Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Розробка телеграм-бота для обробки музичного контенту з використанням штучного інтелекту

Виконала: студентка __4___ курсу
групи_4ПІ-20б___
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Шлапак А.В.
(прізвище та ініціали)
Керівник: доц. каф. ПЗ Черноволик Г.О.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф.ЗІ Маліновський В.І.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри ПЗ _____Романюк О. Н.
«_10_» __червня_____ 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 12 » березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Шлапак Аллі Василівні

1. Тема роботи – розробка телеграм-бота для обробки музичного контенту з використанням штучного інтелекту.

Керівник роботи: Черноволик Галина Олександрівна, доцент кафедри програмного забезпечення, затверджені наказом вищого навчального закладу від 11 березня 2024 р. № 80.

2. Строк подання студентом роботи 3.червня 2024.

3. Вихідні дані до роботи: середовище розробки – PyCharm, мова розробки – Python, операційна система – Windows.

4. Зміст текстової частини: вступ; обґрунтування вибору методу розробки та постановка задач дослідження; розробка моделі та алгоритмів програмного застосунку; розробка програмних модулів застосунку; тестування програми; висновки; література; додатки; графічна частина.

5. Перелік ілюстративного матеріалу: титульний слайд; актуальність теми; мета, об'єкт і предмет дослідження; новизна отриманих результатів; порівняльний аналіз аналогів; розробка алгоритмів роботи системи; тестування програми; висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Черноволик Г.О., к.т.н., доцент кафедри ПЗ	13.03.2024	03.06.2024

7. Дата видачі завдання 13. 03.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання та обґрунтування завдання	14.03.2024 – 29.03.2024	вик.
2	Аналіз вхідних та вихідних даних системи	30.03.2024 – 07.04.2024	вик.
3	Розробка моделі та алгоритмів системи	08.04.2024 – 26.04.2024	вик.
4	Розробка інтерфейсу	27.04.2024 – 04.05.2024	вик.
5	Розробка програмних модулів системи	05.05.2024 – 14.05.2024	вик.
6	Тестування системи	15.05.2024– 20.05.2024	вик.
7	Оформлення матеріалів до захисту БДР	21.05.2024 – 30.05.2024	вик.

Студентка _____ **А.В.Шлапак**
(підпис) (ініціали та прізвище)

Керівник бакалаврської дипломної роботи _____ **Г.О.Черноволик**
(підпис) (ініціали та прізвище)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 85 сторінок формату А4, на яких є 30 рисунків, 3 таблиць, список використаних джерел містить 21 найменування.

У бакалаврській дипломній роботі проведено аналіз стану програмних застосунків для обробки музичного контенту з використанням штучного інтелекту та постановка задач дослідження. Було проведено порівняльний аналіз аналогів програмного застосунку, визначено їх переваги та недоліки та доведено доцільність розробки програмного застосунку.

Під час виконання роботи було створено програмний застосунок, що дозволяє надавати оброблені аудіофайли, розділені на вокал та інструментал.

Застосунок розроблено з використанням мови програмування Python та середовища розробки програмного забезпечення PyCharm. Було використано засоби розробленого штучного інтелекту Spleeter.

Отримані в бакалаврській дипломній роботі результати можуть бути використані музикантами та зацікавленими в музиці користувачами для обробки музичних файлів.

Ключові слова: телеграм-бот, штучний інтелект, музика, музиканти.

ABSTRACT

The bachelor's thesis consists of 85 pages of A4 format, which includes 30 figures, 3 tables, and a list of references containing 21 items.

My bachelor's thesis analyzes the current state of software applications for music content processing using artificial intelligence and sets research objectives. A comparative analysis of analogues of the developed software application was conducted, their advantages and disadvantages were identified, and the feasibility of developing the software application was proven.

In the course of the work, a software application was created that allows you to provide processed audio files divided into vocals and instrumentals.

The bot was developed using the Python language and the PyCharm environment. Spleeter, a custom-designed AI tool, was used.

The results obtained in the bachelor's thesis can be used by musicians and music enthusiasts for processing music files.

Keywords: telegram bot, artificial intelligence, music, musicians.

ЗМІСТ

ВСТУП 3
1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ 7
1.1 Аналіз стану програмних застосунків для обробки музичного контенту з використанням штучного інтелекту 7
1.2 Порівняльний аналіз аналогів 9
1.3 Аналіз методів розв’язання задачі 19
1.4 Висновки 21
2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ЗАСТОСУНКУ 22
2.1 Аналіз вхідних даних системи 22
2.2 Аналіз вимог до UI/UX дизайну програмного застосунку 23
2.3 Розробка моделі системи 26
2.4 Розробка алгоритмів роботи системи 27
2.5 Висновки 29
3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ЗАСТОСУНКУ 30
3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення 30
3.2 Розробка інтерфейсу програмного застосунку 32
3.3 Розробка програмного модулю обробки запитів користувача 38
3.4 Розробка системи збереження оброблених аудіофайлів за допомогою бази даних 44
3.5 Висновки 46
4 ТЕСТУВАННЯ ПРОГРАМИ 47
4.1 Тестування системи 47
4.2 Розробка інструкції користувача 50
4.3 Висновки 51
ВИСНОВКИ 52
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 53
ДОДАТКИ 55
ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ .. **Помилка! Закладку не визначено.**
ДОДАТОК Б – ПЕРЕВІРКА НА ПЛАГІАТ ..... **Помилка! Закладку не визначено.**
ДОДАТОК В – ЛІСТИНГ ПРОГРАМИ 61
ДОДАТОК Г – ІЛЮСТРАТИВНА ЧАСТИНА 76

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі музика стала невід'ємною частиною життя людей. Вона супроводжує нас у повсякденних справах, на роботі, під час відпочинку та подорожей. З розвитком технологій з'явилися нові можливості для доступу до музики, її обробки та створення. Це відкрило двері для нових інструментів, які можуть автоматизувати багато завдань, пов'язаних з музикою, економлячи час та ресурси [1].

Одним із таких інструментів є мобільні та веб-застосунки для обробки музичного контенту. Їх функціонал може варіюватися від простих завдань, таких як конвертація форматів файлів або редагування тегів, до складних операцій, таких як поділ вокалу та інструменталу, автоматичне генерування музики або створення реміксів [2].

Завдяки доступності та зручності використання, ці застосунки стають дедалі популярнішими серед музикантів, продюсерів, любителів музики та інших користувачів. Їх використання може значно полегшити та прискорити роботу з музичним контентом, роблячи її більш доступною та творчою [3].

Наприклад, музиканти можуть використовувати такі застосунки для: запису та редагування своїх творів, створення музичних композицій за допомогою MIDI-контролерів та віртуальних інструментів, додавання ефектів та обробки звуків, співпраці з іншими музикантами над спільними проєктами, публікації своєї музики в Інтернеті.

Продюсери можуть використовувати ці програми для: запису та мастерингу музичних альбомів, створення саундтреків до фільмів та відеоігор, ліцензування музики для використання в комерційних проєктах, просування музики через соціальні мережі та онлайн-платформи.

Любителі музики можуть використовувати ці застосунки для: прослуховування улюбленої музики в будь-який час і в будь-якому місці, створення плейлистів та колекцій музики, відкриття нової музики та виконавців, навчання грі на музичних інструментах, створення власних музичних

композицій.

Зростаюча популярність застосунків для обробки музичного контенту стимулює розвиток цієї галузі. З'являються нові та інноваційні застосунки, які пропонують все ширший спектр функцій та можливостей, тому робота з музикою стає більш доступною та творчою для людей з різним досвідом та навичками [4].

Наявні аналогічні програмні застосунки для обробки музичного контенту не надають достатнього та корисного функціоналу, або недоступні для звичайних користувачів. Застосунки з більшим функціоналом, як правило, мають платний доступ або призначені лише для внутрішнього використання у компаніях та дистрибуціях, тому розробка власного застосунку для обробки музичного контенту стає надзвичайно актуальною. Це дає змогу не лише забезпечити індивідуалізовані рішення, специфічні для конкретних потреб користувача, але й надати доступність подібного функціоналу для широкої авдиторії.

Тому тема роботи «Розробка телеграм-бота для обробки музичного контенту з використанням штучного інтелекту» є актуальною.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно з планом виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Мета програмного застосунку для обробки музичного контену полягає в забезпеченні зручного та надійного сервісу для розділення аудіофайлів на звукові доріжки за вокалом та інструменталом з використанням можливостей штучного інтелекту.

Відповідно до мети роботи необхідно виконати наступні завдання:

– визначити вимоги до програмного застосунку;

– розробити модуль для обробки та зберігання оброблених файлів користувачів за допомогою штучного інтелекту;

– розробити графічний інтерфейс бота, який буде зручним та адаптивним для користувачів різного рівня;

– інтегрувати різні модулі в єдиний програмний застосунок, що може бути використаний для обробки музичного контенту;

– провести тестування програмного застосунку.

Об’єкт дослідження – процес розробки телеграм-бота для обробки музичного контенту з використанням штучного інтелекту.

Предмет дослідження – методи і засоби розробки телеграм-бота для обробки музичного контенту з використанням штучного інтелекту.

Методи дослідження. Під час дослідження було використано методи розробки телеграм-ботів; методи створення клієнтського інтерфейсу; методи побудови блок-схем алгоритмів, методи реалізації програмних застосунків мовою Python. Було використано засоби розробленого штучного інтелекту Spleeter.

