його основі схожі на фундаментальні типи даних у мовах програмування: строки, списки, словники та колекції. Redis порівняно з іншими нереляційними БД, підтримує структури даних найближчі до нативних структур даних що розробники використовують у за стосунках та алгоритмах. Простота використання робить цю систему оптимальною для швидкої розробки та високо ефективних за стосунків, так як основні структури даних можуть легко передаватись між процесами та сервісами.

Як було зазначено, Redis зберігає дані у пам’яті, періодично зберігаючиїх на диск. Таким чином, Redis може слугувати як класична БД, так і для кешування. Коли пам’ять Redis закінчується через заповнення даними, він повертає помилку, але може бути налаштований відкидати старі менш важливі дані та записувати нові. У обох випадках основним обмеженням використання Redis є розмір доступної пам’яті. [23]

У даній роботі Redis можна застосувати для кешування результатів обробки аудіо даних таким чином що при повторному зверненні щодо аналізу аудіо твору, система не буде обчислювати характеристики а виконає запит до серверу Redis де зберігається хеш назви твору та шлях у файловій системі до файлу з результатами, та поверне даний шлях у випадку якщо такі дані існують. Обмеження щодо тимчасовості зберігання даних та максимальної кількості даних не є критичними оскільки результати обчислень не мають зберігатись у швидкому доступі для повернення клієнтові – кінцева ціла

система не буде повертати тональних представлень, а лише результати роботи підсистеми що ідентифікує акорди. Кешування покликане покращити ефективність роботи системи попередньої обробки аудіо за рахунок ймовірності що популярні музичні твори не будуть повинні обчислюватись повторно.

Алгоритм кешування виглядає наступним чином:

1. Здійснюється запит до розроблюваного сервісу
2. Здійснюється *get* запит до сервера Redis
3. Якщо вже існує запис з ім’ям твору, повертається шлях до результатів.
4. Якщо запису немає, дані обчислюються системою та створюється новий запис за допомогою *set.*

Даний алгоритм проілюстровано на рис. 2.7.

Рисунок 2.7 – Послідовність дій простого алгоритму кешування

2.4 Огляд алгоритму компоненту визначення тональності

Компонент визначення тональності аудіо є важливим для подальшого визначення складових гармонії твору та ідентифікаці акордів. Розглянемо можливі варіанти знаходження цього компоненту:

- Читання параметру тональності з метаданих вхідного файлу. Серед великої кількості аудіо форматів лише MIDI формат зберігає характеристику тональності *Key Signature*, тому не є прийнятним варіантом.
- Імплементація алгоритму що визначатиме тональність на основі хроматичного тонального представлення аудіо – алгоритму Крюмхасля Шмукхлера. [24]

Даний алгоритм використовує тональне представлення аудіо – хромаграму, що відповідає за нотне значення звуку (одну з 12 нот).

Алгоритм КШ має в основі відомі обчислені «патерни» – профілі хроматичного представлення музичного твору, що представляють відношення тону у хроматичній тональності до його відносної ваги. На рис. 2.8 та рис. 2.9 проілюстровано патерни для мажорних та мінорних тональностей, утворених у результаті експериментальної аналітичної оцінки важливості тону у тональності деякою кількістю експертів.

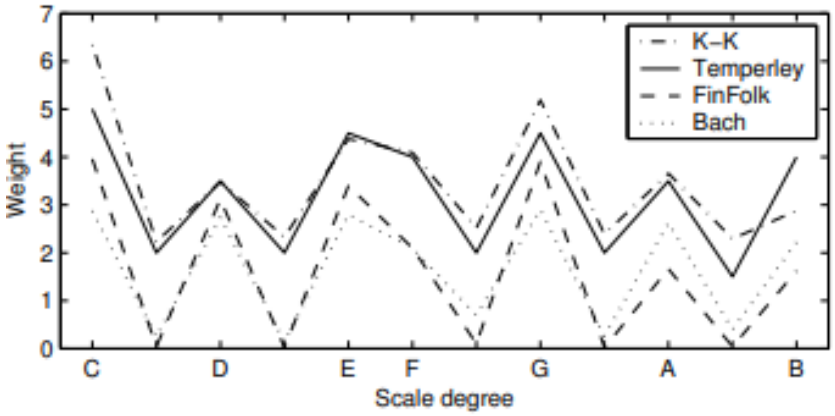


Рисунок 2.8 – Профіль мажорної тональності алгоритму КШ

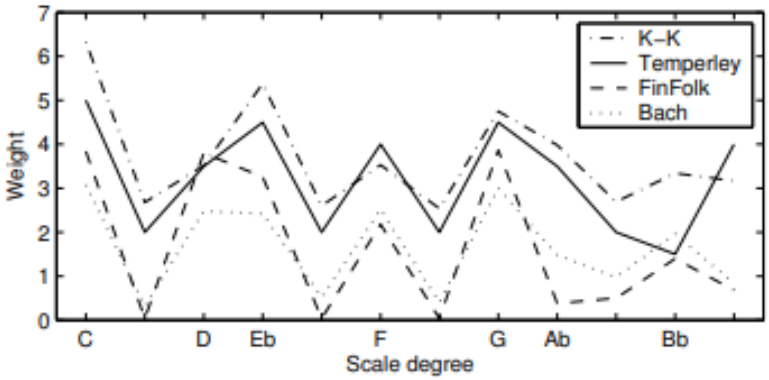


Рисунок 2.9 – Профіль мінорної тональності алгоритму КШ

Дані профілі обчислені для базової тональності С але зберігають свої властивості для інших 23-х тональностей при транспонуванні.

Суть алгоритму КШ полягає у тому що так як тональність аудіо тісно пов'язана з її профілем – сумарною вагою звуків що відповідають різним нотам – вона може бути ідентифікована шляхом обчислення коефіцієнтів кореляції між фактичним профілем музичного твору та кожним з 12 мінорних та 12 мажорних теоретичних профілів тональностей. У результаті тональність що має найвищий коефіцієнт кореляції, є найбільш ймовірно правильною для даного твору. Фактичний профіль музичного твору легко обчислюється з вхідних аудіо даних.

Даний метод є ефективним для передбачення тональності аудіо великої тривалості та з широким спектром гармонії – різними інструментами, басом тощо та є менш ефективним при обчисленні цієї характеристики для ізольованої монофонічної мелодії. Це обмеження не є суттєвим так як система розрахована на аналіз вхідних даних сучасних західних музичних творів, де зазвичай присутній широкий спектр звуків.

Таким чином можна виділити основні кроки алгоритму:

1. Отримання вхідних даних.
2. Обчислення тонального представлення – хромаграми з вхідних даних.

3. Обчислення сум значень кожної з 12 нот протягом тривалості усього аудіо – таким чином отримання профілю аудіо.
4. Знаходження коефіцієнтів кореляції між отриманим профілем аудіо та кожним з 12 профілів тональностей (рис. 2.10), де P_i є теоретичним профілем тональності та D_i є профілем досліджуваного аудіо.
5. З отриманих коефіцієнтів обрати найбільше значення – воно відповідає найбільш ймовірному результату тональності. Також можна отримати альтернативну тональність, що відповідає другому за величиною коефіцієнту.
6. Повернути отримані значення

$$cor(P_i, D_i) = \sum_j \sum_k P_i(j,k) D_i(j,k).$$ [1]

Рисунок 2.10 – Отримання коефіцієнтів кореляції у алгоритмі КШ

За допомогою інструментів *numpy* операції над масивами значень, такі як сума та обчислення кореляції, можна реалізувати ефективно та зрозуміло.

ВИСНОВОК ДО РОЗДІЛУ 2

У даному розділі було досліджено різні засоби розробки компонентів системи та обрано ті що найкраще задовольняють вимоги сервісу. Також було оглянуто алгоритми та засоби для аналізу аудіо даних.

При виборі мови програмування були враховані наступні вимоги:

- популярність використання у сфері аналізу даних;
- наявність та ефективність засобів для обробки даних;
- наявність інструментів для візуалізації даних;
- можливість реалізації компонентів, не пов’язаних з алгоритмами та обробкою аудіо – робота з БД, збирання даних і т.д.

Враховуючи ці вимоги, оптимальним вибором визналася мова Python. Її недоліки (серед них динамічна типізація) було враховано та обгрунтовано їх невелику практичну недосконаліть через наявність технологій що їх покращують та майже нівелюють.

Серед технологій розглянутих для роботи з аудіо файлами, представленнями та аналізом аудіо були бібліотеки *Librosa* та *PyAudioAnalysis*. Велика частина необхідних засобів імплементована обома пакетами, а саме візуалізація та отримання тональних представлень, проте *Librosa* надає більш широкий та низькорівневий набір інструментів для:

- обчислення темпу та ритму аудіо, виокремлення спектральних піків та бітів музики;
- візуалізації аудіо – імплементація не обмежується викликом методу з CLI, є можливість кастомізувати функції візуалізації відповідно до вимог розробника;
- трансформації аудіо сигналу.

Було обґрунтовано вибір засобу зберігання результуючих даних, а саме не практичності використання реляційної БД для зберігання обчислених

характеристик аудіо через великий розмір цих даних та необхідності зберігання їх у файлі.

Натомість було обрано альтернативу – зберігання результатів у файловій системі та кешування запитів обробки за допомогою серверу зберігання даних у пам’яті Redis.

Також було розглянуто варіанти реалізації компоненту виявлення тональності аудіо твору. З врахуванням формату даних та важливістю точності роботи цього компоненту було обрано алгоритм КШ.

РОЗДІЛ 3
ОГЛЯД РЕАЛІЗАЦІЇ СИСТЕМИ

Так як розглянуто основні теоретичні відомості та обрано технології що задовольняють вимоги системи, наступним етапом є реалізація системи. Система має задовольняти основні потреби, а саме:

- Приймати аудіо у форматі wav за допомогою зрозумілого інтерфейсу.
- Здійснювати попередній аналіз та визначати характеристики твору – тональність та BPM, що зберігатимуться для інших етапів.
- Надавати набір інструментів візуалізації для аналітичного розуміння музики розробниками.
- Визначення біту музики та формування фреймів таким чином що один фрейм відповідає тривалості аудіо від одного біта до наступного.
- Обчислення тональних характеристик (у даному випадку MFCC або Мел спетрограму) для кожного з фреймів окремо.
- Нормалізація та зберігання даних у файл та збереження його на диску для подальших операцій.

Таким чином постає потреба розробити декілька модулів щоб розбити систему на підсистеми та використовувати розроблені алгоритми незалежно один від одного у проектах де це буде необхідним.

3.1 Функціонал CoreTools

Для того щоб здійснювати обчислення над аудіо даними, потрібно створити базовий мінімальний набір інструментів для фундаментальних

операцій читання та попереднього аналізу цих даних. З цією метою доцільно створити пакет *CoreTools.py* що імплементує визначені раніше вимоги.

Найважливішою функцією цієї категорії є *load_audio*. Вона є розширенням базової функції *librosa* та приймає наступні параметри:

- *audio_data* – строкове значення що репрезентує шлях у файловій системі до аудіо файлу, над яким проводяться подальші обчислення. За конвенцією має закінчуватись на “.wav” або імплементувати об’єкт файлового інтерфейсу Python;
- *verbose* - параметр, за замовчуванням встановлений у значення *False*, що визначає необхідність логування отриманих при читанні даних про необроблене аудіо у форматі, ілюстрованому на рис. 3.1, де перше значення відповідає за кількість аудіо сигналів у творі, а друге означає SR даних;
- *sample_rate* – опціональний параметр, що визначає, з яким SR потрібно прочитати аудіо. За замовчуванням встановлений у найбільш уживане значення – 22050 (Гц);
- *offset* – опціональний параметр, що встановлюється у секундах у випадку коли потрібно прочитати значення сигналу не від початку твору, а починаючи з певного часу. За замовчуванням встановлений у 0.0, що означає що твір буде прочитаний повністю.

```
INFO: Audio reading complete, data details:  68548 22050
INFO: Audio reading complete, data details:  6570432 22050
INFO: Audio reading complete, data details:  68548 22050
```

Рисунок 3.1 – Логування функції читання аудіо даних

Load_audio повертає словник з двума полями: *signal* та *sample rate* (рис. 3.2), де перше це масив значень сигналу у часі, а друге – SR даного аудіо.

```
signal_info = {
    "sample_rate": sample_rate,
    "signal": []
}
```

Рисунок 3.2 – Формат даних що повертає функція читання

На початку функція перевіряє один з параметрів на логічну можливість. Параметр *offset* не може бути встановлений більшим ніж за значення тривалості усього аудіо. Щоб впевнитись у цьому, заданий параметр порівнюється зі значенням тривалості твору та встановлюється у 0.0, що означає читання з початку, у випадку якщо він більший або дорівнює цій тривалості (рис. 3.3).

```
if offset != 0.0:
    duration = librosa.get_duration(audio_data, sr = sample_rate)
    if offset >= duration:
        offset = 0.0
```

Рисунок 3.2 – Перевірка валідності параметру *load_audio*

Після валідації параметрів читання даних відбувається за допомогою функції *librosa.load*. Результати даної операції записуються у значення полів словника *signal_info* та повертається для подальшої обробки.

Наступні функції є основою візуального аналізу аудіо сигналів, що є важливим етапом для класифікації закономірностей та незвичних сегментів аудіо користувачем у випадку якщо результат оброблених даних не відповідає фактичному представленні звуку або є неточним.

Функції що утворюють сегмент для візуалізації даних це *plot_waveshow* , *plot_stft* та *plot_chroma.* Розглянемо детально їх реалізацію та використання. Функція що виконує схожу мету візуалізації – *generate_beatmap* - також знаходиться у іншому пакеті але не відноситься до даного пакету через особливість імплементації яку буде розглянуто у наступному підрозділі.

Функція *plot_waveshow* приймає значення сигналу *signal,* що представляє масив значень сигналу у часі і може бути отриманим зі значення поля *signal* словника, що повертається функцією *load_audio*. Параметр *sample_rate,* що за замовчуванням встановлюється, як і у всіх функцій модуля, у значення 22050, відповідає за SR аудіо даних.

Для побудови графіку використовуються функції *plt.figure* та *librosa.display.waveshow*. Для першої функції використовуються значення

вимірів графіку 15 на 5. Дані виміри зберігатимуться для кожної функції реалізованого пакету.