Новизна отриманих результатів. Запропонований новий інструмент для обробки музичного контену, який має функціонал для роботи майже на всіх сучасних пристроях, що підвищує доступність для користувача, а також здійснює обробку та збереження музичних файлів.

1. Удосконалено метод обробки запитів користувача, який, на відміну від існуючих реалізацій, використовує інструменти Python з використанням засобів штучного інтелекту для розробки телеграм-бота, що дозволяє розділяти музичні композиції на аудіодоріжки за вокалом та інструменталом. Це значно спрощує процес використання застосунку для кінцевого користувача, забезпечуючи високу сумісність та зручність у експлуатації.

2. Удосконалено метод збереження оброблених аудіофайлів, який відрізняється використанням бази даних для завантажених користувачами пісень і дозволяє повторно використовувати ці аудіотреки у разі потреби доступу інших користувачів. Це сприяє зменшенню навантаження на серверні ресурси та підвищує загальну продуктивність системи.

Практична цінність отриманих результатів. Практична цінність отриманих результатів полягає у покращенні розуміння музичної теорії на

практиці через розділення аудіофайлів на звукові доріжки за вокалом та інструменталом з використанням можливостей штучного інтелекту.

Особистий внесок здобувача. Отримані результати дослідження та розробки представлені автором особисто. Зокрема, у тезах доповіді автор представляє аналіз відомих рішень.

Апробація результатів здійснена на міжнародній науково-практичній інтернет-конференції студентів, аспірантів та молодих науковців «МОЛОДЬ В НАУЦІ: ДОСЛІДЖЕННЯ, ПРОБЛЕМИ, ПЕРСПЕКТИВИ. МН-2024», (Вінниця, 2023).

Публікації. Тези доповідей представлені в збірнику міжнародної науково-практичної інтернет-конференції студентів, аспірантів та молодих науковців «МОЛОДЬ В НАУЦІ: ДОСЛІДЖЕННЯ, ПРОБЛЕМИ, ПЕРСПЕКТИВИ. МН-2024», (Вінниця, 2024).

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану програмних застосунків для обробки музичного контенту з використанням штучного інтелекту

У сучасному світі музика стала невід'ємним супутником нашого життя. Вона лунає з динаміків гаджетів, супроводжує нас у повсякденних справах, на роботі, під час відпочинку та подорожей. Технологічний прогрес відкрив нові горизонти для доступу до музики, її обробки та створення. Завдяки розвитку технологій ми маємо можливість слухати улюблені треки у будь-який момент, використовувати різноманітні застосунки для редагування музичних файлів та навіть створювати власні музичні композиції без професійного обладнання. З'явилися інструменти, які автоматизують багато завдань, пов'язаних з музикою, економлячи час та ресурси. Сучасні технології дозволяють не лише слухати музику, але й активно взаємодіяти з нею, що відкриває нові перспективи для творчості та самовираження.

Одним із таких інструментів є мобільні та веб-застосунки для обробки музичного контенту. Їх функціонал вражає різноманіттям: від простих завдань, таких як конвертація форматів файлів або редагування, до складних операцій, таких як поділ вокалу та інструменталу або автоматичне генерування музики. Ці застосунки стали незамінними помічниками для багатьох людей, адже вони дозволяють швидко та ефективно вирішувати задачі, пов'язані з музикою, незалежно від рівня професіоналізму користувача. Вони значно спрощують процеси створення та редагування музичних треків, роблячи ці процеси доступними для широкого кола користувачів.

Простота та доступність цих програм роблять їх дедалі популярнішими серед музикантів, продюсерів, любителів музики та широкого загалу. Їх використання значно полегшує та прискорює роботу з музичним контентом, роблячи її більш доступною та легкою для незалежних артистів. Завдяки цим інструментам будь-хто може займатися музикою, експериментувати з різними

жанрами та стилями, створювати нові композиції та ділитися ними зі світом. Ці програми відкривають нові можливості для творчості, дозволяючи користувачам реалізовувати свої музичні ідеї без необхідності володіти спеціальними знаннями чи обладнанням [5].

Зростаючий попит на мобільні та веб-застосунки для обробки музичного контенту стимулює розвиток галузі інформаційних технологій [6]. З'являються нові та інноваційні програми, які пропонують все ширший спектр функцій та можливостей. У майбутньому можна очікувати, що ці програми стануть ще більш потужними та зручними у використанні, відкриваючи нові горизонти для музичного самовираження та творчості. Такі програми постійно вдосконалюються, враховуючи потреби користувачів і тенденції ринку, що робить їх незамінними у процесі створення та обробки музики. З розвитком штучного інтелекту та машинного навчання ці застосунки можуть надавати ще більш інноваційні та зручні рішення для музикантів та любителів музики.

Водночас, важливо зазначити, що розробка такого телеграм-бота потребує значних технічних зусиль та інноваційних підходів. Необхідно враховувати вимоги до обробки аудіо, забезпечення високої якості розділення звукових доріжок, інтеграцію з різними музичними сервісами та забезпечення безпеки даних користувачів. Такий бот повинен мати простий користувацький інтерфейс, що дозволить користувачам легко взаємодіяти з ним, навіть без глибоких технічних знань. Впровадження сучасних алгоритмів штучного інтелекту та машинного навчання допоможе досягти високої точності та ефективності у розпізнаванні та обробці музичних файлів, що, у свою чергу, забезпечить високу якість кінцевого продукту та задоволення потреб користувачів.

Отже, розробка телеграм-бота для обробки музичного контенту з використанням штучного інтелекту є актуальною і важливою через постійне зростання кількості зацікавлених в музиці людей та розвитком технологій. З його використанням користувачі зможуть самостійно досліджувати музичні твори через розділення аудіофайлів на звукові доріжки за вокалом та інструменталом з використанням можливостей штучного інтелекту. Це дозволить їм глибше

зануритися в музичні композиції, аналізувати кожну складову окремо і отримувати більше задоволення від прослуховування та створення музики. Завдяки таким інструментам користувачі матимуть змогу краще розуміти структуру музичних творів, експериментувати з різними елементами композицій та відкривати нові аспекти своєї творчості.

1.2 Порівняльний аналіз аналогів

В умовах, коли все більше людей цікавиться музикою та її створенням, важливо мати інструменти, які дозволяють ефективно керувати музичним контентом та оброблювати його. Сучасний світ надає безліч можливостей для самовираження та творчості, і музика не є винятком. Почати розвиватись у музичному напрямку непросто, і програмні застосунки для обробки музичного контенту стають незамінними інструментами у забезпеченні ефективності та успішності в цьому новому напрямку розвитку [7]. Завдяки таким інструментам, музиканти, продюсери та аматори мають можливість легко та швидко редагувати свої музичні треки, експериментувати з різними звуками та стилями, а також ділитися своєю творчістю з іншими. Ці застосунки дозволяють не лише покращувати якість музики, але й відкривають нові горизонти для творчих експериментів.

Сучасні програмні рішення надають широкий спектр можливостей для обробки музики. Вони дозволяють розділяти звукові доріжки, додавати ефекти, змінювати темп та тональність, а також проводити глибокий аналіз музичних творів. Це значно спрощує процес створення музики та робить його доступним навіть для тих, хто тільки починає свій шлях у музичній індустрії. Рішенням є різноманітні застосунки та платформи, які пропонують користувачам простий зрозумілий інтерфейс, потужні інструменти для редагування та зручні функції для збереження та експорту готових треків. Вони допомагають не лише професіоналам, але й аматорам розкривати свій творчий потенціал та досягати нових висот у створенні музики.

До найбільш відомих рішень відносять:

– Phonic;
– Lalal.ai;
– Audacity;
– FL Studio.

Одним із прикладів сервісу для обробки музичного контенту є вебсайт Phonic [8]. Phonic – це безкоштовний онлайн-сервіс, який використовує штучний інтелект для розбиття пісень на вокал, інструментальні партії та інші звукові доріжки (рис. 1.1). Цей сервіс став популярним серед музикантів та продюсерів завдяки своїй простоті у використанні та високій ефективності. Він дозволяє користувачам швидко та легко розділяти музичні треки на окремі компоненти, що значно полегшує процес обробки та редагування музики. Крім того, Phonic надає можливість зберігати отримані доріжки у різних форматах, що робить його ще більш зручним для роботи.

Phonic є чудовим прикладом того, як сучасні технології штучного інтелекту можуть бути використані для покращення та спрощення процесу обробки музичного контенту. Використовуючи складні алгоритми, цей сервіс здатен точно та швидко розпізнавати різні елементи музичних творів і розділяти їх на окремі доріжки. Це особливо корисно для тих, хто хоче створювати ремікси, вивчати окремі інструментальні партії або просто аналізувати структуру музичних композицій. Безкоштовний доступ до таких потужних інструментів відкриває нові можливості для музикантів різного рівня, від початківців до професіоналів, дозволяючи їм експериментувати та вдосконалювати свої навички.

Переваги сервісу Phonic:

– безкоштовний;
– легкий у використанні;
– швидкий та точний;
– підтримує широкий спектр форматів файлів;
– може розбивати пісні на кілька звукових доріжок.

Недоліки сервісу Phonic:

– не пропонує таких просунутих функцій, як деякі платні сервіси;

– іноді може давати неточні результати.

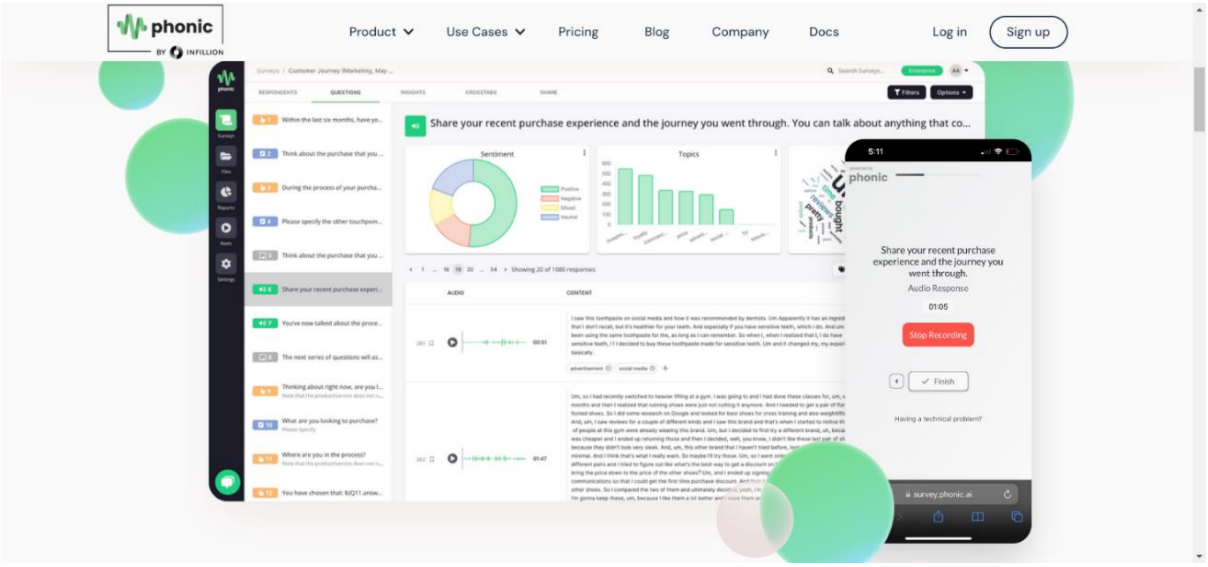

Рисунок 1.1 – Головне вікно вебсайту Phonic

Phonic – це хороший вибір для початківців та користувачів, яким потрібен швидкий і простий спосіб розбити пісні на звукові доріжки. Завдяки інтуїтивно зрозумілому інтерфейсу, навіть ті, хто не має великого досвіду у роботі з музичним софтом, можуть легко скористатися можливостями цього сервісу. Процес розбиття пісні займає всього кілька хвилин і не потребує спеціальних знань чи навичок. Користувачам достатньо завантажити файл і обрати необхідні параметри, після чого Phonic автоматично виконає всю роботу. Це значно економить час і дозволяє зосередитися на творчих аспектах музики.

Інший приклад – вебсайт Lalal.ai [9]. Lalal.ai – це платний онлайн-сервіс, який використовує штучний інтелект для розбиття пісень на вокал та інструментальні партії (рис. 1.2). Цей сервіс здобув популярність серед професіоналів музичної індустрії завдяки високій точності та якості обробки аудіофайлів. Використовуючи передові алгоритми машинного навчання, Lalal.ai здатен з великою точністю відокремлювати вокальні доріжки від інструментальних, що робить його чудовим інструментом для створення

реміксів, караоке-треків або детального аналізу музичних композицій.

Lalal.ai відрізняється своєю потужністю та функціональністю, що робить його привабливим вибором для музикантів, продюсерів та звукових інженерів. Сервіс надає користувачам можливість завантажувати великі аудіофайли і отримувати результати обробки за короткий час. Крім того, Lalal.ai підтримує різні формати файлів, що дозволяє використовувати його для обробки музики з будь-якого джерела. Ця універсальність та зручність у використанні роблять Lalal.ai популярним серед тих, хто прагне отримати якісний результат без зайвих зусиль.

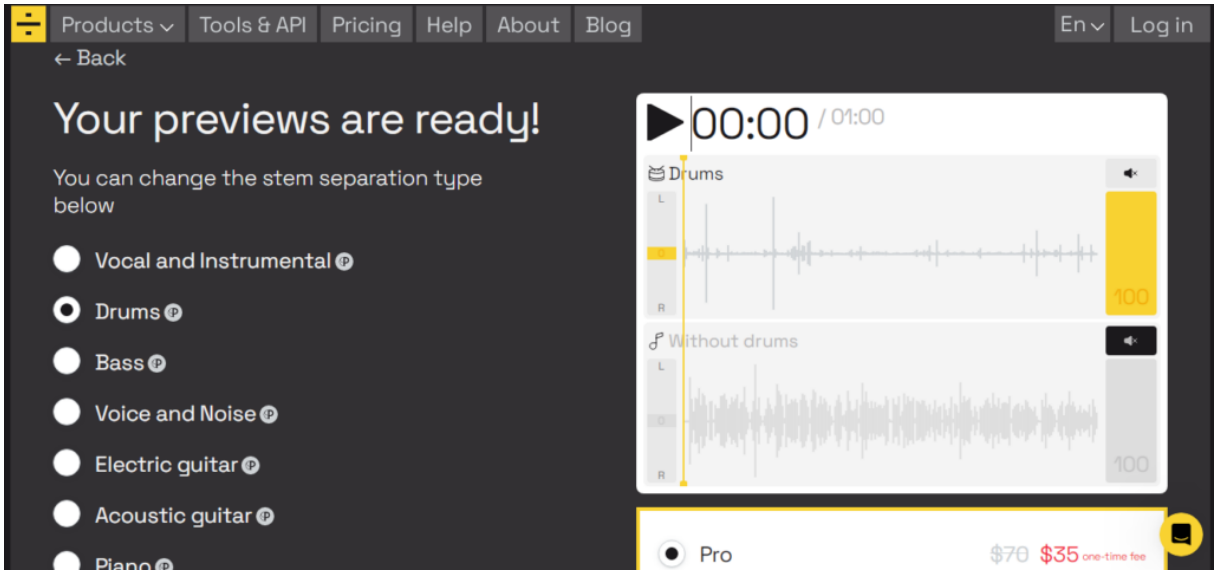

Рисунок 1.2 – Головне вікно вебсайту Lalal.ai

Переваги сервісу Lalal.ai:

– висока точність розбиття;
– можливість налаштування процесу розбиття;
– збереження результатів у різних форматах;
– додаткові функції, такі як видалення шуму та ремастеринг.

Недоліки сервісу Lalal.ai:

– платний;
– не пропонує розбиття на кілька звукових доріжок вручну.

Окрім основної функції розділення пісень на вокал та інструментальні партії, Lalal.ai пропонує додаткові можливості для покращення та оптимізації аудіо. Наприклад, користувачі можуть налаштовувати параметри обробки для досягнення бажаного ефекту або користуватися різними фільтрами для покращення якості звуку. Це дозволяє Lalal.ai задовольняти потреби найвибагливіших користувачів, надаючи їм потужний інструмент для роботи з музикою на професійному рівні.

Завдяки цим перевагам, Lalal.ai стає важливим інструментом у арсеналі сучасного музиканта, який прагне досягти високих результатів у своїй творчості.

Ще одним прикладом застосунку для обробки музичного контенту є Audacity (рис. 1.3). Audacity – це безкоштовне програмне забезпечення з відкритим кодом для редагування аудіо, яке можна використовувати для розбиття пісень на звукові доріжки вручну [10].

Audacity пропонує широкий спектр функцій для редагування та аналізу аудіо, що робить його цінним інструментом для музикантів, продюсерів та любителів аудіо. Audacity може записувати аудіо з мікрофона або інших джерел, а також відтворювати записані файли або імпортовані аудіофайли. Можливо записувати як моно-, так і стереозвук, а також регулювати рівень гучності та інші налаштування запису.

Також сервіс дозволяє редагувати аудіофайли з точністю до зразка, а саме: вирізати, копіювати, вставляти та видаляти частини запису, а також застосовувати різні ефекти, такі як посилення, зменшення шуму, еквалайзер та компресія. Також можливо використовувати інструменти для редагування, такі як інструмент "Time Shift" для переміщення частин запису в часі, та інструмент "Draw" для малювання безпосередньо на звуковій хвилі. Audacity дозволяє розбивати стереофайли на окремі монофонічні звукові доріжки. Це може бути корисно для ізоляції окремих інструментів або вокалу в пісні. Можливо розділити файли за допомогою інструменту "Split Stereo Track" або вручну виділити та розділити канали. Також його можна використовувати для створення музики з нуля: записувати власні інструменти та вокал, а також використовувати

вбудовані генератори звуків для створення MIDI-мелодій та ритмів, імпортувати MIDI-файли та редагувати. Audacity має вбудовані інструменти для аналізу аудіо, такі як спектроаналізатор та осцилограф. Це може бути корисно для вивчення частотного вмісту ваших записів, а також для виявлення та видалення шуму.