Детальніше ознайомитись з прикладом реалізації *plot_waveshow* можна на рис. 3.4. Дана реалізація є простою і є шаблоном для інших функцій візуалізації.

```
def plot_waveshow(signal, sample_rate = 22050):
    librosa.display.waveshow(y = signal, sr = sample_rate)
    plt.figure(figsize = (15, 5))
    plt.show()
```

Рисунок 3.4 – Реалізація функції *plot_waveshow*

Іншою функцією візуалізації є *plot_stft*. Схожій сигнатурі та принципу обчислення, нормалізації та побудови представлення даних (рис. 3.5) також відповідають функції:

- *plot_mfcc*;
- *plot_mel*.

```
def plot_mel(signal, sample_rate = 22050):
    # Implementation here
    return

def plot_mfcc(signal, sample_rate = 22050):
    mfccs = librosa.feature.mfcc(signal, sr=sample_rate)
    librosa.display.specshow(mfccs, sr = sample_rate, x_axis='time')
    plt.title('MFCC Spectrogram')
    plt.show()
```

Рисунок 3.5 – Приклад реалізації типових функцій *plot_**

Функція *plot_stft* приймає ті ж параметри що і *plot_waveshow*, але проводить декілька обчислень для перетворення та нормалізації даних:

1. За допомогою функції *librosa.stft*, що приймає параметр необробленого сигналу, обчислюється STFT.
2. За допомогою функції *librosa.amplitude_to_db* значення, отримані з попереднього кроку, конвертуються зі значень амплітуди у

значення децибел, що будуть використовуватись для ілюстрації сили звуку певної частоти відносно часу.

3. *Plt.figure* використовується для побудови графіку як для попередньої функції візуалізації.
4. Для побудови спектрограми використовується функція *librosa.display.specshow*, що приймає значення обчислені у пункті 2, SR та назви вертикальної та горизонтальної осі значеннями нормалізованого сигналу та часу відповідно.

Сигнатура та реалізація даної функції проілюстрована на рис. 3.6.

```
def plot_stft(signal, sample_rate = 22050):
    S = librosa.stft(signal)
    Sdb = librosa.amplitude_to_db(abs(S))
    plt.figure(figsize=(15, 5))
    librosa.display.specshow(Sdb, sr = sample_rate, x_axis='time', y_axis=
    plt.colorbar()
    plt.show()
```

Рисунок 3.6 – Реалізація функції візуалізації *plot_stft*

Остання функція даного пакету має таку ж сигнатуру як і попередньо розглянуті функції.

Plot_chroma використовує функцію *librosa.feature.chroma_stft* та *librosa.display.specshow*, проте натомість з параметрам вертикальної осі, що відповідає за тональне представлення аудіо.

Сигнатура даної функції представлена на рис. 3.7.

```
def plot_chroma(signal, sample_rate = 22050):
```

Рисунок 3.7 – Сигнатура стандартної функції візуалізації

Функція *plot_chroma_cqt* є варіацією попередньої функції тональної візуалізації. Вона відрізняється тим, що використовує функцію *librosa.feature.chroma_cqt* для побудови тональних значень залежно від часу.

Сигнатура функції відрізняється також тим що приймає параметр *bins* що є значенням типу *int* та визначає кількість проміжків для обчислень на октаву.

Сигнатура *plot_chroma_cqt* представлена на рис. 3.8:

```
def plot_chroma_cqt(signal,  bins = 24, sample_rate = 22050):
```

Рисунок 3.8 – Варіація сигнатури функції візуалізації

Для використання всіх функцій даного пакету потрібно користуватись даними, отриманими з функції *load_audio*. Приклад типового використання цих функцій наведено на рис. 3.9.

```
# Usage
audio_data = './samples/dope.wav'
res = load_audio(audio_data, verbose = True)

plot_chroma(res["signal"])
plot_waveshow(res["signal"], res["sample_rate"])
```

Рисунок 3.9 – Приклад використання функцій CoreTools

3.2 Функціонал KeyDetection

Модуль KeyDetection виконує задачу визначення тональності та альтернативної тональності музичного твору. Ці дані є важливими для подальшого гармонійного аналізу – визначенні акордів. Модуль включає в себе дві функції: separate_harmony та generate_key.

Функція separate_harmony слугує як функція підготовки аудіо сигналу до аналізу та виявлення тональності і є обгорткою для стандартної функції *librosa.effects.hpss.*

Separate_harmony, на відміну від функцій попередніх модулів, приймає єдиний параметр – аудіо сигнал та за допомогою *librosa.effects.hpss* розділяє його на елементи гармонії та перкусії. Це є важливим адже звуки перкусії не несуть тональної інформації і можуть слугувати шумом під час утворення нотного представлення.

Функція повертає частину аудіо сигналу що відповідає за гармонічну частину (рис. 3.10).

```
def separate_harmony(signal):
    s_harmonic, s_percussive = librosa.effects.hpss(y = signal)
    return s_harmonic
```

Рисунок 3.10 – Реалізація *Separate_harmony*

Функція generate_key імплементує алгоритм ШК та використовує вище описану функцію для більшої точності результатів. Формат даних що повертаються – словник з полями результату з найвищим показником коефіцієнту кореляції та результату з другим кращим показником (рис.3.11).

```
return {
    "key": res_key,
    "alt_key": alt_key
}
```

Рисунок 3.11 – Словник результатів імплементації *generate_key*

На відміну від функцій попереднього модуля, окрім сигналу та SR *generate_key* приймає на вхід декілька додаткових параметрів:

- *start_sample* та *end_sample* – індекси окремих значень сигналу, що дозволяють проаналізувати частину аудіо. Дана можливість є корисною коли аудіо може містити декілька частин з різними тональностями або модуляцією;
- *bins* визначає кількість значень у хроматичному представленні на октаву.

Для обчислення тональності відповідно до алгоритму КШ необхідно зберігати теоретичні значення профілів – вони зберігаються як константи у модулі (рис. 3.12):

```
KRUM_MAJ = [6.35, 2.23, 3.48, 2.33, 4.38, 4.09, 2.52, 5.19, 2.39, 3.66, 2.29, 2.88]
KRUM_MIN = [6.33, 2.68, 3.52, 5.38, 2.60, 3.53, 2.54, 4.75, 3.98, 2.69, 3.34, 3.17]
```

Рисунок 3.12 – Масиви профілів тональностей КШ

Функція імплементує алгоритм КШ наступним чином:

1. Отримує вхідні параметри та використовує separate_harmony для відокремлення гармонійної частини.
2. Валідує значення *end_sample* таким чином щоб оброблюваний проміжок аудіо мав тривалість не менше 1 мс.
3. За допомогою функції перетворення librosa.feature.chroma_cqt обчислюється хроматичне представлення частини сигналу, слайса з індексами початку та кінця сигналу.
4. Обчислюється сума сигналів для кожної з 12 нот, використовуючи функцію np.sum та формується словник з назвами полів відповідно нот американської системи нотації та відповідними їм значеннями суми сигналу.
5. Для кожного з 12 хроматичних значень нот формується його власний профіль наступним чином: зміщується масив тональних сум відповідно до зміщення цієї ноти відносно ноти C. Цей профіль порівнюється з теоретичними значеннями еталонного профілю для тональності C – за допомогою функції *np.corrcoef* обчислюється коефіцієнт кореляції між ними (рис. 3.13). Чим вищий коефіцієнт, тим більша ймовірність що профіль відповідає профілю певної тональності.

```
for i in range(pitch):
    key = []
    for j in range(pitch):
        c = CHORDS[(i + j)%12]
        key.append(freqs[c])
    corr_maj = np.corrcoef(KRUM_MAJ, key)[0, 1]
    corr_min = np.corrcoef(KRUM_MIN, key)[0, 1]

    cors_maj.append(corr_maj)
    cors_min.append(corr_min)
```

Рисунок 3.13 – Частина імплементації алгоритму КШ

6. У результаті отримуємо словник зі значеннями коефіцієнтів кореляції для кожної з 24-х тональностей. За допомогою функції max визначаємо назви двух тональстей з найкращими показниками та повертаємо їх як значення словника.

Сигнатура та частина імплементації (а саме виокремлення гармонійної частини, валідація параметрів, обчислення суми сигналу) *generate_key* проілюстрована на рис. 3.14:

```
def generate_key(signal, sample_rate, start_sample = 0, end_sample = 0, bins = 24):
    signal = separate_harmony(signal)

    if (end_sample == 0) or (end_sample < sample_rate/100):
        end_sample = len(signal)

    chroma = librosa.feature.chroma_cqt(y = signal[start_sample: end_sample], sr=sample_rate, bins_per_octave=bins)
    pitch = 12

    chroma_sum = []
    for i in range(pitch):
        chroma_sum.append(np.sum(chroma[i]))
```

Рисунок 3.14 – Сигнатура та частина імплементації *generate_key*

Використання функції є стандартним – потрібно використати функцію load_audio та передати параметри аудіо у *generate_key* (рис. 3.15):

```
# Usage

audio_path = './samples/CD 1 - 02, Alejandro.wav'
signal_info = load_audio(audio_path)
signal = signal_info["signal"]
sample_rate = signal_info["sample_rate"]

keys= generate_key(signal, sample_rate, end_frame = 20 * sample_rate)
```

Рисунок 3.15 – Приклад застосування *generate_key*

У даному прикладі обчислюється тональність для перших двадцяти секунд аудіо.

3.3 Функціонал Beatmap

Модуль *Beatmap* надає функціонал для обчислення таких даних як BPM та біти аудіо, що є критичним кроком у аналізі аудіо. Модуль складається з функцій *generate_beatmap* яка слугує для отримання вище вказаних даних та

(опціонально) для візуалізації цих обчислень з метою усунення неточностей, та *beatmap_split*, що надає можливість сегментувати дані за відрізками бміж бітами.

3.3.1 Функція generate_beatmap

Функція *generate_beatmap* приймає наступні параметри:

- *signal* – масив значень сигналу у часі, який можна отримати за допомогою *load_audio;*
- *sample_rate* – SR аудіо, що отримується аналогічно;
- *enableVisual* – параметр типу *bool* що за замовчуванням встановлений у значення *False*. При значенні *True* функція не лише генерує вище описані дані, а і візуалізує сигнал разом з обчисленими результатами для аналітичного розуміння.

Частина реалізації функції проілюстрована на рис. 3.10.

```
# Set enableVisual to True if visualization is neccessary

def generate_beatmap(signal, sample_rate = 22050, enableVisual = False):
    onset_frames = librosa.onset.onset_detect(signal, sr = sample_rate)

    bpm, beats = librosa.beat.beat_track(y = signal, sr=sample_rate)

    beatmap_info = {
        "BPM": bpm,
        "beats": beats
    }

    if enableVisual:
        # Displaying melspectrogram
```

Рисунок 3.10 – Частина реалізації *generate_beatmap*

Generate_beatmap повертає словник з двума полями: *BPM* та *beats*, де перше це значення, виміряне в ударах на хвилину, а друге – масив бітів у часі що репрезентується індексами фреймів (за замовчуванням розмір фрейму становить 512 сигналів).

Розмір фрейму представлений у більшості функцій як параметр *hop_size* за конвенцією використання *librosa.*

Функція реалізована наступним чином:

1. Вхідний сигнал за допомогою функції *librosa.onset.onset_detect* аналізується на наявність спектральних вершин – можливе знаходження бітів аудіо.
2. Використовується функція *librosa.beat.beat_track* для аналізу та знаходження наступних величин: *BPM*, що є значенням типу *float* та *beats*, що є масивом значень типу *float.*
3. Якщо параметр візуалізації *enableVisual* дорівнює *True*, то за допомогою розглянутих у попередньому підрозділі базових функцій будуються та виводяться два графіки.
4. Повертається словник *beatmap_info* що містить обчислені значення

Приклад використання *generate_beatmap* наведено на рис. 3.11. Варто зазначити, щоданий приклад ілюструє випадок коли візуалізація не потрібна.

```
# Usage

audio_path = './samples/CD 1 - 02, Alejandro.wav'
signal_info = load_audio(audio_path)
signal = signal_info["signal"]
sample_rate = signal_info["sample_rate"]

bit_info = generate_beatmap(signal)
```

Рисунок 3.11 – Приклад одного з використань *generate_beatmap*

Візуалізація даних у даному випалку потребує значних обчислювальних ресурсів тому компонент візуалізації пов’язаний з функцією обчислення параметрів та не може викликатись окремо.

3.3.2 Функція beatmap_split

Дана функція допомагає структуризувати обчислені дані – вона розбиває аудіо сигнал на фрейми що містять кількість значень сигналу, що відповідає *hop_size* – за замовчуванням 512.

Подальші обчислення характеристик та представлень проводяться саме над фреймами для оптимізації роботи над ними у інших сервісах. Функція використовує *generate_beatmap* для визначення позицій бітів та розбиття сигналу на інтервали відповідно до них.

Beatmap_split повертає словник даних з полями що відповідають характеристикам аудіо та масивом фрагментів аудіо що відповідають сигналу між інтервалами (рис. 3.12).

```
track_info = {
    "BPM": beat_info["BPM"],
    "frame_size": hop_size,
    "sample_rate": sample_rate,
    "samples": []
}
```

Рисунок 3.12 – Формат результату обчислень *beatmap_split*

Фрагменти (поле *samples*) у свою чергу містять значення сигналу та індекси фреймів початку та кінця інтервалу (3.13). Ці поля необхідні для збирання фрагментів у цілий музичний твір після обчислень гармонії іншою підсистемою – вони можуть бути конвертовані у абсолютний час.

```
sample = {
    "signal": [],
    "frame_start": start,
    "frame_end": end,
}
```

Рисунок 3.13 – Формат зберігання фреймів аудіо

Розглянемо реалізацію *beatmap_split*:

- За допомогою виклику функції модуля *Beatmap* *beatmap.generate_beatmap* отримується інформація про темп та позиції бітів аудіо. Значення темпу повертається як значення поля *BPM* словника *track_info*.

- До отриманого масиву позицій бітів, що відповідає індексам фреймів, додається початкове значення нульового індексу фрейму для коректності обчислень.
- Ітеруючись по проміжках між бітами, за допомогою слайсингу виокремлюється масив значень сигналу у даному інтервалі (рис. 3.14). Результат разом з індексами фреймів додається до масиву сегментів.

```
sample["signal "] = signal[start * hop_size : end * hop_size]
```

Рисунок 3.14 – Утворення сегментів сигналу

- Якщо параметр enableVerbose, аналогічно до функцій зі схожою сигнатурою, є True, виводиться інформація про результат роботи функції.
- *Track_info* повертається для подальшої роботи над даними.