Переваги застосунку Audacity:

– безкоштовний;

– повний контроль над процесом розбиття;

– широкий спектр функцій редагування аудіо;

– можна використовувати для розбиття пісень на кілька звукових доріжок.

Недоліки застосунку Audacity:

– може бути складним для початківців;

– не пропонує автоматичного розбиття аудіодоріжок;

– вимагає більше часу та зусиль, ніж автоматичні сервіси.

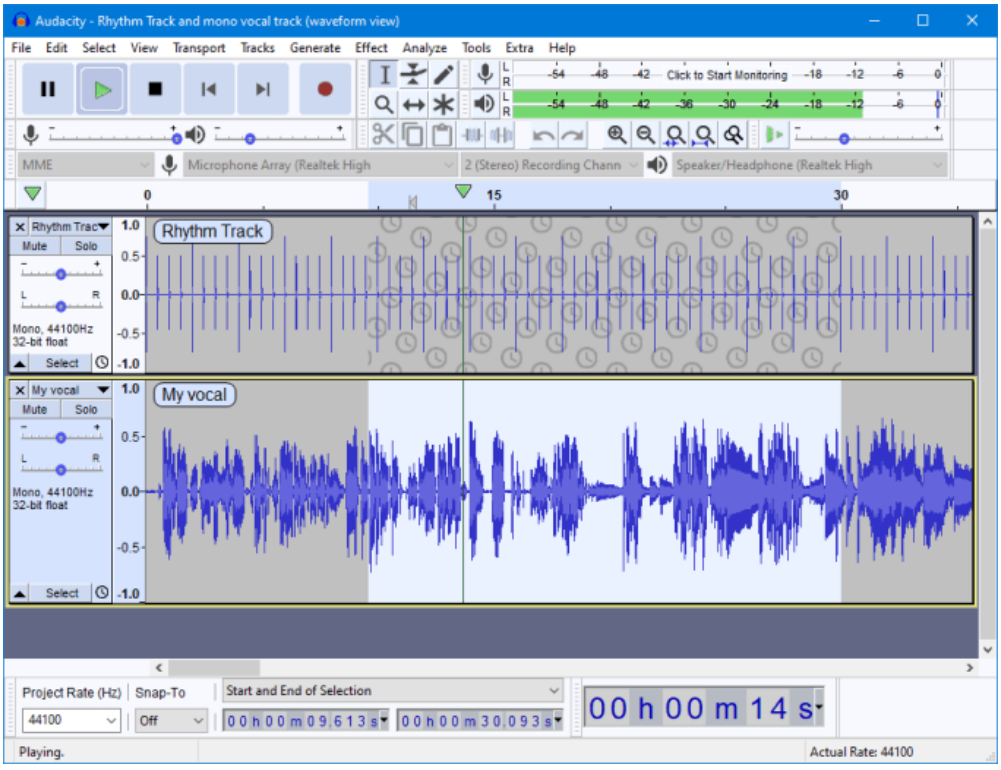

Рисунок 1.3 – Приклад інтерфейсу застосунку Audacity

Audacity простий у використанні для початківців, але також пропонує достатньо функцій для задоволення потреб досвідчених користувачів. Він доступний для різних операційних систем, включаючи Windows, macOS та Linux. Крім того, Audacity має велику та активну спільноту користувачів, які забезпечують підтримку та ресурси. Можна знайти онлайн-форуми, навчальні посібники та інші ресурси, які допоможуть розпочати роботу з Audacity.

Audacity – це потужний і універсальний інструмент, який можна використовувати для різних цілей, пов'язаних з обробкою музики. Він є цінним доповненням до набору інструментів будь-якого музиканта, продюсера чи любителя аудіо.

FL Studio – це платне програмне забезпечення для створення музики, яке також можна використовувати для розбиття пісень на звукові доріжки. Це програмне забезпечення є дуже потужним інструментом для музикантів, продюсерів та композиторів, оскільки воно забезпечує широкий спектр функцій для створення, редагування та мікшування музичних композицій.

FL Studio дозволяє користувачам працювати з різними типами аудіофайлів, застосовувати ефекти, автоматизувати процеси і використовувати віртуальні інструменти. Крім того, його інтуїтивний інтерфейс і великий набір вбудованих інструментів роблять його доступним як для професіоналів, так і для початківців. FL Studio підтримує багатодоріжковий запис, редагування і мікшування, що дозволяє користувачам створювати складні музичні проекти з високою точністю і контролем (рис. 1.4).

Переваги застосунку FL Studio:

- широкий спектр функцій для створення музики;
- можливість розбиття пісень на кілька звукових доріжок;
- висока точність розбиття.

Недоліки застосунку FL Studio:

- платний;
- може бути складним для початківців.

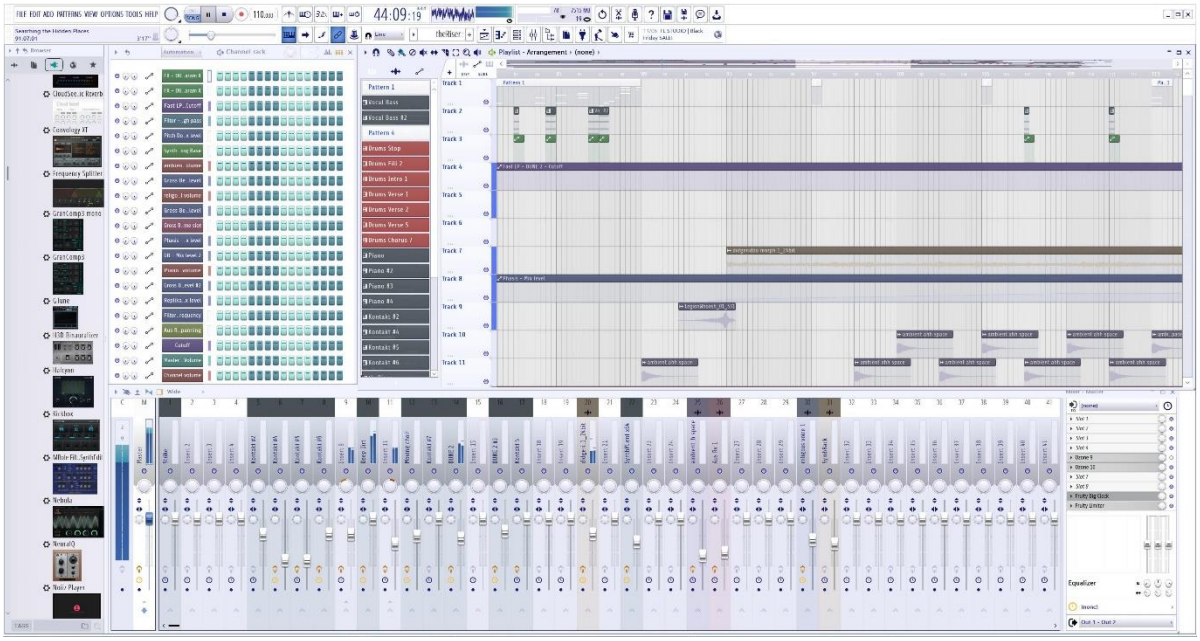

Рисунок 1.4 – Приклад інтерфейсу FL Studio

Загалом, FL Studio – це потужний інструмент для створення музики та редагування аудіо, який пропонує широкий спектр функцій. Хоча він не пропонує автоматичного розбиття композиції на звукові доріжки, існує кілька способів зробити це вручну або за допомогою сторонніх плагінів. FL Studio відомий своєю гнучкістю і можливістю налаштування, що дозволяє користувачам адаптувати його під свої потреби. Цей софт підтримує роботу з численними форматами аудіо, а також пропонує великий вибір вбудованих інструментів і ефектів, які можна використовувати для створення унікальних музичних треків. Крім того, FL Studio має активну спільноту користувачів, яка постійно ділиться новими плагінами, скриптами і порадами, що робить його ще більш універсальним.

Незважаючи на те, що розбиття композиції на звукові доріжки може вимагати додаткових зусиль і знань, FL Studio надає всі необхідні інструменти для досягнення професійного результату. Це програмне забезпечення ідеально підходить як для новачків, так і для досвідчених музикантів, які прагнуть максимального контролю над своїми музичними проектами [11].

Вибір вебсайту або застосунку для обробки музичного контенту залежить від потреб та бюджету. Кожен інструмент має свої особливості і переваги, що

робить його більш підходящим для певних користувачів.

Phonic – це хороший вибір для початківців та користувачів, яким потрібен швидкий і простий спосіб розбити пісні на звукові доріжки. Цей інструмент ідеально підходить для тих, хто тільки починає свій шлях у музиці і шукає доступний і ефективний спосіб роботи з аудіо.

Lalal.ai – це хороший вибір для користувачів, яким потрібна висока точність розбиття вокалу та інструментальних партій. Цей сервіс використовує передові алгоритми штучного інтелекту для забезпечення найвищої якості розбиття аудіо, що робить його ідеальним для професійних музикантів та продюсерів.

Audacity підходить для користувачів, які мають досвід роботи з редагуванням аудіо та хочуть повного контролю над процесом розбиття. Завдяки своїм розширеним можливостям і відкритому коду, Audacity дозволяє детально налаштовувати процес редагування, що робить його вибором для тих, хто прагне до максимального контролю та гнучкості.

FL Studio – це хороший вибір для користувачів, які хочуть не лише розбивати пісні на звукові доріжки, але й створювати свою музику. Цей потужний інструмент надає безліч можливостей для створення, редагування і мікшування музики, що робить його ідеальним для композиторів, продюсерів і музикантів, які прагнуть створювати унікальні музичні твори з нуля. Незалежно від того, чи користувач початківець, який шукає простий інструмент, чи досвідчений професіонал, який потребує розширених можливостей, існує рішення, яке задовольнить його потреби та допоможе досягти бажаних результатів у роботі з музичним контентом.

Після аналізу усіх аналогів, було проведено порівняння із розроблюваним телеграм-ботом «HarmonyBot».

Результат порівняння аналогів представлено в таблиці 1.1

Після аналізу існуючих аналогів та їхніх недоліків, можна зробити висновок про доцільність розробки власного проєкту. У порівнянні з існуючими платформами, телеграм-бот «HarmonyBot» буде мати важливі переваги.

Таблиця 1.1 – Порівняльні характеристика

Характеристика	**Phonic**	**Lalal.ai**	**Audacity**	**FL Studio**	**HarmonyBot**
Інтерфейс українською мовою	-	-	-	-	+
Вільне безкоштовне користування	+	-	+	-	+
Автоматичне розбиття інструменталу на звукові доріжки	+	+	-	-	+
Можливість користуватись вільно у месенджері Telegram	-	-	-	-	+
Загальна оцінка	50%	25%	25%	0%	100%

По-перше, бот буде мати інтерфейс українською мовою. Це значно полегшить користування платформою для українців, роблячи її доступнішою та зрозумілішою для широкого кола користувачів у нашій країні.

По-друге, основний функціонал телеграм-бота буде безкоштовним, що надасть більше доступності для різних користувачів, що буде впливати на популярність бота. Безкоштовний доступ до основних можливостей робить платформу привабливою для новачків та професіоналів, які шукають ефективний інструмент без додаткових витрат.

По-третє, буде розроблено функціонал автоматичного розбиття композиції на звукові доріжки, що значно полегшить роботу з музикою для користувачів. Ця функція дозволить користувачам швидко й ефективно обробляти музичні файли, роблячи процес роботи з музикою більш зручним та продуктивним.

По-четверте, проєкт буде запущено у месенджері Telegram, що надасть

більше доступності та полегшить роботу користувачам. Telegram є популярним месенджером з великою базою користувачів, що забезпечить широке охоплення та легкість доступу до бота з будь-якого пристрою. Це також забезпечить швидке впровадження нових функцій та оновлень, підтримуючи високий рівень задоволеності користувачів.

Таким чином, «HarmonyBot» стане потужним інструментом для обробки музичного контенту, забезпечуючи зручність, доступність та інноваційні можливості для користувачів.

Отже, розробка телеграм-бота для обробки музичного контенту з використанням штучного інтелекту надасть більше доступних інструментів для обробки та аналізу музичних композицій. Завдяки широкому спектру функцій, включаючи автоматичне розбиття композицій на звукові доріжки, користувачі зможуть значно підвищити свою продуктивність та творчий потенціал. Бот також забезпечить зручний доступ до цих можливостей через популярний месенджер Telegram, що робить його доступним для широкої аудиторії. Розширені можливості штучного інтелекту дозволять ботові виконувати складні завдання з мінімальними зусиллями з боку користувачів. Таким чином, телеграм-бот для обробки музичного контенту стане незамінним помічником, пропонуючи якісні інструменти для максимально ефективного використання музичних композицій у повсякденному житті та професійній діяльності.

1.3 Аналіз методів розв’язання задачі

Розробка телеграм-бота для обробки музичного контенту з використанням штучного інтелекту вимагає ретельного розгляду підходів до вирішення проблеми. Існують різні методи, кожен з яких має свої переваги та недоліки. У цьому розділі проаналізовано найбільш ефективні методи розробки таких застосунків.

Одним з варіантів вирішення проблеми розробки застосунку для обробки музичного контенту з використанням штучного інтелекту є створення стандартного десктопного застосунку. Такий підхід дає можливість працювати

за персональним комп’ютером, маючи доступ до інтернету та інтеграцію інструментів операційної системи. Однак такий підхід має недоліки: він вимагає встановлення та оновлення на кожному комп'ютері, що може бути незручним для звичайних користувачів. Також це ускладнює інтеграцію штучного інтелекту до розроблюваного застосунку. Тому такий підхід не варто застосовувати

Інший підхід полягає в розробці вебсервісу. У цьому варіанті програмний застосунок розробляється як вебсервіс, який доступний через браузер. Це дозволяє користувачам отримувати доступ до функціоналу застосунку з будь-якого пристрою, що підключений до Інтернету. Також у такому разі є можливість легко масштабуватись для великої кількості користувачів, це вимагає менше зусиль для розгортання та оновлення, оскільки оновлення відбуваються централізовано на сервері. Проте, у зв’язку з роботою з браузера, застосунок матиме обмежений доступ до операційної системи, і користувачу доведеться реєструвати свої дії вручну та проходити повноцінну реєстрацію. Такий підхід може бути зручним для деяких користувачів, але все ж не забезпечує повної гнучкості та зручності в роботі з музичним контентом.

Ще одним підходом до розробки програмного застосунку для обробки музичного контенту з використанням штучного інтелекту є телеграм-бот. Телеграм-бот може бути найкращим вибором для проєкту через широку аудиторію: Telegram має понад 500 мільйонів активних користувачів, що робить його однією з найпопулярніших платформ для обміну повідомленнями. Також важливим є зручний інтерфейс: Telegram має простий та інтуїтивно зрозумілий інтерфейс, який робить його легким у використанні як для розробників, так і для користувачів. Це може скоротити час розробки та покращити досвід роботи користувачів. Месенджер пропонує широкий спектр функцій, які можна використовувати для створення потужного та гнучкого бота. Також Telegram використовує наскрізне шифрування, що робить його однією з найбезпечніших платформ для обміну повідомленнями. Розробка та розгортання телеграм-бота є відносно недорогим процесом. Одним із найголовніших аспектів є доступність: Telegram доступний на різних платформах, включаючи iOS, Android, Windows,

macOS та Linux, а також його можна легко інтегрувати з іншими платформами та сервісами. Але цей підхід також має свої недоліки, а саме обмеженість інтерфейсу можливостями зовнішнього застосунку. Тим не менш, ці недоліки можна мінімізувати завдяки використанню сторонніх плагінів та розширень, що дозволить створити максимально функціональний і зручний бот.

Розглянувши переваги та недоліки кожного методу, було вирішено використати метод розробки телеграм-бота для реалізації програмного застосунку для обробки музичного контенту з використанням штучного інтелекту. Такий підхід дозволить отримати повноцінний програмний продукт, що буде доступним для якнайбільшої кількості користувачів, не жертвуючи важливим функціоналом. Телеграм-бот стане зручним інструментом як для початківців, так і для професіоналів, забезпечуючи їх усіма необхідними можливостями для обробки музичного контенту та розширюючи горизонти музичного самовираження.

1.4 Висновки

У першому розділі було розглянуто стан програмних застосунків для обробки музичного контенту з використанням штучного інтелекту. Було проведено порівняння відомих рішень для обробки музичного контенту, а саме: Phonic, Lalal.ai. Audacity та FL Studio.

У результаті порівняння було виявлено, що досліджувані рішення набули значної популярності завдяки своєму функціоналу та підтримці користувачів. Проаналізувавши переваги та недоліки відомих платформ, було виявлено, що всі рішення надають можливість обробки музичного контенту, проте не всі роблять це автоматично. Також половина з них є платними. Таким чином було доведено доцільність розробки власного програмного застосунку. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власного програмного застосунку.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ЗАСТОСУНКУ

2.1 Аналіз вхідних даних системи

Аналіз вхідних даних є ключовим етапом у розробці програмного застосунку для обробки музичного контенту. Цей процес передбачає не лише прийом інформації, а й її обробку для подальшого передавання користувачу для подальшого використання.

Особиста інформація користувача є одним з перших типів вхідних даних, які варто ретельно розглянути. Телеграм надає можливість збирати певну частину даних користувачів в залежності від налаштувань приватності. Персональні дані, такі як ім'я та контактні дані не лише допомагають ідентифікувати користувача, але й створюють основу для персоналізованих послуг та взаємодії з ним. Важливо вберегти цю інформацію від третіх осіб задля безпеки самих користувачів.

Основним типом вхідних даних для застосунку є музичні файли. Застосування штучного інтелекту для розділення аудіофайлів на звукові доріжки за вокалом та інструменталом неможливе без музичних файлів. Якість розділення музичних файлів буде основним критерієм оцінки роботи програмного застосунку. Також музичні файли є тим, що користувачі хочуть обробляти і заради чого використовуватимуть бот. Розділення музичних файлів на доріжки за вокалом та інструменталом може допомогти користувачам краще зрозуміти структуру та компонування музики. До того ж, вони можуть бути використані для розширення функціональності програмного застосунку в майбутньому, наприклад, для створення інструментів для створення музики або автоматичного генерування музики.

Важливо зазначити, що якість та різноманітність музичних файлів, які використовуватимуть користувачі, впливатимуть на результати роботи програмного застосунку. Тому забезпечення підтримки різних форматів аудіофайлів та різних жанрів музики стане важливим аспектом у процесі

розробки та тестування застосунку.