Приклад використання функції проілюстровано на рис. 3.15.

```
# Usage

signal_info = load_audio('./samples/CD 1 - 02, Alejandro.wav')

beatmap_split(signal_info["signal"], 512, signal_info["sample_rate"], enableVerbose = True)
```

Рисунок 3.15 – Приклад використання *beatmap_split*

З наведеного прикладу спостерігаємо зберігання конвенції звернення до функцій, що є сталою для різних модулів – використання *load_audio* та звернення до значень полів її результату.

3.4 Функціонал Extraction

Модуль Extraction використовує усі функції аналізу та форматування даних для комплектування та утворення файлу з метаданими та обчисленими представленнями аудіо. Модуль підтримує обчислення двох тональних

представлень – MFCC [21] та мелспектрограми та представлений двома функціями – process_mel та process_mfcc що мають схожу сигнатуру.

Розглянемо вхідні параметри функцій:

- *path* – шлях у файловій системі до файлу вхідних аудіо даних – має бути формату wav;
- *out* – шлях у файловій системі, де будуть збережені обчислені дані у форматі json;
- *enableVisual* – параметр типу bool що визначає необхідність візуалізації представлень. За замовчуванням встановлений у False через велику кількість обчислень необхідну для візуалізації аудіо твору;
- *n_mfcc* є параметром виключно для process_mfcc та визначає кількість тональних значень яким може відповідати сигнал. За замовчуванням встановлений у 13 за конвенцією librosa

Важливо також зрозуміти структуру вихідних даних адже вони є результатом роботи усіх модулів системи (рис. 3.16).

```
out_sample = {
    "name": name,
    "BPM": samples["BPM"],
    "key": key_set["key"],
    "alt_key": key_set["alt_key"],
    "frame_size": samples["frame_size"],
    "sample_rate": samples["sample_rate"],
    "mfccs": []
}
```

```
header = {
    "id": counter,
    "frame_start": int(sample["frame_start"])
    "frame_end": int(sample["frame_end"])
}
```

Рисунок 3.16 – Структура даних результатів обчислень системи

Розглянемо дану структуру та визначимо які характеристики репрезентують поля у порядку «з гори до низу»:

- name – назва аудіо твору;
- BPM – темп твору;
- key – тональність твору;
- alt_key – альтернативна тональність твору;

- frame_size – розмір фрейму, кількість значень сигналу у фреймі;
- sample rate – SR сигналу, кількість значень сигналу на секунду;
- mfccs/mels – масив представлень MFCC/Мел представлень для фреймів кожного з сегментів аудіо, що відповідають проміжкам між бітами. Кожне з цих представлень включає заголовок з метаданими про сегмент (id, та індекси фреймів що визначають сигнал у часі) та масив значень MFCC/ Мел представлень

Як було зазначено раніше, дані функції вирисовують різні модулі аналізу даних для утворення файлу з даними (рис. 3.17):

```
def process_mel(path, out, enableVisual = False):
    signal_info = load_audio(path)
    if enableVisual:
        plot_mel(signal_info["signal"], signal_info["sample_rate"])


    samples = split. beatmap_split(signal_info["signal"], 512, sample_rate= signal_info["sample_rate"])
    key_set = generate_key(signal_info["signal"], signal_info["sample_rate"])
```

Рисунок 3.17 – Сигнатура та частина реалізації однієї з функції *Extraction (Мел представлення)*

Реалізація MFCC аналогу є аналогічною (рис. 3.18):

```
def process_mfcc(path, out, n_mfcc = 13, enableVisual= False):
    signal_info = load_audio(path)
    if enableVisual:
        plot_mfcc(signal_info["signal"], signal_info["sample_rate"])


    samples = split. beatmap_split(signal_info["signal"], 512, sample_rate= signal_info["sample_rate"])
    key_set = generate_key(signal_info["signal"], signal_info["sample_rate"])
```

Рисунок 3.18 – Сигнатура та частина реалізації однієї з функції *Extraction (MFCC представлення)*

Файл з даними описаними вище зберігається на диску за допомогою функції серіалізації (рис. 3.19):

```
with open(out, "w") as f:
    json.dump(out_sample, f, indent=4)
```

Рисунок 3.19 – Імплементація зберігання серіалізованих даних

Варто зазначити, що параметр *indent* встановлюється у значення 4 для зручності візуального сприйняття даних у форматі *json*.

3.5 Реалізація кешування результатів

Для покращення ефективності та швидкості роботи системи запити до даного сервісу кешуються у Redis [15] та у разі повторного запиту характеристики аудіо не обчислюються а повертаються з існуючого файлу, шлях до якого зберігається на диску. Розглянемо компоненти системи що імплементують дане рішення:

1. Redis_connect – встановлює з'єднання з сервером Redis та повертає клієнта, за допомогою якого можна взаємодіяти з БД (рис. 3.20):

```
def redis_connect(config):
    client = {
        "config": config,
        "redis": False,
    }

    rd = redis.Redis(host = config["host"], port = config["port"])

    if rd.ping():
        client['redis'] = rd
        return client
    return False
```

Рисунок 3.20 – Імплементація підключення до Redis

2. Set_value – створює запис з ключом що відповідає нормалізованим хеш значенням обчисленим функцією *compute_hash* з назви твору та значенням шляху у файловій системі до обчислених парметрів аудіо (рис. 3.21):

```
def set_value(name, path, client, enableVerbose = False):
    key = hash.compute_hash(name, client["config"]["salt"])
    redis = client["redis"]
    set = redis.set(key, path)
    if enableVerbose:
        print("INFO: Caching result: ", path)
    return set
```

Рисунок 3.21 – Імплементація запиту до Redis

3. Get_value – здійснює запит до Redis за ключом що відповідає обчисленому функцією *compute_hash* значенню та якщо такий запис існує, повертає шлях до обчислених раніше даних.
4. Compute_hash – нормалізує назву аудіо [13] твору для уникнення повторів через різний формат назви творів та обчислює хеш [14] з даної назви за допомогою алгоритму хешування sha256 (рис. 3.22)

```
def compute_hash(name, salt):
    items = re.split(' +|; +|, +|\n|_+', name)
    items = [item.lower() for item in items]
    items.sort()
    raw_name = ''.join(items)
    hashed = hashlib.sha256((salt + raw_name).encode('utf-8')).hexdigest()

    return hashed
```

Рисунок 3.22 – Обчислення хеш ключа аудіо творів

Разом дані функції утворюють механізм зберігання результатів аудіо обробки що може суттєво покращити ефективність системи.

3.6 Реалізація інтерфейсу

Інтерфейс даної системи реалізований як CLI застосунок. Для зручності взаємодії користувача з інструментами аналізу аудіо система підтримує параметри командного рядка за допомогою якого можна визначати параметри аудіо аналізу та відображення результатів (рис. 3.23):

```
def cli_parse():
    parser = argparse.ArgumentParser(description='Audio service')

    # Add the arguments
    parser.add_argument('path', metavar='path', type=str, help='the
    parser.add_argument('output',action='store', nargs='?', metavar=

    parser.add_argument('-d', '--directory', action='store_true', he
    parser.add_argument('-v', action='store_true', help='enable logg
    parser.add_argument('-c', '--caching', action='store_true', help
    parser.add_argument('-mfcc', action = 'store_true', help = 'use

    args = parser.parse_args()
    return args
```

Рисунок 3.23 – Створення парметрів CLI

Функція *cli_parse* зчитує аргументи командного рядка та повертає дані параметри для специфікації режиму роботи системи.

3.7 Огляд та тестування розробленої системи

Так як компоненти сервісу розроблені таким чином що підтримують демонстрацію проміжних результатів, варто розглянути результати роботи різних модулів та взаємодії системи з сервісом зберігання даних Redis. Розглянемо та оцінимо представлення аудіо та результати обчислень.

3.6.1 Результати роботи CoreTools

Розлянемо та порівняємо результати обчислень та візуалізації представлень для двох популярних музичних творів жанру поп – Poker face та Alejandro виконавця Lady Gaga (далі пісня 1 та пісня 2 відповідно).

На рис. 3.24 результат обчислення *plot_waveshow*.

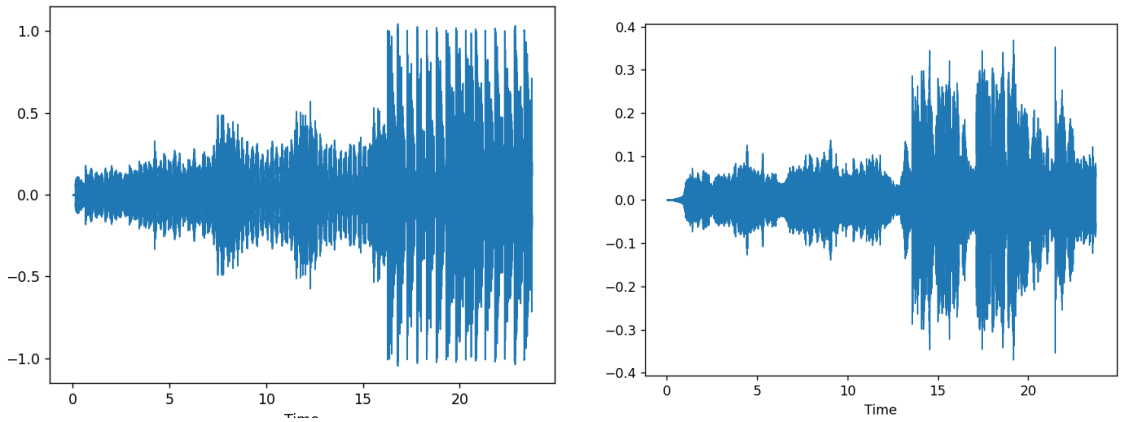


Рисунок 3.24 – Представлення *waveshow* різних творів

Завдяки візуалізації можна зробити висновок що обидва аудіо починаються тихо та не надто ритмічно, та з часом пісня 1 стає ритмічно інтенсивнішоб за пісню 2.

Варто зазначити що представлення обчислюються не для цілого твору, а для перших 20-30 секунд для простоти аналізу.

На рис. 3.25 результати *plot_chroma*:

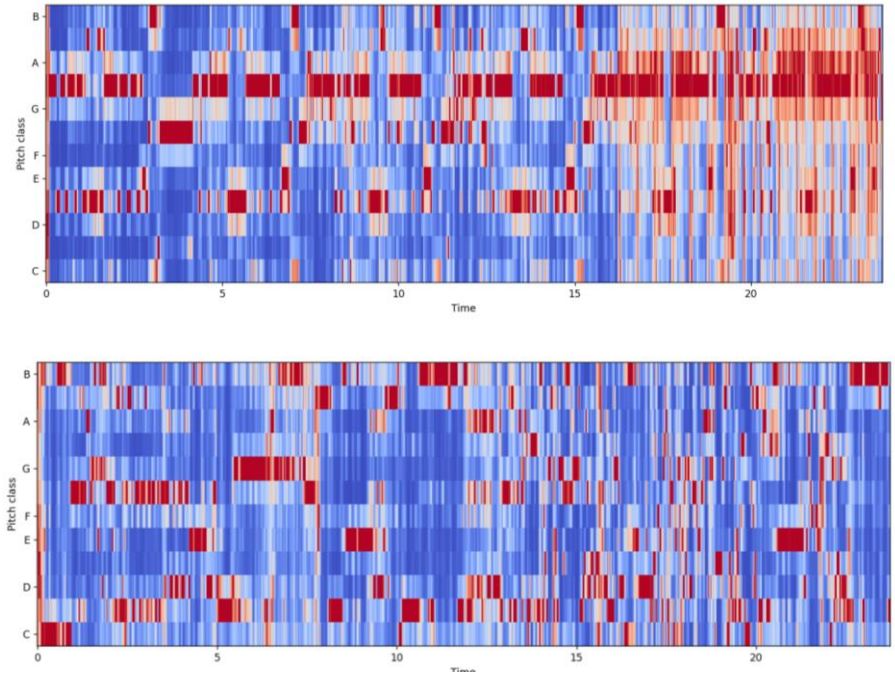


Рисунок 3.25 – Представлення *chroma* різних творів

Даний вид представлення не несе великої візуальної цінності для розуміння характеристик твору.

На рис. 3.26 представлення сили сигналів різної частоти в залежності від часу – *plot_stft.*

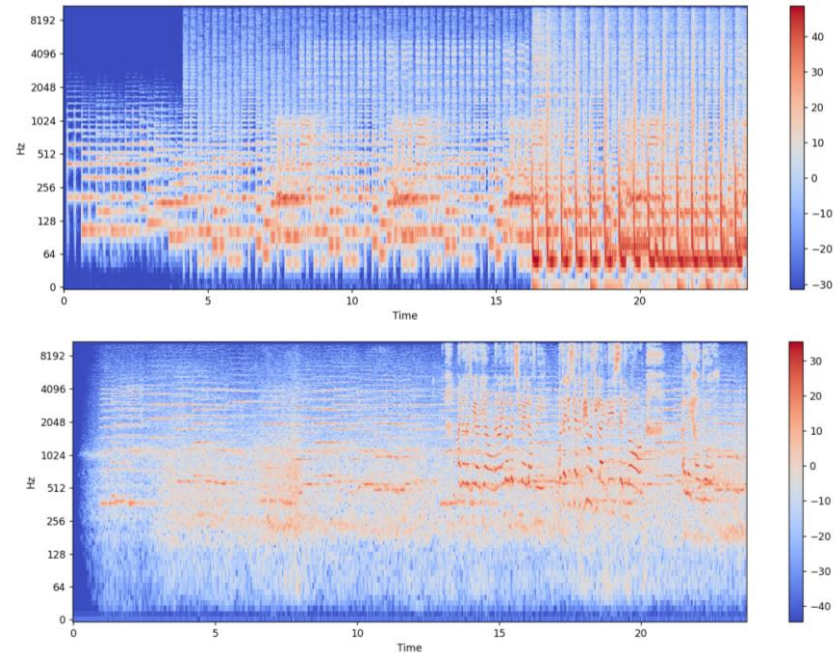


Рисунок 3.26 – Представлення *stft* різних творів

З даних представлень можна зробити висновок що пісня 1 має більш визначені низькі частоти – лінії басу.