Узагальнюючи, важливість аналізу вхідних даних полягає в зрозумінні потреб користувачів і покращенні функціоналу програмного застосунку. Цей процес допомагає оптимізувати обробку музичного контенту, що впливатиме на загальну ефективність роботи бота. Проаналізувавши ці дані, розробники можуть знайти шляхи для поліпшення функціоналу та вдосконалення користувацького досвіду.

Зрештою, розуміння і врахування потреб користувачів дозволить створити телеграм-бота, який відповідатиме їхнім очікуванням та забезпечить високу якість обробки музичного контенту. Використання штучного інтелекту для аналізу та розділення аудіофайлів надає можливість створити інноваційний продукт, який буде корисним як для професійних музикантів, так і для любителів музики. Телеграм-бот «HarmonyBot» стане інструментом, що надасть користувачам зручний доступ до потужних функцій обробки музики, сприяючи їхньому творчому розвитку та задоволенню потреб у сфері музики.

2.2 Аналіз вимог до UI/UX дизайну програмного застосунку

Створення ефективного та зручного інтерфейсу для телеграм-бота є важливим етапом розробки програмного забезпечення. Інтерфейс визначає, як користувачі взаємодіють з ботом і як вони отримують інформацію та функціональні можливості, які бот надає. Враховуючи специфіку платформи Telegram, необхідно ретельно розглянути вимоги до UI/UX дизайну, щоб забезпечити зрозумілий та приємний досвід користування.

UI (User Interface) дизайн зосереджується на візуальних аспектах і макеті інтерфейсу, тоді як UX (User Experience) дизайн зосереджується на загальному користувацькому досвіді, включаючи зручність та ефективність взаємодії з ботом [12]. Для створення телеграм-бота обидва аспекти повинні бути гармонійно інтегровані.

Ключові елементи UI для телеграм-ботів:

1. Текстові повідомлення:

– привітальні повідомлення: коли користувач вперше ініціює бота, він отримує привітальне повідомлення, яке пояснює, що може робити бот і як ним користуватися;

– команди: бот має підтримувати основні команди, такі як «/start», «/help», «/info», які надають користувачам доступ до необхідної інформації та функціональності.

2. Кнопки:

– інлайн-кнопки: ці кнопки з'являються безпосередньо під повідомленням і дозволяють користувачам виконувати дії, не виходячи з чату. Наприклад, кнопки для завантаження вокальних та інструментальних треків;

– кнопки меню: користувачі можуть використовувати ці кнопки для навігації по різних функціях бота.

3. Меню швидкого доступу:

– клавіатура: налаштовувана клавіатура, яка надає користувачам швидкий доступ до основних команд і функцій бота;

– меню команд: перелік команд, які користувач може ввести для взаємодії з ботом.

4. Сповіщення та відповіді:

– сповіщення: інформування користувача про завершення певних дій, наприклад, про успішне завантаження або обробку файлів;

– автовідповіді: заздалегідь визначені відповіді на часто задавані питання або дії користувачів.

UX для телеграм-ботів:

1. Інтуїтивність:

– простота використання: інтерфейс повинен бути максимально простим і зрозумілим. Користувачі повинні швидко розуміти, як використовувати бота;

– чіткі інструкції: кожен крок і функція повинні супроводжуватися чіткими інструкціями та підказками.

2. Швидкість та ефективність:

– швидка реакція: бот повинен швидко відповідати на запити користувача, щоб забезпечити плавний досвід;

– ефективність взаємодії: мінімізація кількості кроків, необхідних для виконання завдання.

3. Персоналізація:

– персоналізовані повідомлення: вітання користувачів за іменем, якщо це можливо, і надання персоналізованих рекомендаці;

– індивідуальні налаштування: можливість налаштування інтерфейсу та функціоналу бота під індивідуальні потреби користувача.

4. Доступність:

– мультиплатформенність: бот повинен бути доступним на всіх пристроях, де підтримується Telegram, включаючи смартфони, планшети та комп’ютери;

– локалізація: інтерфейс повинен бути доступний на кількох мовах, включаючи українську.

Проектування інтерфейсу для телеграм-бота:

1. Вітальний екран:

– привітальне повідомлення з коротким описом можливостей бота;

– кнопки для швидкого доступу до основних функцій: "Завантажити пісню", "Отримати інформацію", "Допомога".

2. Основний екран:

– кнопки для завантаження файлів і отримання результатів обробки;

– меню команд для вибору дій (завантажити пісню, переглянути оброблені файли, змінити налаштування).

3. Екран обробки:

– інформація про поточний статус обробки файлу;

– повідомлення про завершення обробки з посиланнями на результати.

4. Екран налаштувань:

– опції для зміни мови інтерфейсу;

– налаштування форматів файлів і інших параметрів роботи бота.

Розробка структури інтерфейсу програмного застосунку для телеграм-бота вимагає детального аналізу потреб користувачів та врахування специфіки платформи Telegram. Важливо забезпечити інтуїтивно зрозумілий, ефективний та персоналізований інтерфейс, що дозволить користувачам легко взаємодіяти з ботом та отримувати максимальну користь від його функціоналу. Ретельно спроєктований UI/UX дизайн є ключем до створення успішного телеграм-бота для обробки музичного контенту.

2.3 Розробка моделі системи із використанням методу обробки запитів користувачів

UML (Unified Modeling Language) діаграма діяльності – це тип діаграми, який використовується для моделювання поведінки системи [13]. Вона зображує послідовність дій або кроків, що виконуються в процесі, а також потік управління між цими діями.

Для кращого розуміння роботи програмного застосунку для обробки музичного контенту з використанням штучного інтелекту, було розроблено модель роботи системи на прикладі UML діаграми діяльності (рис. 2.1).

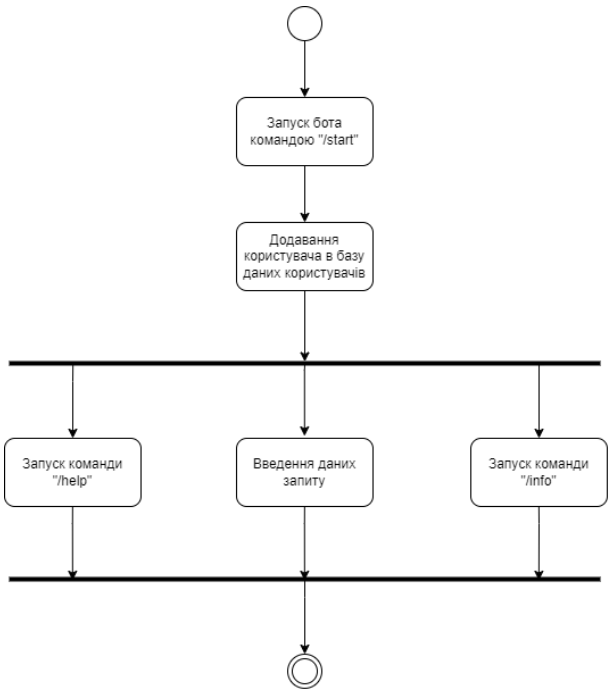

Рисунок 2.1 – Діаграма діяльності телеграм-бота

Діаграми діяльності є одним із найбільш корисних інструментів для аналізу і проєктування складних процесів і систем, оскільки вони допомагають візуалізувати динамічні аспекти системи.

Згідно з цією діаграмою, першою після запуску бота запускається команда «/start», яка дає користувачеві інформацію, як коректно поставити запит. Наступним кроком користувач додається в базу даних і після цього може обирати наступний крок. Користувач може обрати одну із наявних команд для ознайомлення з роботою бота або ж надати інформацію для обробки.

2.4 Розробка алгоритмів збереження оброблених аудіофайлів

Для розробки програмного застосунку для обробки музичного контенту з використанням штучного інтелекту, необхідно розробити алгоритм, згідно з яким буде працювати цей застосунок. Розробка алгоритму є ключовим етапом, який визначає логіку та послідовність дій, що виконуватимуться під час роботи програмного забезпечення. Щоб забезпечити чітке та зрозуміле уявлення про алгоритм, було створено блок-схему.

Блок-схема являє собою графічне зображення алгоритму, що дозволяє візуалізувати послідовність операцій та умов, які повинні бути виконані під час роботи програми. Це надає можливість не лише зрозуміти, як саме працює алгоритм, але й полегшує процес розробки та налагодження програмного коду.

Блок-схема також служить важливим інструментом для документації процесу розробки. Вона дозволяє легко зберігати та передавати інформацію про алгоритм іншим розробникам або зацікавленим сторонам, що може бути особливо корисним у випадках, коли над проектом працює велика команда або коли проект передається іншим розробникам для подальшої підтримки та розвитку.

Ця блок-схема описує загальний алгоритм роботи бота, який працює циклічно до завершення роботи застосунку (рис. 2.2).