На рис. 3.27 наведено результат обчислення хроматичного тонального представленняя *plot_chroma_cqt*

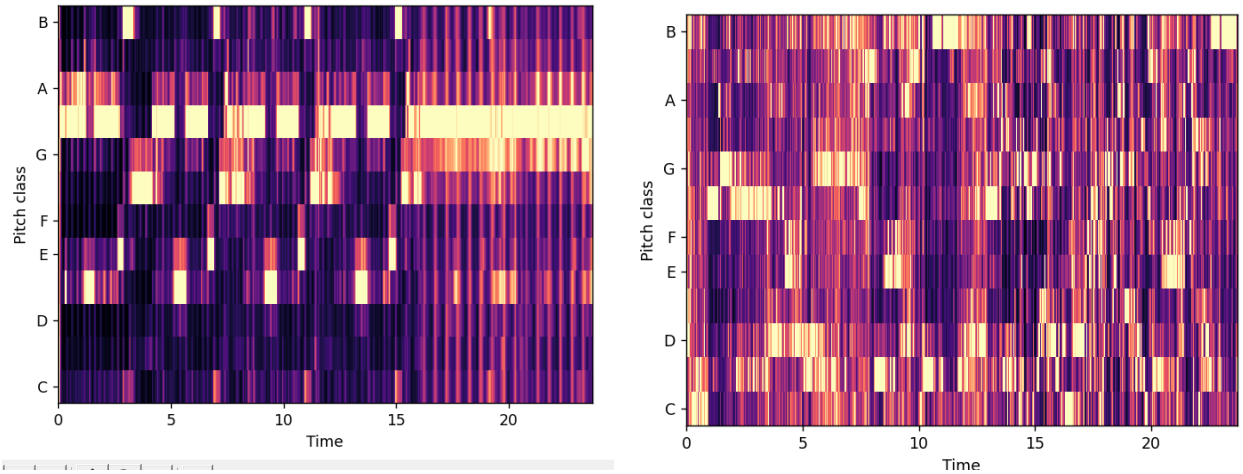


Рисунок 3.27 – Представлення *chroma_cqt* різних творів

Дане представлення ілюструє нотний розподіл сигналу – можна помітити що нота G# часто трапляється у пісні 1.

Іншим типом попереднього представлення, що враховує час, є *plot_mel,* яка враховує значення часу (рис. 3.28).

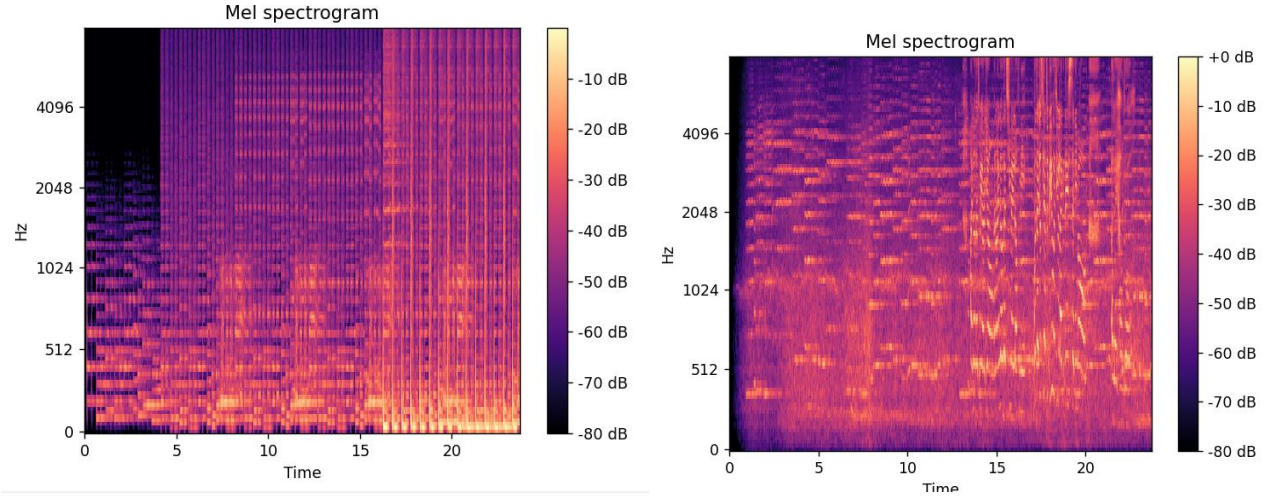


Рисунок 3.27 – Представлення *mel* різних творів

Дане представлення є наочною спектрограмою, де значення вище відповідають за звуки високих частот, а низькі – за басові значення. Можна зробити висновок що пісня 1 має більш визначений ритм та басову лінії.

Нарешті, представлення *plot_mfcc* наведено на рис. 3.29:

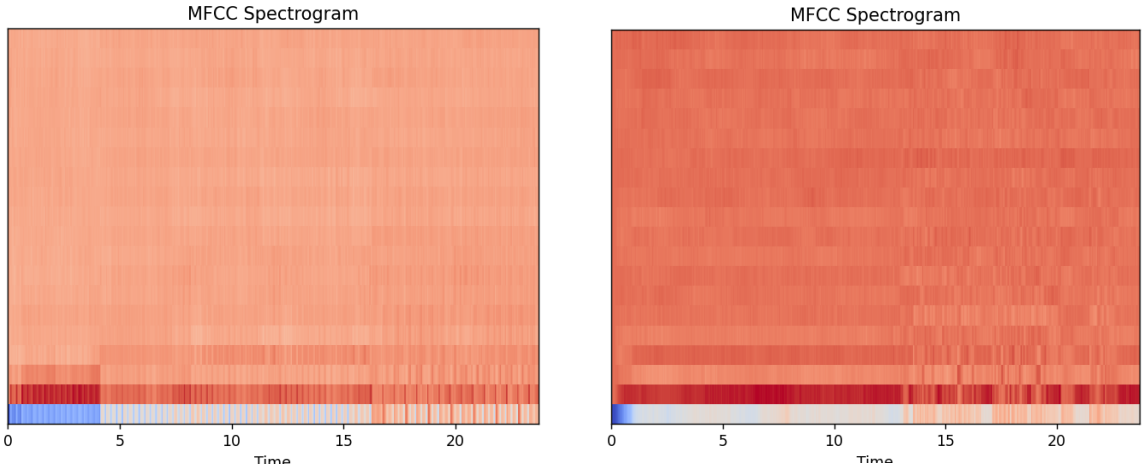


Рисунок 3.27 – Представлення *mel* різних творів

Хоча дане тональне представлення є важливим для подальшого аналізу, візуально воно не має великої цінності.

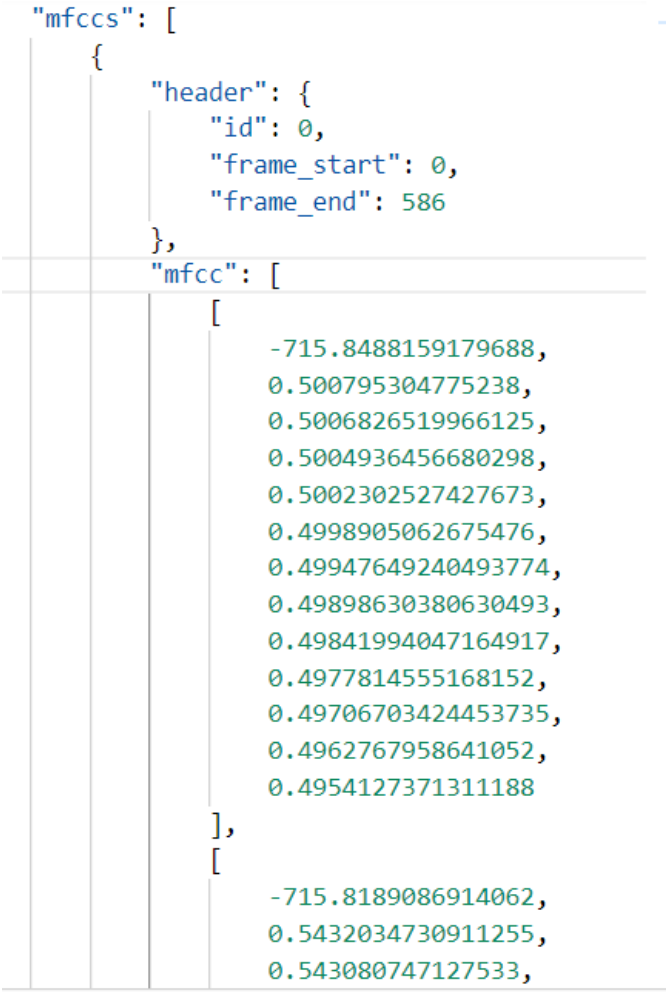
3.7.2 Результати роботи Extraction

Розглянемо результати обчислення даних модулем *Extraction*. На рис. 3.28 наведено частину отриманих праметрів для пісні 2:


```
{
    "name": "CD 1 - 02, Alejandro",
    "BPM": 99.38401442307692,
    "key": "G",
    "alt_key": "D",
    "frame_size": 512,
    "sample_rate": 22050,
    "mfccs": [
        {
            "header": {
                "id": 0,
                "frame_start": 0,
                "frame_end": 586
            },
            "mfcc": [
```


Рисунок 3.28 – Результати обчислень для аудіо твору – параметри аудіо

Результат має характеристики що відповідають дійсності – параметри аудіо спрогнозовано правильно. На рис. 3.29 наведено приклад формату у якому збережене обчислене тональне представлення:



```
"mfccs": [
    {
        "header": {
            "id": 0,
            "frame_start": 0,
            "frame_end": 586
        },
        "mfcc": [
            [
                -715.8488159179688,
                0.500795304775238,
                0.5006826519966125,
                0.5004936456680298,
                0.5002302527427673,
                0.4998905062675476,
                0.49947649240493774,
                0.49898630380630493,
                0.49841994047164917,
                0.4977814555168152,
                0.49706703424453735,
                0.4962767958641052,
                0.4954127371311188
            ],
            [
                -715.8189086914062,
                0.5432034730911255,
                0.543080747127533,
```


Рисунок 3.29 – Вміст файлу з масивом тонального представлення

Таким чином задача аналізу та виявлення характеристик реалізована коректно, та файл з результатом є надійним набором метаданих про аудіо.

3.7.3 Взаємодія з Redis

Розглянемо результати взаємодії з сервісом кешування даних. Першим кроком є запуск сервісу та утворення з'єднання з Redis для можливості здійснювати запити та зберігати дані інструментами, розробленими для цього (за умови вибору користувачем опції кешування).

На рис. 3.30 продемонстровано запуск серверу Redis:

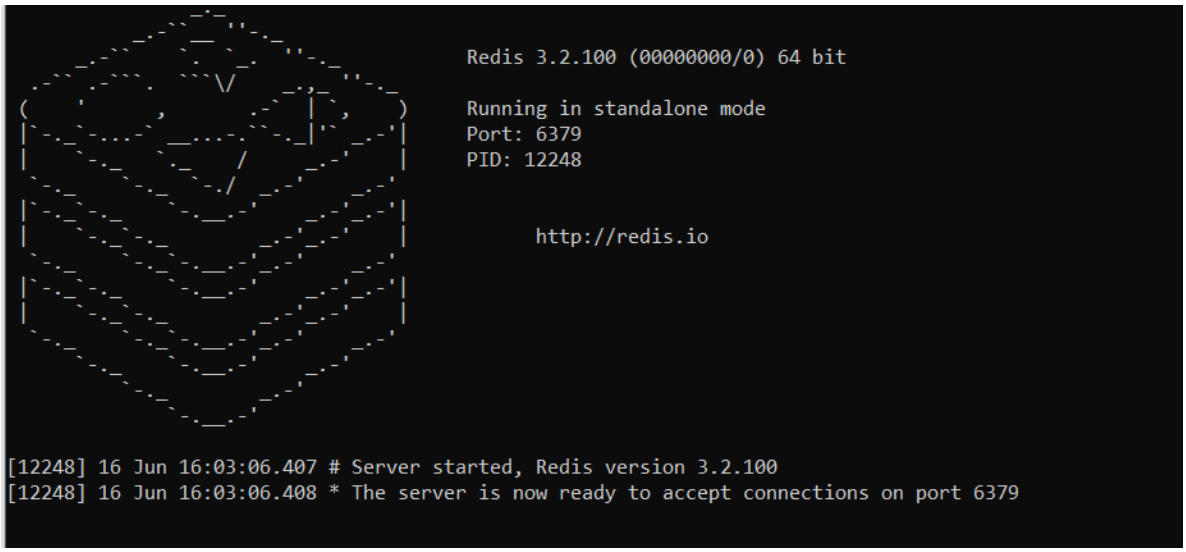


Рисунок 3.30 – Запуск Redis

Як бачимо, сервіс прослуховує запити стандартний на порт 6379. На рис. 3.31 наведений приклад перевіркі з'єднання клієнта сервісом:

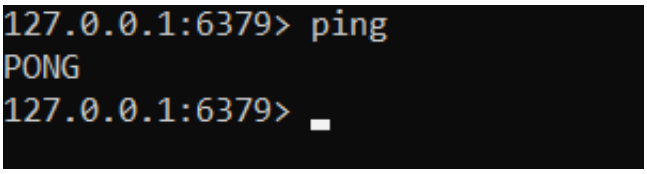


Рисунок 3.31 – Перевірка з’єднання з Redis

Сервіс відповідає, отже можна перевірити коректність модуля кешування. Приклад тестування операцій запису та отримання значень та перевірка нормалізованості зберігання незалежно від імені наведено на рис. 3.32:

```
cl = redis_connect(config)
set_value("Lady Gaga Poker Face", "testDir", cl, True)
get_value("Lady Gaga Poker Face", cl, True)

get_value("lady__gaga poker  face", cl, True)
```

Рисунок 3.31 – Тестування кешування даних

У результаті перший результат має зберегтися у БД, і повертатись як значення при запитах *get* за ключем хеш значення назви твору, навіть якщо формат назви відрізняється.

Перевіримо чи це справджується (рис. 3.32):

```
INFO: Caching result:  testDir
INFO: Accessing cashed result:  b'testDir'
INFO: Accessing cashed result:  b'testDir'
```

Рисунок 3.32 – Тестування кешування даних – результат

Визначаємо, що система працює коректно – отже, модуль може бути використаним для підвищення ефективності роботи системи.