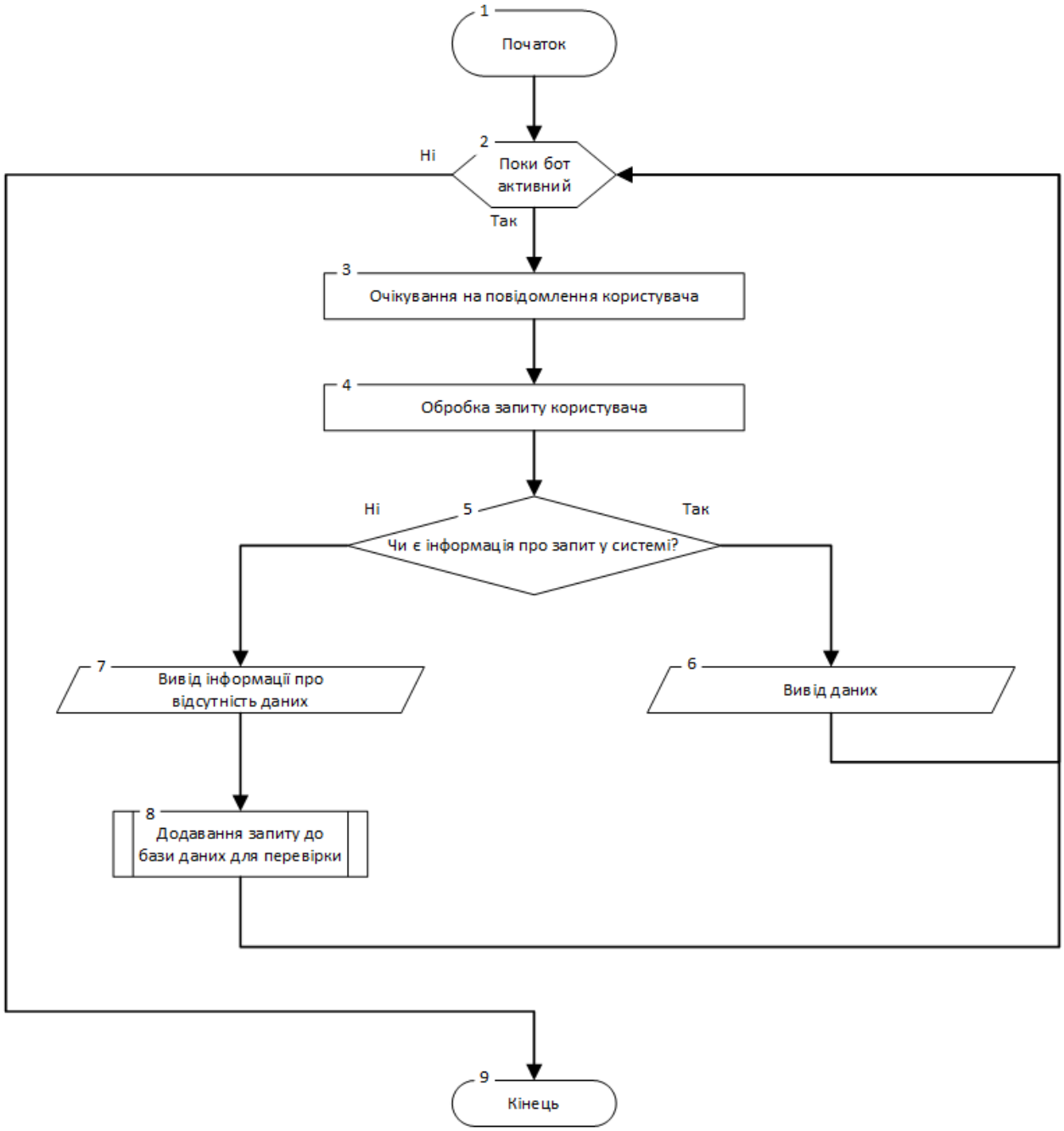

Рисунок 2.2 – Блок-схема роботи алгоритму телеграм-бота

Блок-схема описує наступний алгоритм:

1. «Початок»: цей блок описує початок роботи з програмою, користувач запускає програму.

2. «Поки бот активний»: перевіряється, чи активний бот.

3. «Очікування на повідомлення користувача»: цей блок описує, що програма очікує на введення користувача, користувач може ввести будь-який текст або команду.

4. «Обробка запиту користувача»: цей блок описує, що програма обробляє введений користувачем текст або команду, бот може виконувати різні дії залежно від введеного тексту або команди.

5. «Чи є інформація про запит у системі?»: цей блок описує, що програма перевіряє, чи є інформація про запит користувача в системі. Якщо інформація доступна, програма переходить до блоку 6. Якщо інформація недоступна, програма переходить до блоку 7.

6. «Вивід інформації»: цей блок описує, що програма виводить інформацію про запит користувача. Інформація може бути у вигляді тексту або аудіофайлу.

7. «Вивід інформації про відсутність даних»: цей блок описує, що програма виводить інформацію про те, що запит користувача не знайдено в системі. Користувачеві буде запропоновано спробувати ще раз або ввести інший запит.

8. «Додавання запиту до бази даних для перевірки»: цей блок описує, що застосунок додає запит користувача до бази даних для перевірки. Це може бути корисно для того, щоб застосунок міг навчатися на нових запитах і покращувати свою роботу з часом.

9. «Кінець»: цей блок описує завершення роботи з програмою. Користувач може вийти з бота або продовжити роботу з ним.

Розглянувши вищенаведений алгоритм, можна зрозуміти, що розроблений програмний застосунок матиме надійний алгоритм, що дозволить надійно реєструвати запити користувачів та зрозуміло давати інформацію після обробки.

2.5 Висновки

У другому розділі було проведено аналіз вхідних та вихідних даних програмного застосунку. Було проведено аналіз вимог до UI/UX дизайну програмного застосунку. Також було розроблено основний алгоритм роботи телеграм-бота та модель роботи програмного продукту за допомогою UML діаграм.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ ЗАСТОСУНКУ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Для розробки цього програмного застосунку для обробки музичного контенту з використанням штучного інтелекту, було необхідно вибрати сучасні та стабільні інструменти розробки. Тому було вибрано Python з метою забезпечення надійності та ефективності.

Python – це мова програмування загального призначення, яка набула широкої популярності завдяки своїй простоті, універсальності та потужності [14]. Ось чому було обрано саме цю мову програмування:

1. Простота: Python має чіткий і лаконічний синтаксис, схожий на англійську мову, що робить його легким для вивчення та читання, навіть для початківців. Завдяки мінімальній кількості ключових слів та спеціальних символів код Python легко зрозуміти та модифікувати.

2. Універсальність: Python можна використовувати для широкого кола завдань, включаючи веброзробку, машинне навчання, аналітику даних, роботу з штучним інтелектом, системне адміністрування та багато іншого. Існує безліч бібліотек та фреймворків Python, які надають готові рішення для різних задач, що робить Python ще більш універсальним.

3. Потужність: Python – це потужна мова програмування, яка здатна реалізовувати складні проєкти. Вона має динамічну систему типів, що дозволяє гнучко працювати з даними. Python підтримує об'єктно-орієнтоване програмування, що робить його зручним для розробки складних програмних систем.

4. Продуктивність: Python – це інтерпретована мова, що означає, що код не потрібно компілювати перед запуском. Це робить розробку швидшою та економнішою. Python також має високу продуктивність завдяки своїй вбудованій віртуальній машині та JIT-компілятору.

5. Спільнота: Python має велику та активну спільноту розробників, які

завжди готові допомогти та поділитися своїми знаннями. Існує безліч онлайн-ресурсів, таких як документація, форуми та навчальні посібники для вивчення мови програмування.

Застосунок було розроблено у середовищі PyCharm [15]. PyCharm – це інтегроване середовище розробки (IDE), спеціально розроблене для мови програмування Python. Він пропонує широкий спектр функцій, які роблять його незамінним інструментом для розробників Python будь-якого рівня, від початківців до професіоналів. Ось деякі з ключових особливостей PyCharm:

1. Інтелектуальне автозаповнення коду: PyCharm може передбачати, що користувач збирається написати, і пропонує підказки та фрагменти коду, що економить час та допомагає уникати помилок.

2. Аналіз коду: PyCharm постійно аналізує код на наявність помилок, попереджень та потенційних проблем, допомагаючи писати чистий та надійний код.

3. Відлагодження: PyCharm пропонує потужні інструменти для відлагодження, які дозволяють крок за кроком виконувати ваш код, переглядати змінні та виявляти проблеми.

4. Рефакторинг: PyCharm пропонує широкий спектр функцій рефакторингу, які дозволяють змінювати структуру коду без зміни його поведінки.

5. Інтеграція з системами контролю версій: PyCharm підтримує популярні системи контролю версій, такі як Git і Mercurial, що полегшує спільну роботу над проєктами.

6. Розширюваність: PyCharm можна розширювати за допомогою плагінів, які додають нові функції та можливості.

Отже, вибір цих технологій є оптимальним для реалізації функціональності програмного застосунку та забезпечить його успішну розробку, впровадження та подальшу підтримку. Python надає весь необхідний функціонал для розробки телеграм-бота.

3.2 Розробка інтерфейсу програмного застосунку

Розробка інтерфейсу є дуже важливою частиною створення бота для обробки музичного контенту, оскільки вона визначає спосіб, яким користувачі будуть взаємодіяти з програмним застосунком. Гарно спроєктований та інтуїтивно зрозумілий інтерфейс допомагає забезпечити приємний та ефективний досвід користувача, підвищуючи ймовірність того, що користувачі будуть продовжувати використовувати застосунок [16].