3.7.4 Тестування компонентів системи

Модулі *beatmap* та *key_detection* покрито юніт тестами для перевірки коректності їх роботи та параметрів (рис. 3.33):

```
PS C:\Users\User\Desktop\tw> python cli.py -v --visual  .\samples\LykkeLi-aLittlebit.wav
INFO: Initializing analyser
INFO: Successfully creating a mfcc sample
OUT:  out_samples
PS C:\Users\User\Desktop\tw> python -W ignore .\test_generate_key.py -v
test_generate_params (__main__.TestGenerateKey) ... ok
test_generate_params_2 (__main__.TestGenerateKey) ... ok
test_generate_params_middle (__main__.TestGenerateKey) ... ok
test_generate_params_middle_2 (__main__.TestGenerateKey) ... ok

----------------------------------------------------------------------
Ran 4 tests in 6.707s

OK

PS C:\Users\User\Desktop\tw> python -W ignore .\test_beatmap.py -v
test_segment (__main__.TestBeatMap) ... ok
test_segment_2 (__main__.TestBeatMap) ... ok
test_segment_double_middle_3 (__main__.TestBeatMap) ... ok
test_segment_end (__main__.TestBeatMap) ... ok
test_segment_end_2 (__main__.TestBeatMap) ... ok
test_segment_middle_1 (__main__.TestBeatMap) ... ok
test_segment_middle_2 (__main__.TestBeatMap) ... ok
test_segment_middle_4 (__main__.TestBeatMap) ... ok

----------------------------------------------------------------------
Ran 8 tests in 5.161s

OK
```

Рисунок 3.33 – Тестування коректності модулів *beatmap* та *key_detection*

З результатів тестів можна зробити висновок, що модулі працюють коректно.

Варто також перевірити ефективність роботи даних методів. Для цього потрібно порівняти теоретичні значення параметрів аудіо з отриманими у результаті обчислень програми (рис. 3.34):

```
PS C:\Users\User\Desktop\tw> python .\test_key_accuracy.py
CD 1 - 02, Alejandro.wav        actual key      Bm      generated key    G / Bm
ImperialMarch.wav               actual key      Gm      generated key    Gm / D#
01, Livin' la vida loca.wav     actual key      C#m     generated key    C#m / C#
01, Lucky star.wav              actual key      Em      generated key    Dm / D
02, Toxic.wav                   actual key      Cm      generated key    Cm / C
04, Oops!....wav                actual key      C#m     generated key    C#m / G#m
05, Holiday.wav                 actual key      Gm      generated key    Gm / A#m
03, Burning up.wav              actual key      F#m     generated key    Bm / G
08, Everyb...wav                actual key      Am      generated key    Gm / Am
CD 2 - 03, Paparazzi.wav        actual key      C#      generated key    C# / C#m
CD 2 - 04, Poker face.wav       actual key      G#m     generated key    G#m / G#
CD 2 - 01, Just dance.wav       actual key      Em      generated key    Em / E
CD 2 - 02, Lovegame.wav         actual key      Bm      generated key    Bm / B
CD 1 - 06, Telephone.wav        actual key      Fm      generated key    Fm / F
CD 1 - 03, Monster.wav          actual key      Am      generated key    C / Am
05, Me against the music.wav    actual key      F#m     generated key    D / F#m
02, María ...wav                actual key      Gm      generated key    Gm / D#
10, Shes all I ever had.wav     actual key      B       generated key    B / E
12, Overprotected.wav           actual key      Cm      generated key    Gm / Fm
16, I love rock n roll.wav      actual key      A       generated key    F / Dm
Total correctness mark:  0.8
PS C:\Users\User\Desktop\tw> python .\test_beatmap_accuracy.py
CD 1 - 02, Alejandro.wav        actual BPM      99      generated BPM    99
ImperialMarch.wav               actual BPM      103     generated BPM    103
01, Livin' la vida loca.wav     actual BPM      178     generated BPM    89
01, Lucky star.wav              actual BPM      117     generated BPM    117
02, Toxic.wav                   actual BPM      143     generated BPM    143
04, Oops!....wav                actual BPM      95      generated BPM    95
05, Holiday.wav                 actual BPM      117     generated BPM    117
03, Burning up.wav              actual BPM      138     generated BPM    135
08, Everyb...wav                actual BPM      120     generated BPM    117
CD 2 - 03, Paparazzi.wav        actual BPM      115     generated BPM    117
CD 2 - 04, Poker face.wav       actual BPM      119     generated BPM    117
CD 2 - 01, Just dance.wav       actual BPM      119     generated BPM    117
CD 2 - 02, Lovegame.wav         actual BPM      105     generated BPM    103
CD 1 - 06, Telephone.wav        actual BPM      122     generated BPM    123
CD 1 - 03, Monster.wav          actual BPM      120     generated BPM    117
05, Me against the music.wav    actual BPM      120     generated BPM    117
02, María ...wav                actual BPM      126     generated BPM    129
10, Shes all I ever had.wav     actual BPM      81      generated BPM    80
12, Overprotected.wav           actual BPM      96      generated BPM    95
16, I love rock n roll.wav      actual BPM      180     generated BPM    89
Total correctness mark:  0.9
```

Рисунок 3.34 – Тестування точності модулів *beatmap* та *key_detection*

З результатів можна зробити висновок що модулі обчислень працюють досить точно – на 0.8 та 0.9 показники.

Також можна перевірити швидкість обчислень з різними вхідними даними та з параметром кешування.

Для цього перевірений час обробки даних різного розміру – від пари секунд до кількох хвилин для *process_mfcc* та *process_mel* (рис. 3.35):

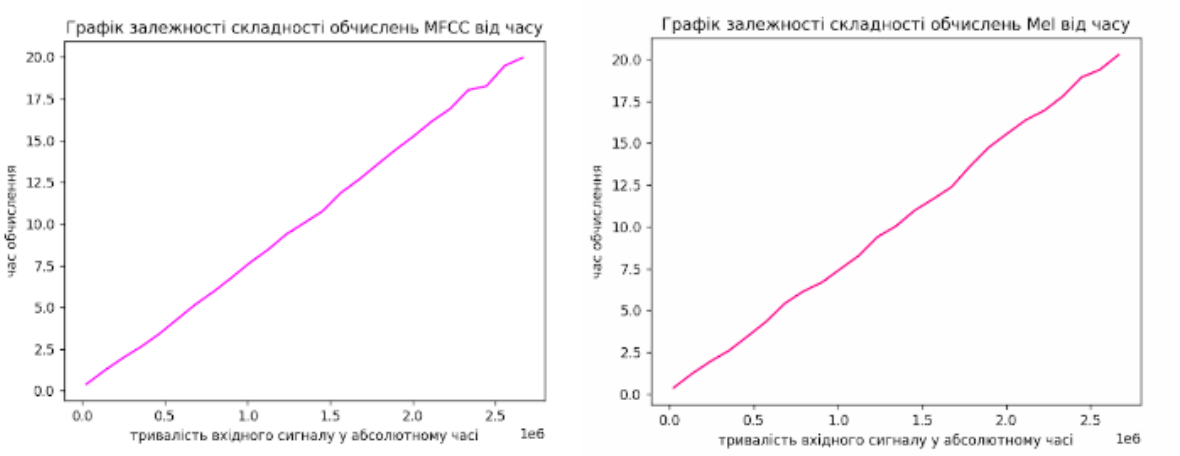


Рисунок 3.35 – Графік залежності часу обчислень від розміру вхідних даних

З графіку очевидна лінійна складність обчислень що складають елементи системи, причому аналіз твору тривалістю 100 секунд (1,5 хвилини) може займати 20 секунд реального часу.

У свою чергу отримання результату з кешу займає у середньому 400 мс, незалежно від кількості записів у Redis (рис. 3.36):

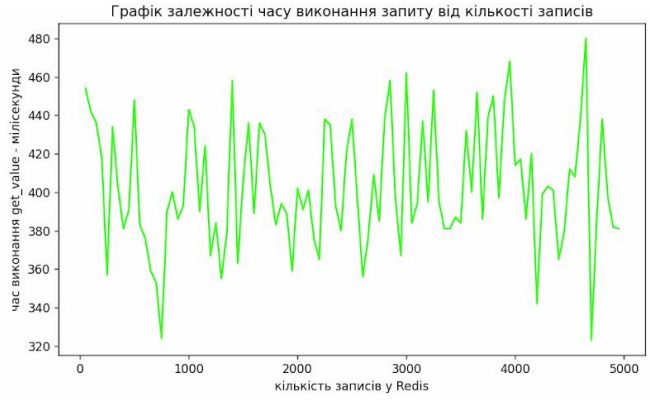


Рисунок 3.36 – Графік залежності часу запиту від кількості записів Redis

Можна зробити висновок що з точки зору часу кешування є оптимальним рішенням, що може значно пришвидшити роботу системи.

ВИСНОВОК ДО РОЗДІЛУ 3

У даному розділі було розглянуто реалізацію компонентів системи відповідно до вимог сервісу. Було розроблено модулі що виконують задачі:

- візуалізації аудіо за допомогою різних частотних та тональних представлень – *CoreTools*;
- обчисленні характеристик аудіо: тональності, BPM та ритму та сегментовано аудіо значення за значенням останньої – *GenerateKey, Beatmap*;
- збереження файлу з даними у нормалізованому вигляді на диску – *Extraction*;
- кешування результатів обчислень – *Caching*
- організації командного інтерфейсу – *CLI.*

Також було протестовано роботу сервісу та оцінено втілення поставлених задач. Було виявлено що деякі тональні представлення краще підходять для візуального аналізу аудіо – зокрема Мел спектрограма та результат STFT.

Було перевірено коректність аналізу даних та його зберігання. Також було перевірено механізм кешування результатів даних та протестовано різні компоненти системи, перевірено ефективність їх роботи.

Можна зробити висновок, що основні функціональні вимоги системи виконано.

ВИСНОВКИ

У ході виконання дипломного проєкту було досліджено алгоритми та технології аналізу аудіо та розроблено сервіс для обчислення параметрів аудіо даних музикального та статистичного роду. Даний сервіс надає можливість попередньої обробки даних для наступних етапів аналізу. Дані організовані у оптимальному форматі для подальшої оброки підсистемою ідентифікації акордів. При цьому визначаються характеристики аудіо що можуть бути корисними не тільки як частина мета даних, а мати практичне значення окремо.

У першому розділі було досліджено фундаментальні поняття музики та її складових для визначення ключових компонентів аудіо аналізу. Було розглянуто та порівняно системи аналізу музичних творів та їх використання у сервісах стримінгу та обробки аудіо. Внаслідок порівняння було виявлено що системи які оптимально задовольняють вимоги даного проекту, не мають API для можливості їх застосування та розширення іншими підсистемами.

У другому розділі було розглянуто технології аудіо аналізу та роботи з даними загалом. Було порівняно мови програмування та виокремлено ключові характеристики для оптимальної розробки системи. Також було досліджено алгоритм КШ для виокремлення тональних характеристик та порівняно технології отримання представлень аудіо та спектральних піків сигналу. Так як важливою частиною системи є зберігання даних, було оцінено практичність використання БД та прийнято рішення щодо аналогу, а саме зберігання файдів великого розміру у файловій системі.

Третій розділ описував деталі реалізації модулів системи та її окремих алгоритмів. Було реалізовано компоненти для вирішення задач аналізу та обчислення аудіо представлень та сформовано набір інструментів для візуалізації аудіо сигналу. Також було втілено механізм кешування результатів обчислень для покращення ефективності роботи системи. Також

було розглянуто результати роботи сервісу та виявлено що всі компоненти працюють коректно та результат є правильним.

У результаті розробки проекту було втілено усі ключові вимоги до сервісу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Thomas Christensen. History of Music Theory / Thomas Christensen.
2. Mark Andrew Cook. Music Theory / Mark Andrew Cook.. – 216 с.
3. Nicolas Carter. Music Theory: From Beginner to Expert - The Ultimate Step-By-Step Guide to Understanding and Learning / Nicolas Carter., 2018. – 220 с.
4. Samuel Chase. What Is Melody In Music? A Complete Guide [Електронний ресурс] / Samuel Chase – Режим доступу до ресурсу: https://hellomusictheory.com/learn/melody/.
5. Phillip Magnuson. A Structural Examination of Tonality, Vocabulary, Texture, Sonorities, and Time Organization in Western Art Music [Електронний ресурс] / Phillip Magnuson – Режим доступу до ресурсу: https://academic.udayton.edu/phillipmagnuson/soundpatterns/compbasics/5otherparts.html.
6. Spotify. Web API Guides [Електронний ресурс] / Spotify – Режим доступу до ресурсу: https://developer.spotify.com/documentation/web-api/guides/.
7. IBM Cloud. Python vs. R: What’s the Difference? [Електронний ресурс] / IBM Cloud. – 2021. – Режим доступу до ресурсу: https://www.ibm.com/cloud/blog/python-vs-r.
8. Таблица соотношения нот и частот [Електронний ресурс] – Режим доступу до ресурсу: https://nch-nch.ru/apps/frequency/.
9. BPM [Електронний ресурс] – Режим доступу до ресурсу: https://songbpm.com/
10. Moises [Електронний ресурс] – Режим доступу до ресурсу: https://moises.ai/.

11. How fast Numpy really is [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/how-fast-numpy-really-is-e9111df44347
12. Mcfree B. librosa: Audio and Music Signal Analysis in Python / Mcfree Brian, 2015. – 24 с.
13. Python String | split() [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://www.geeksforgeeks.org/python-string-split/
14. Python SHA256 Hashing Algorithm: Explained [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://datagy.io/python-sha256/
15. How To Use Redis With Python [Електронний ресурс] – Режим доступу до ресурсу: https://hackthedeveloper.com/use-redis-with-python/.
16. http://www.piano-keyboard-guide.com/wp-content/uploads/2018/12/Piano-Keys-and-Notes-Piano-Keyboard-Diagram-1-1024x576.jpg
17. Intro to Audio Analysis: Recognizing Sounds Using Machine Learning [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://hackernoon.com/intro-to-audio-analysis-recognizing-sounds-using-machine-learning-qy2r3ufl.
18. Harte C. Detecting Harmonic Change in Musical Audio. In Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia / Harte C., 2006.
19. pyAudioAnalysis wiki [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/tyiannak/pyAudioAnalysis/wiki.
20. Game Of Thrones Theme [Електронний ресурс] – Режим доступу до ресурсу: https://musescore.com/user/158751/scores/2163051.

21. librosa.feature.mfcc [Електронний ресурс] – Режим доступу до ресурсу: https://librosa.org/doc/main/generated/librosa.feature.mfcc.html.
22. Orlow M. Why Storing Files in the Database Is Considered Bad Practice [Електронний ресурс] / Maxim Orlow – Режим доступу до ресурсу: https://maximorlov.com/why-storing-files-database-bad-practice/.
23. What is Redis [Електронний ресурс] – Режим доступу до ресурсу: https://www.instaclustr.com/blog/what-is-redis/.
24. Madsen S. KEY-FINDING WITH INTERVAL PROFILES / Madsen Søren – Linz. – 4 с.

ДОДАТОК 1

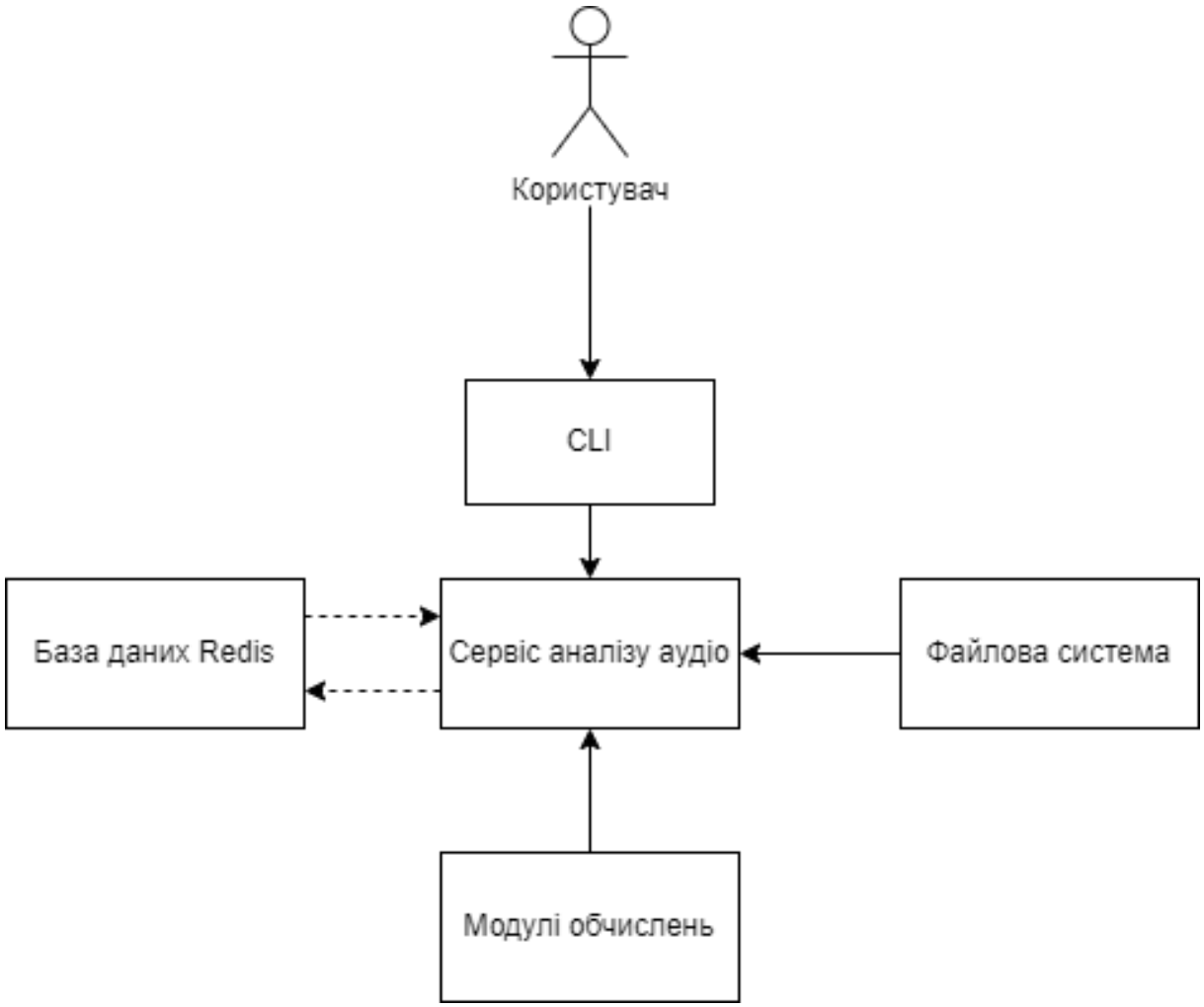
Музикальний сервіс для аналізу аудіо

Структурна діаграма системи

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.006 ДЗ		
		№ докум.	Підпис	Дата			
Розробила		Коломієць Є. В.			**Музикальний серівс для аналізу аудіо** Структурна діаграма системи	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.					
						НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-84	
Н. Контр.		Сімоненко В. П.					
Затвердив							

ДОДАТОК 2

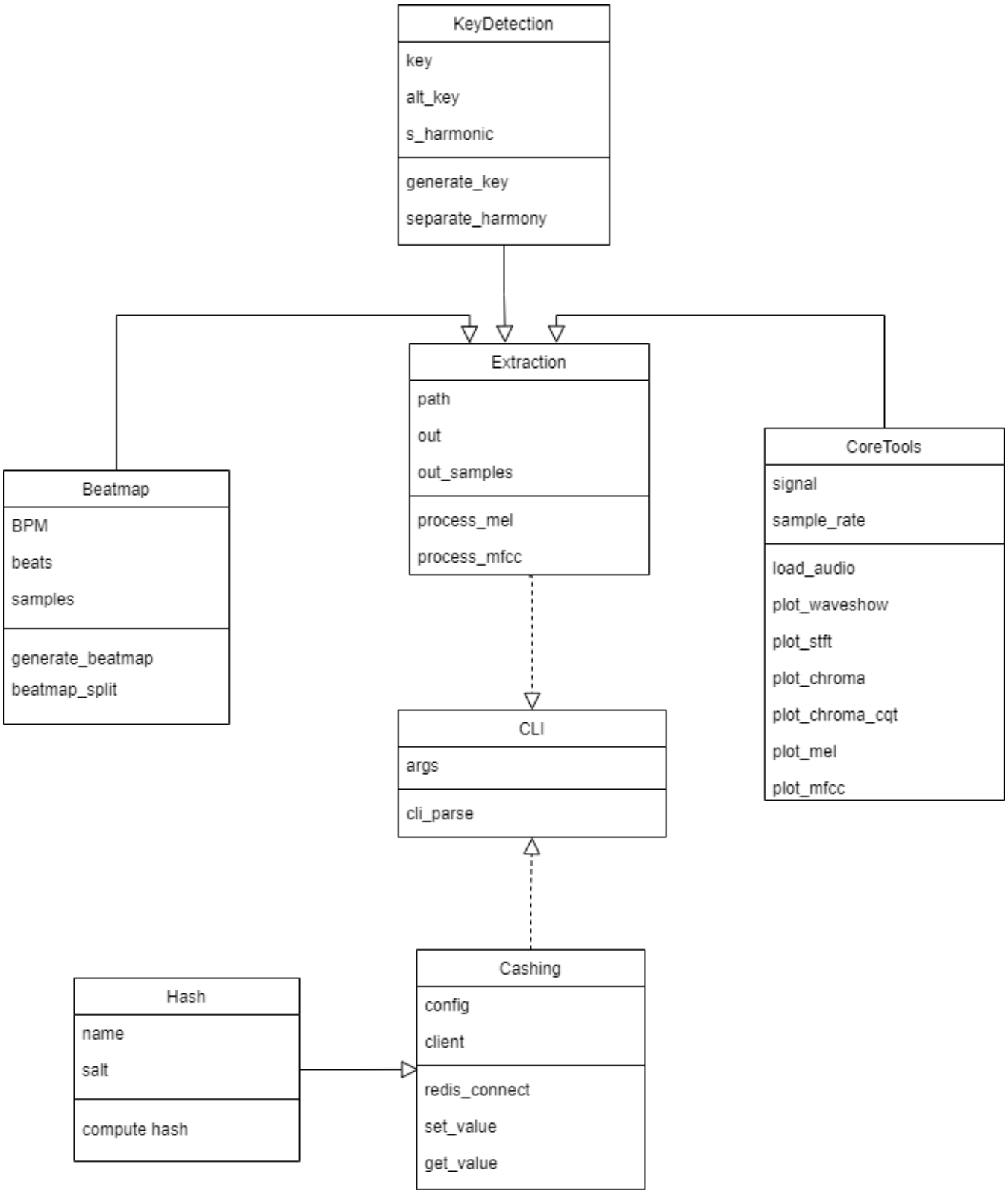
Музикальний сервіс для аналізу аудіо

Діаграма модулів

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробила		Коломієць Є.В..			**Музикальний сервіс для аналізу аудіо** Діаграма модулів	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.				
Н. Контр.		Сімоненко В. П.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-84
Затвердив						

ДОДАТОК 3

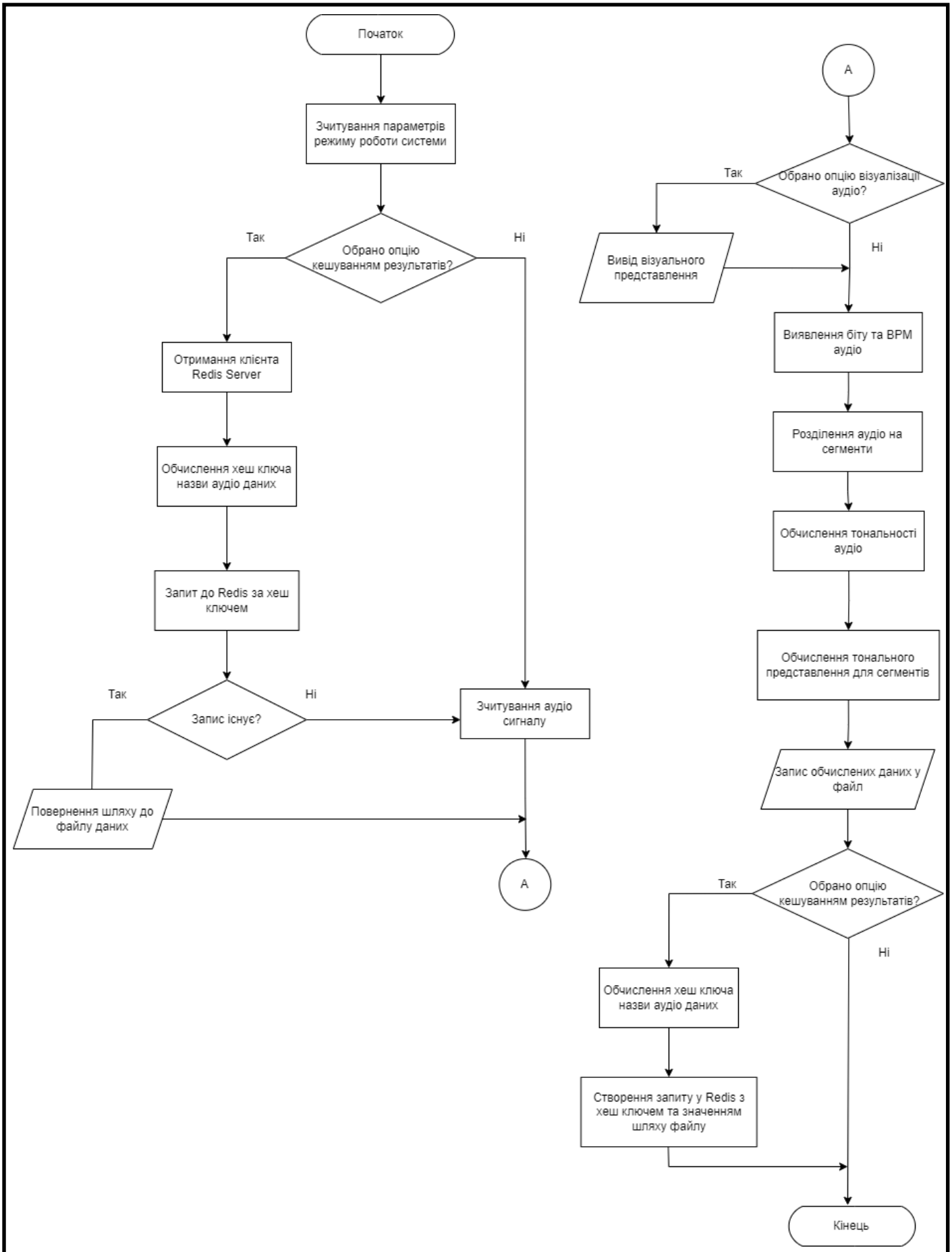
Музикальний сервіс для аналізу аудіо

Послідовність дій програмного забезпечення

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2022 р



					ІАЛЦ.467200.004 Д1	
		№ докум.	Підпис	Дата		
Розробила		Коломієць Є.В.			**Музикальний сервіс для аналізу аудіо** Послідовність дій програмного забезпечення	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Волокита А. М.				
Н. Контр.		Сімоненко В. П.				НТУУ КПІ ім. Ігоря Сікорського, ФІОТ, ІП-84
Затвердив						

ДОДАТОК 4

Музикальний сервіс для аналізу аудіо

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 29

Київ 2022 р

```
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np



# Function that acts as a wrapper to the loader function
# Takes the audio path, sample_rate (defaults to 22050)
# and offset in seconds which specifies whether the audio is
# read from the beginning.
#
# Returns a dictionary containing audio signal series and SR
# Use verbose = True for logging

def load_audio(audio_data, verbose = False, sample_rate = 22050, offset = 0.0):

    signal_info = {
        "sample_rate": sample_rate,
        "signal": []
    }

    if offset != 0.0:
        duration = librosa.get_duration(audio_data, sr = sample_rate)
        if offset >= duration:
            offset = 0.0

    x , sample_rate = librosa.load(audio_data, sr = sample_rate, offset = offset)
    signal_info['signal'] = x
```

```
        if verbose:
            print("INFO: Audio reading complete, data details: ", x.shape[0], sample_rate)

        return signal_info


    # Function that gives base visualization of signal
    # Takes the raw audio input and sample_rate (defaults to 22050)
    #
    # Shows a plot of audio signal to time

    def plot_waveshow(signal, sample_rate = 22050):
        librosa.display.waveshow(y = signal, sr = sample_rate)
        plt.figure(figsize = (15, 5))
        plt.show()


    # Function that gives a STFT visualization of signal
    # Takes the raw audio input and sample_rate (defaults to 22050)
    #
    # Shows a plot of audio frequency to time

    def plot_stft(signal, sample_rate = 22050):
        S = librosa.stft(signal)
        Sdb = librosa.amplitude_to_db(abs(S))
        plt.figure(figsize=(15, 5))
        librosa.display.specshow(Sdb, sr = sample_rate, x_axis='time', y_axis='log')
        plt.colorbar()
        plt.show()
```

```
    # Function that gives a chromagram visualization of signal
    # Takes the raw audio input and sample_rate (defaults to 22050)
    #
    # Shows a plot of chroma features to time

    def plot_chroma(signal, sample_rate = 22050):
        chroma = librosa.feature.chroma_stft(signal, sr = sample_rate)
        plt.figure(figsize=(15, 5))
        librosa.display.specshow(chroma, x_axis = 'time', y_axis = 'chroma', cmap =
'coolwarm')
        plt.show()


    # Function that gives a variation of chromagram visualization of signal
    # Takes the raw audio input and sample_rate (defaults to 22050)
    #
    # Shows a plot of chroma features to time

    def plot_chroma_cqt(signal,  bins = 24, sample_rate = 22050):

        chroma = librosa.feature.chroma_cqt(y = signal, sr = sample_rate,
bins_per_octave=bins)
        librosa.display.specshow(chroma, sr = sample_rate, y_axis = 'chroma',
x_axis= 'time')
        plt.show()

    def plot_mel(signal, sample_rate = 22050):
        S = librosa.feature.melspectrogram(y=signal, sr=sample_rate, n_mels=128,
fmax=8000)
        fig, ax = plt.subplots()
        S_dB = librosa.power_to_db(S, ref = np.max)
```

```
        img = librosa.display.specshow(S_dB, x_axis='time', y_axis='mel',
sr=sample_rate, fmax=8000, ax=ax)
        fig.colorbar(img, ax=ax, format='%+2.0f dB')
        ax.set(title='Mel spectrogram')
        plt.show()

    def plot_mfcc(signal, sample_rate = 22050):
        mfccs = librosa.feature.mfcc(y = signal, sr=sample_rate)
        librosa.display.specshow(mfccs, sr = sample_rate, x_axis='time')
        plt.title('MFCC Spectrogram')
        plt.show()

    from random import sample
    from signal import signal
    import numpy as np
    import matplotlib.pyplot as plt
    import librosa
    import librosa.display
    from core_tools import load_audio

    CHORDS = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']

    KRUM_MAJ = [6.35, 2.23, 3.48, 2.33, 4.38, 4.09, 2.52, 5.19, 2.39, 3.66, 2.29,
2.88]
    KRUM_MIN = [6.33, 2.68, 3.52, 5.38, 2.60, 3.53, 2.54, 4.75, 3.98, 2.69, 3.34,
3.17]



    def separate_harmony(signal):
        s_harmonic, s_percussive = librosa.effects.hpss(y = signal)
        return s_harmonic
```

```
    def generate_key(signal, sample_rate, start_sample = 0, end_sample = 0, bins =
24):
        signal = separate_harmony(signal)

        if (end_sample == 0) or (end_sample < sample_rate/100):
            end_sample = len(signal)

        chroma = librosa.feature.chroma_cqt(y = signal[start_sample: end_sample],
sr=sample_rate, bins_per_octave=bins)
        pitch = 12

        chroma_sum = []
        for i in range(pitch):
            chroma_sum.append(np.sum(chroma[i]))

        freqs = {}
        for i in range(pitch):
            freqs[CHORDS[i]] = chroma_sum[i]



        cors_min = []
        cors_maj = []

        for i in range(pitch):
            key = []
            for j in range(pitch):
                c = CHORDS[(i + j)%12]
                key.append(freqs[c])
            corr_maj = np.corrcoef(KRUM_MAJ, key)[0, 1]
            corr_min = np.corrcoef(KRUM_MIN, key)[0, 1]
```

```
        cors_maj.append(corr_maj)
        cors_min.append(corr_min)

    key_set = CHORDS

    for i in range(pitch):
        key_set.append(key_set[i] + 'm')

    corr_set = {}
    for i in range(pitch):
        corr_set[key_set[i]] = cors_maj[i]

    for i in range(pitch, pitch*2):
        corr_set[key_set[i]] = cors_min[i-12]

    res_key = max(corr_set, key=corr_set.get)
    best_value = corr_set.pop(res_key)

    alt_key = max(corr_set, key=corr_set.get)
    return {
        "key": res_key,
        "alt_key": alt_key
     }



# # Usage

# audio_path = './samples/CD 1 - 02, Alejandro.wav'
# signal_info = load_audio(audio_path)
# signal = signal_info["signal"]
# sample_rate = signal_info["sample_rate"]
```

```
# keys= generate_key(signal, sample_rate, end_sample = 20 * sample_rate)


from email.mime import audio
from signal import signal
import sys
from turtle import shape
import librosa, librosa.display
import matplotlib.pyplot as plt
import numpy as np
import os
import json
import beatmap
from core_tools import load_audio, plot_mel, plot_mfcc
from key_detection import generate_key
import split



CHORDS = ['C', 'D', 'E', 'F', 'G', 'A', 'B', 'Cm', 'Dm', 'Em', 'Fm', 'Gm', 'Am', 'Bm',
'Cmaj7', 'Dmaj7', 'Emaj7', 'Fmaj7', 'Gmaj7', 'Amaj7', 'Bmaj7', 'Cmin7', 'Dmin7',
'Emin7', 'Fmin7', 'Gmin7', 'Amin7', 'Bmin7']

audio_data = "./samples/CD 1 - 02, Alejandro.wav"



def process_mel(path, out, enableVisual = False):
    signal_info = load_audio(path)
    if enableVisual:
        plot_mel(signal_info["signal"], signal_info["sample_rate"])
```

```
        samples = split. beatmap_split(signal_info["signal"], 512, sample_rate=
signal_info["sample_rate"])
        key_set = generate_key(signal_info["signal"], signal_info["sample_rate"])

        name = os.path.split(path)[-1][:-4]
        out_sample = {
            "name": name,
            "BPM": samples["BPM"],
            "key": key_set["key"],
            "alt_key": key_set["alt_key"],
            "frame_size": samples["frame_size"],
            "sample_rate": samples["sample_rate"],
            "melograms": []
        }
        counter = 0

        for sample in samples["samples"]:
            header = {
                "id": counter,
                "frame_start": int(sample["frame_start"]),
                "frame_end": int(sample["frame_end"])
            }

            mel = librosa.feature.melspectrogram(y = sample["signal"], sr =
samples["sample_rate"], hop_length = samples["frame_size"])
            mel = mel.T
            out_sample["melograms"].append({
                "header": header,
                "melogram": mel.tolist()
                }
            )
            counter += 1
```

```
    with open(out, "w") as f:
        json.dump(out_sample, f, indent=4)


def process_mfcc(path, out, n_mfcc = 13, enableVisual= False):
    signal_info = load_audio(path)
    if enableVisual:
        plot_mfcc(signal_info["signal"], signal_info["sample_rate"])


    samples = split. beatmap_split(signal_info["signal"], 512, sample_rate= signal_info["sample_rate"])
    key_set = generate_key(signal_info["signal"], signal_info["sample_rate"])

    name = os.path.split(path)[-1][:-4]
    out_sample = {
        "name": name,
        "BPM": samples["BPM"],
        "key": key_set["key"],
        "alt_key": key_set["alt_key"],
        "frame_size": samples["frame_size"],
        "sample_rate": samples["sample_rate"],
        "mfccs": []
    }
    counter = 0

    for sample in samples["samples"]:
        header = {
            "id": counter,
            "frame_start": int(sample["frame_start"]),
```

```
            "frame_end": int(sample["frame_end"])
        }

        mfcc = librosa.feature.mfcc(y = sample["signal"], sr = samples["sample_rate"], hop_length = samples["frame_size"], n_mfcc= n_mfcc)
        mfcc = mfcc.T
        out_sample["mfccs"].append({
            "header": header,
            "mfcc": mfcc.tolist()
            }
        )
        counter += 1


    with open(out, "w") as f:
        json.dump(out_sample, f, indent=4)

import librosa, librosa.display
import os
import matplotlib.pyplot as plt
import numpy as np
from core_tools import load_audio

from matplotlib.pyplot import plot

# Function that takes raw input signal and it`s sample rate (optional)`
# and returns a dictionary with BPM value and beats array
#
# Set enableVisual to True if visualization is neccessary

def generate_beatmap(signal, sample_rate = 22050, enableVisual = False):
    onset_frames = librosa.onset.onset_detect(y = signal, sr = sample_rate)
```

```
        bpm, beats = librosa.beat.beat_track(y = signal, sr=sample_rate)

        beatmap_info = {
          "BPM": bpm,
          "beats": beats
        }

        if enableVisual:
            # Displaying melspectrogram

            hop_length = 512
            fig, ax = plt.subplots(nrows=2, sharex=True)
            onset_env = librosa.onset.onset_strength(y=signal, sr=sample_rate,
aggregate=np.median)
            times = librosa.times_like(onset_env, sr=sample_rate,
hop_length=hop_length)
            M = librosa.feature.melspectrogram(y=signal, sr=sample_rate,
hop_length=hop_length)

            # Displaying beats

            librosa.display.specshow(librosa.power_to_db(M, ref=np.max),
y_axis='mel', x_axis='time', hop_length=hop_length, ax=ax[0])
            ax[0].label_outer()
            ax[0].set(title='Mel spectrogram')
            ax[1].plot(times, librosa.util.normalize(onset_env), label='Onset strength')
            ax[1].vlines(times[beats], 0, 1, alpha=0.5, color='r', linestyle='--',
label='Beats')
            ax[1].legend()
            plt.show()
```

```
        return beatmap_info



    # # Usage

    # audio_path = './samples/CD 1 - 02, Alejandro.wav'
    # signal_info = load_audio(audio_path)
    # signal = signal_info["signal"]
    # sample_rate = signal_info["sample_rate"]

    # bit_info = generate_beatmap(signal)

    from email.mime import audio
    import librosa, librosa.display
    import matplotlib.pyplot as plt
    import numpy as np
    import os
    import beatmap
    from core_tools import load_audio

    # Function that takes input signal and it`s sample rate (optional)`
    # and it`s beat array computed by beatmap module
    # and returns a dictionary with samples split by passed beats
    # and general info
    # Hop_size should be specified by number of signals per frame
    #
    # Set enableVerbose to True if logging is neccessary

    def beatmap_split(signal, hop_size, sample_rate = 22050, enableVerbose =
False):
        beat_info = beatmap.generate_beatmap(signal)
```

```
        track_info = {
            "BPM": beat_info["BPM"],
            "frame_size": hop_size,
            "sample_rate": sample_rate,
            "samples": []
        }

        beats = np.append(beat_info["beats"], 0)
        i = 0
        while i < len(beats) - 1:
            start = beats[i - 1]
            end =  beats[i]

            sample = {
                "signal": [],
                "frame_start": start,
                "frame_end": end,
            }
            sample["signal"] = signal[start * hop_size : end * hop_size]

            track_info["samples"].append(sample)
            i += 1
        if enableVerbose:
            print("INFO: splitting into beat samples, total ",
len(track_info["samples"]))
        return track_info


    # # Usage

    # signal_info = load_audio('./samples/CD 1 - 02, Alejandro.wav')
```

```
    # beatmap_split(signal_info["signal"], 512, signal_info["sample_rate"],
enableVerbose = True)

    import re
    import hashlib

    def compute_hash(name, salt):
        items = re.split(' +|; +|, +|\n|_+', name)
        items = [item.lower() for item in items]
        items.sort()
        raw_name = ''.join(items)
        hashed = hashlib.sha256((salt + raw_name).encode('utf-8')).hexdigest()

        return hashed

    import hash
    import redis

    config = {
        "host": '127.0.0.1',
        "port": 6379,
        "salt": ''
    }

    def redis_connect(config):
        client = {
            "config": config,
            "redis": False,
        }

        rd = redis.Redis(host = config["host"], port = config["port"])
```

```
    if rd.ping():
        client['redis'] = rd
        return client
    return False


def set_value(name, path, client, enableVerbose = False):
    key = hash.compute_hash(name, client["config"]["salt"])
    redis = client["redis"]
    set = redis.set(key, path)
    if enableVerbose:
        print("INFO: Caching result: ", path)
    return set

def get_value(name, client, enableVerbose = False):
    key = hash.compute_hash(name, client["config"]["salt"])
    redis = client["redis"]
    value = redis.get(key)
    if enableVerbose:
        print("INFO: Accessing cashed result: ", value)
    return value



# cl = redis_connect(config)
# set_value("Lady Gaga Telephone", "/d/d", cl, True)
# get_value("Lady Gaga Telephone", cl, True)

# get_value("lady__gaga Telephone", cl, True)
import argparse
import os
import sys
```

```
    # Function for parsing cli arguments
    def cli_parse():
        parser = argparse.ArgumentParser(description='Audio service')

        # Add the arguments
        parser.add_argument('path', metavar='path', type=str, help='the path to audio or audio directory')
        parser.add_argument('output',action='store', nargs='?', metavar='out', type=str, help='the path to results', default='/out')

        parser.add_argument('-d', '--directory', action='store_true', help='enable analyzing directory content')
        parser.add_argument('-v', action='store_true', help='enable logging')
        parser.add_argument('-c', '--caching', action='store_true', help='enable caching in redis database')
        parser.add_argument('-mfcc', action = 'store_true', help = 'use mfcc analysis. System uses melogram by default')

        args = parser.parse_args()
        return args

    from pytube import YouTube
    import ffmpeg
    import argparse

    parser = argparse.ArgumentParser(description='Audio analysing service')
    parser.add_argument('url', metavar='audio_url', type=str, help='the path to audio or audio directory')
    parser.add_argument('output', metavar='out', type=str, help='the name of results')
```

```
args = parser.parse_args()

# audio_url = 'https://www.youtube.com/watch?v=vJyy1yUsPvk'
# name = 'Lykke Li - a Little bit.wav'

def extract_audio(audio_url, name):
    yt = YouTube(audio_url)
    stream_url = yt.streams.all()[0].url  # Get the URL of the video stream

    audio, err = (
        ffmpeg
        .input(stream_url)
         .output("pipe:", format='wav', acodec='pcm_s16le')  # Select WAV output format, and pcm_s16le auidio codec
        .run(capture_stdout=True)
    )

    # Write the audio buffer to file for testing
    with open(name, 'wb') as f:
        f.write(audio)

extract_audio(args.url, args.output)
import os
import re
from cashing import set_value, get_value, redis_connect
from pre import process_mel, process_mfcc

def analyse(name, out = 'out_samples', verbose = False, visual = False, enable_cashing = False, mel = False):
    cashed = False
    client = ''
    file_name = name
```

```
        song_name = os.path.split(file_name)[-1][:-4]
        res_name = out + '/' + song_name

        if enable_cashing:
            client = redis_connect()
            res = get_value(song_name, client, verbose)
            if res != None:
                if verbose:
                    print ("INFO: Successfully getting a stored value")
                cashed = True
        if not cashed:
            if mel:
                process_mel(name, out, visual)
                if verbose:
                    print ("INFO: Successfully creating a mel sample")
            else:
                process_mfcc(name, out, enableVisual = visual)
                if verbose:
                    print ("INFO: Successfully creating a mfcc sample")
            if enable_cashing:
                res = set_value(song_name, res_name, client, verbose)
                if res != False:
                    if verbose:
                        print ("INFO: Successfully storing a value")

        print ("OUT: ", out)

    # Usage
    # analyse(name= './samples/agl.wav', out = 'out_mfcc', visual= True, verbose =
True, mel=True, enable_cashing= True)
```

```
def main():
    args = cli_parse()
    if args.v:
        print("INFO: Initializing analyser")
    analyse(args.path, args.output, args.v, args.visual, args.caching, args.mel)

if __name__ == "__main__":
    main()



from time import perf_counter, perf_counter_ns

from matplotlib import pyplot as plt
from cashing import get_value, redis_connect, set_value
from core_tools import load_audio

client = redis_connect()
print(redis_connect)
value_count = [x*50 for x in range(1, 100)]

times = []

for values in value_count:
    value_margin = 50
    for i in range(value_margin):
        set_value((str(i) + "key"), (str(i) + "value"), client)

    start_time = perf_counter_ns()

    v = get_value(str(int(values/2)), client)

    print("processing for ", values)
```

```
    end_time = perf_counter_ns()

    times.append(int((end_time - start_time)/1000))


plt.plot(value_count, times, color = '#39ff14')
plt.xlabel('кількість записів у Redis')
plt.ylabel('час виконання get_value - мілісекунди')
plt.title('Графік залежності часу виконання запиту від кількості записів')
plt.show()


import os
from time import perf_counter
from key_detection import generate_key
from matplotlib import pyplot as plt
from core_tools import load_audio
from split import beatmap_split
import librosa

audio_data = './samples/CD 1 - 03, Monster.wav'

signal_info = load_audio(audio_data)
signal = signal_info['signal']
sample_rate = signal_info['sample_rate']

max_time = int(len(signal)/sample_rate)
intervals = [x*sample_rate for x in range(1, int(max_time/2), 5)]

times = []

for sample_interval in intervals:
```

```
        start_time = perf_counter()

            samples = beatmap_split(signal[:sample_interval], 512, sample_rate=
signal_info["sample_rate"])
        key_set = generate_key(signal[:sample_interval], signal_info["sample_rate"])

        name = os.path.split(audio_data)[-1][:-4]
        out_sample = {
            "name": name,
            "BPM": samples["BPM"],
            "key": key_set["key"],
            "alt_key": key_set["alt_key"],
            "frame_size": samples["frame_size"],
            "sample_rate": samples["sample_rate"],
            "mfccs": []
        }
        counter = 0

        for sample in samples["samples"]:
            header = {
                "id": counter,
                "frame_start": int(sample["frame_start"]),
                "frame_end": int(sample["frame_end"])
            }

                    mfcc = librosa.feature.mfcc(y = sample["signal"], sr =
samples["sample_rate"], hop_length = samples["frame_size"], n_mfcc= 13)
            mfcc = mfcc.T
            out_sample["mfccs"].append({
                "header": header,
                "mfcc": mfcc.tolist()
```

```
            }
        )
        counter += 1

    print("processing for ", sample_interval)
    end_time = perf_counter()
    times.append(end_time - start_time)



plt.plot(intervals, times, color = '#ff00ff')
plt.xlabel('тривалість вхідного сигналу у абсолютному часі')
plt.ylabel('час обчислення')
plt.title('Графік залежності складності обчислень MFCC від часу')
plt.show()

from beatmap import generate_beatmap
from core_tools import load_audio

samples = {
    'CD 1 - 02, Alejandro.wav':         99,
    'ImperialMarch.wav':                103,
    '01, Livin\' la vida loca.wav' :    178,
    '01, Lucky star.wav':               117,
    '02, Toxic.wav':                    143,
    '04, Oops!...I did it again.wav':   95,
    '05, Holiday.wav':                  117,
    '03, Burning up.wav':               138,
    '08, Everybody (original version).wav': 120,
    'CD 2 - 03, Paparazzi.wav':         115,
    'CD 2 - 04, Poker face.wav':        119,
    'CD 2 - 01, Just dance.wav':        119,
    'CD 2 - 02, Lovegame.wav':          105,
```

```
    'CD 1 - 06, Telephone.wav':         122,
    'CD 1 - 03, Monster.wav':           120,
    '05, Me against the music.wav':     120,
    '02, María (Pablo Flores Spanglish radio edit).wav':    126,
    '10, Shes all I ever had.wav':      81,
    '12, Overprotected.wav':            96,
    '16, I love rock n roll.wav':       180
}

fit = 0
accept = 0.03

for sample, expected in samples.items():
    signal_info = load_audio('./samples/' + sample)
    signal = signal_info['signal']
    sample_length = int(len(signal)/10)
    signal = signal[sample_length:sample_length*6]
    sample_rate = signal_info['sample_rate']
    beat_info = generate_beatmap(signal, sample_rate)
    bpm = int(beat_info['BPM'])

    if len(sample) < 14:
        print(sample, '\t\t\tactual BPM\t', expected, '\tgenerated BPM\t', bpm)
    elif len(sample) < 23:
        print(sample, '\t\tactual BPM\t', expected, '\tgenerated BPM\t', bpm)
    elif len(sample) > 28:
        sample = sample[:10] + '...wav'
        print(sample, '\t\tactual BPM\t', expected, '\tgenerated BPM\t', bpm)
    else:
        print(sample, '\tactual BPM\t', expected, '\tgenerated BPM\t', bpm)

    margin = int(bpm * accept)
```

```
    if expected - margin <= bpm <= expected + margin:
        fit += 1
    elif (expected - margin)*2 <= bpm <= (expected + margin)*2:
        fit += 1



mark = fit/len(samples.keys())
print('\nTotal correctness mark: ', mark)



import unittest
from split import beatmap_split
from core_tools import load_audio

track = load_audio('./samples/CD 1 - 02, Alejandro.wav')
audio_signal = track['signal']
track_2 = load_audio('./samples/ImperialMarch.wav')
audio_signal_2 = track_2['signal']

test_length = int(len(audio_signal)/30)
test_length_2 = int(len(audio_signal_2)/10)

class TestBeatMap(unittest.TestCase):

    def test_segment(self):
        track_info = beatmap_split(signal=audio_signal[:test_length])
        extracted = len(track_info['samples'])


        self.assertEqual(extracted, 14, "Should be 14 beats")

    def test_segment_middle_1(self):
```

```
                track_info = beatmap_split(signal=audio_signal[test_length*3:test_length*4])
        extracted = len(track_info['samples'])
        self.assertEqual(extracted, 13, "Middle - Should be 14 with +- 1 variance")

    def test_segment_middle_2(self):
                track_info = beatmap_split(signal=audio_signal[test_length*10:test_length*11])
        extracted = len(track_info['samples'])
        self.assertEqual(extracted, 14, "Middle - Should be 14 with +- 1 variance")

    def test_segment_double_middle_3(self):
                track_info = beatmap_split(signal=audio_signal[test_length*15:test_length*17])
        extracted = len(track_info['samples'])
        self.assertEqual(extracted, 28, "Middle - Should be 28 with +- 2 variance")

    def test_segment_end(self):
        track_info = beatmap_split(signal=audio_signal[test_length*29:])
        extracted = len(track_info['samples'])
        less = extracted < 14
        self.assertTrue(less, "Should be less than 14")

    def test_segment_2(self):
            track_info = beatmap_split(signal=audio_signal_2[:test_length_2], enableVisual=True)
        extracted = len(track_info['samples'])

        self.assertEqual(extracted, 9, "Should be 9 beats")

    def test_segment_middle_4(self):
```

```
                                                    track_info          =
beatmap_split(signal=audio_signal_2[test_length_2*3:test_length_2*4])
            extracted = len(track_info['samples'])
            self.assertEqual(extracted, 10, "Middle - Should be 9 with +- 1 variance")

        def test_segment_end_2(self):
            track_info = beatmap_split(signal=audio_signal_2[test_length_2*9:])
            extracted = len(track_info['samples'])
            less = extracted < 5*10
            self.assertTrue(less, "Should be around no longer than half audio")



    if __name__ == '__main__':
        unittest.main()

    import unittest
    from key_detection import generate_key
    from core_tools import load_audio

    track = load_audio('./samples/CD 1 - 02, Alejandro.wav')
    audio_signal = track['signal']
    sample_rate = track['sample_rate']
    track_2 = load_audio('./samples/ImperialMarch.wav')
    audio_signal_2 = track_2['signal']

    test_length = int(len(audio_signal)/30)
    test_length_2 = int(len(audio_signal_2)/10)

    class TestGenerateKey(unittest.TestCase):

        def test_generate_params(self):
```

```
            keys = generate_key(signal=audio_signal[:test_length], sample_rate = sample_rate)
            empty = (keys['key'] == '') or (keys ['alt_key'] == '')

            self.assertFalse(empty, "Keys should not be empty")
            self.assertIn(len(keys['key']), [1, 2], "Keys should have a 1-2 character length")
            self.assertIn(len(keys['alt_key']), [1, 2], "Keys should have a 1-2 character length")

        def test_generate_params_middle(self):
            keys = generate_key(signal=audio_signal[test_length*10:test_length*12], sample_rate = sample_rate)
            empty = (keys['key'] == '') or (keys ['alt_key'] == '')

            self.assertFalse(empty, "Keys should not be empty")
            self.assertIn(len(keys['key']), [1, 2], "Keys should have a 1-2 character length")
            self.assertIn(len(keys['alt_key']), [1, 2], "Keys should have a 1-2 character length")

        def test_generate_params_2(self):
            keys = generate_key(signal=audio_signal_2[:test_length_2], sample_rate = sample_rate)
            empty = (keys['key'] == '') or (keys ['alt_key'] == '')

            self.assertFalse(empty, "Keys should not be empty")
            self.assertIn(len(keys['key']), [1, 2], "Keys should have a 1-2 character length")
            self.assertIn(len(keys['alt_key']), [1, 2], "Keys should have a 1-2 character length")
```

```
        def test_generate_params_middle_2(self):
                                                                keys = generate_key(signal=audio_signal[test_length_2*5:test_length_2*7], sample_rate = sample_rate)
            empty = (keys['key'] == '') or (keys ['alt_key'] == '')

            self.assertFalse(empty, "Keys should not be empty")
              self.assertIn(len(keys['key']), [1, 2], "Keys should have a 1-2 character length")
             self.assertIn(len(keys['alt_key']), [1, 2], "Keys should have a 1-2 character length")



    if __name__ == '__main__':
        unittest.main()



    from key_detection import generate_key
    from core_tools import load_audio

    samples = {
        'CD 1 - 02, Alejandro.wav':        ['Bm', 'G'],
        'ImperialMarch.wav':               ['Gm', ''],
        '01, Livin\' la vida loca.wav' :   ['C#m', ''],
        '01, Lucky star.wav':              ['Em', 'G'],
        '02, Toxic.wav':                   ['Cm', 'Ebm'],
        '04, Oops!...I did it again.wav':  ['C#m', ''],
        '05, Holiday.wav':                 ['Gm', 'Bm'],
        '03, Burning up.wav':              ['F#m', ''],
        '08, Everybody (original version).wav': ['Am', ''],
        'CD 2 - 03, Paparazzi.wav':        ['C#', 'G#'],
        'CD 2 - 04, Poker face.wav':       ['G#m', ''],
```

```
        'CD 2 - 01, Just dance.wav':       ['Em', ''],
        'CD 2 - 02, Lovegame.wav':         ['Bm', ''],
        'CD 1 - 06, Telephone.wav':        ['Fm', ''],
        'CD 1 - 03, Monster.wav':          ['Am', ''],
        '05, Me against the music.wav':    ['F#m', 'Bm'],
        '02, María (Pablo Flores Spanglish radio edit).wav':   ['Gm', ''],
        '10, Shes all I ever had.wav':     ['B', ''],
        '12, Overprotected.wav':           ['Cm', ''],
        '16, I love rock n roll.wav':      ['A', ''],
    }

    fit = 0

    for sample, expected in samples.items():
        signal_info = load_audio('./samples/' + sample)
        signal = signal_info['signal']
        sample_length = int(len(signal)/10)
        signal = signal[sample_length:sample_length*6]
        sample_rate = signal_info['sample_rate']
        key_set = generate_key(signal, sample_rate)

        if len(sample) < 14:
                print(sample, '\t\t\tactual key\t', expected[0], '\tgenerated key\t', key_set['key'], '/', key_set['alt_key'])
        elif len(sample) < 23:
                print(sample, '\t\tactual key\t', expected[0], '\tgenerated key\t', key_set['key'], '/', key_set['alt_key'])
        elif len(sample) > 28:
            sample = sample[:10] + '...wav'
                print(sample, '\t\tactual key\t', expected[0], '\tgenerated key\t', key_set['key'], '/', key_set['alt_key'])
        else:
```

```
            print(sample, '\tactual key\t', expected[0], '\tgenerated key\t', key_set['key'],
'/', key_set['alt_key'])

            if key_set['key'] in expected or key_set['alt_key'] in expected:
                fit += 1



        mark = fit/len(samples.keys())
        print('Total correctness mark: ', mark)
```