Оскільки бот розроблюється у Телеграмі, лише користувачі можуть впливати на кольори у розроблюваному застосунку, адже колір інтерфейсу в даному месенджері можна обирати самостійно (рис. 3.1).

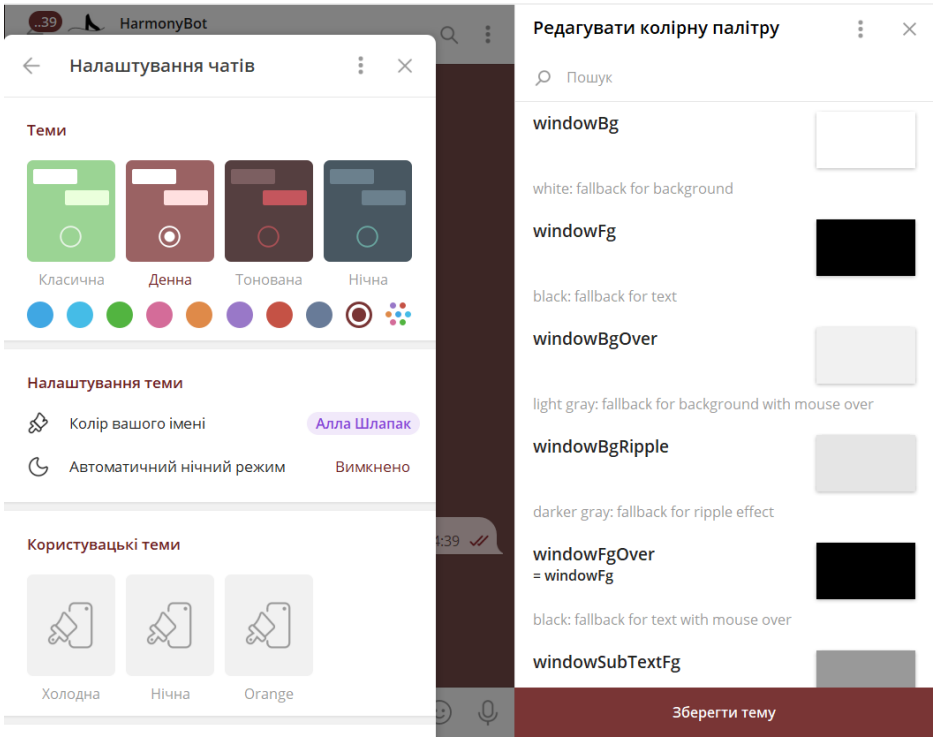

Рисунок 3.1 – Кастомізація кольорової палітри застосунку Telegram

При запуску бота, користувач бачить привітання та основний запит бота – надати дані пісні для обробки (рис. 3.2). Запит потрібно надавати чітко як вказано у повідомленні, щоб здійснити пошук у базі даних для швидкої перевірки наявного музичного твору в ній.

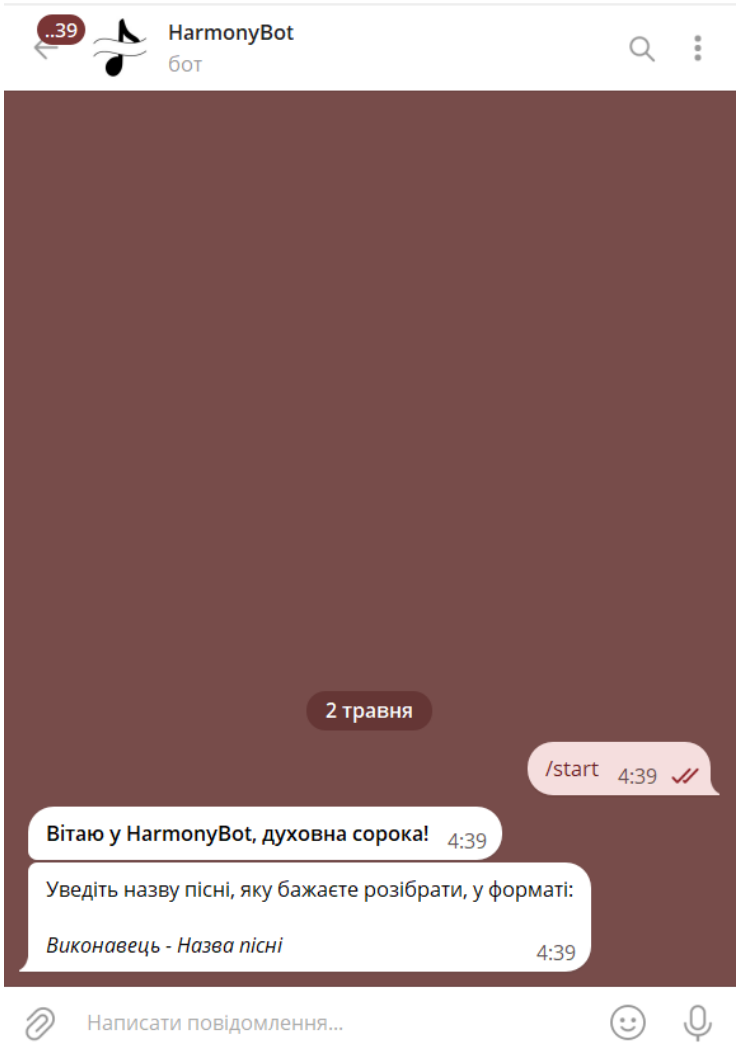

Рисунок 3.2 – Чат з ботом після запуску

Після введення назви пісні, якщо вона уже збережена у базі даних, користувач отримує три музичні файли: оригінальний аудіозапис, «vocals» – файл з вокалом, та «music» – файл з інструменталом (рис. 3.3). Це означає, що користувач має доступ до окремих компонентів музичного треку, що може бути корисним для різних цілей, таких як створення реміксів, караоке або аналізу музичної структури. Важливо зазначити, що успіх пошуку не залежить від регістру літер, написаних в назві. Це означає, що користувач може вводити назву пісні у верхньому, нижньому регістрі або змішаному, і система все одно зможе знайти відповідну інформацію в базі даних.

Ця система також дозволяє підвищити рівень задоволеності користувачів, оскільки надає їм можливість отримати доступ до потрібних музичних файлів з

мінімальними зусиллями. Це є важливою частиною загального користувацького досвіду, який впливає на те, наскільки користувачі будуть задоволені роботою з програмним забезпеченням і наскільки часто вони будуть використовувати його в майбутньому. Таким чином, ця функція є важливим елементом, який допомагає забезпечити позитивний досвід використання та сприяє успішному використанню програмного застосунку.

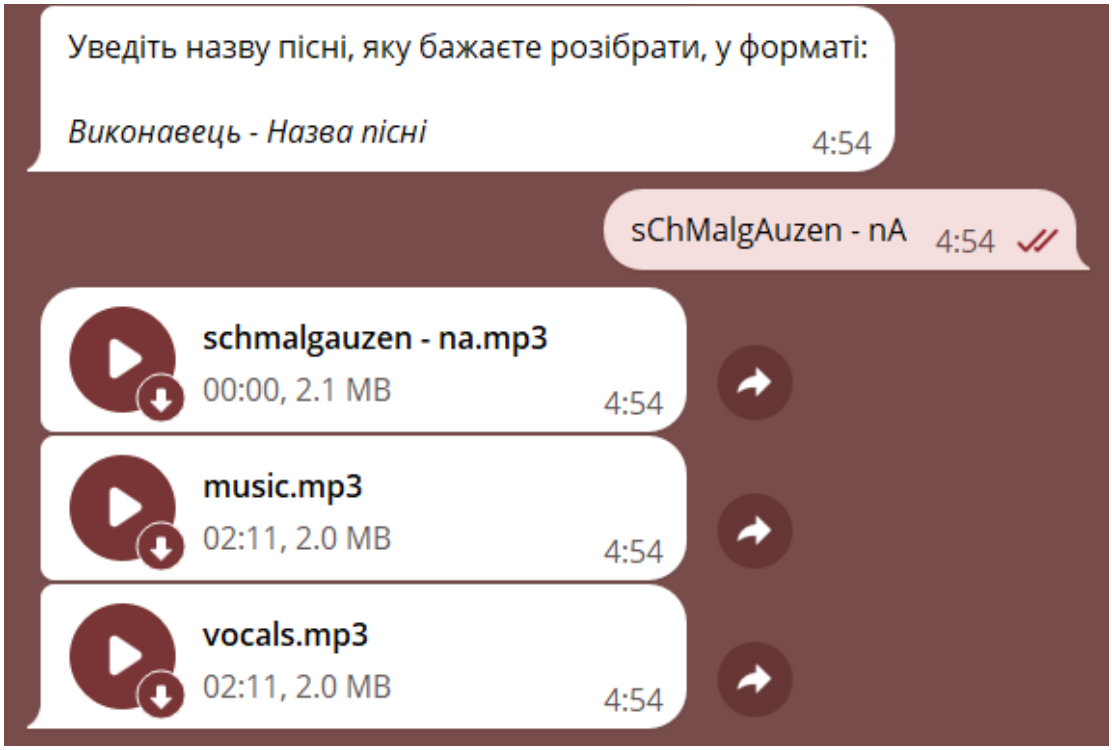

Рисунок 3.3 – Чат з ботом після надсилання запиту на знаходження пісні

У випадку, якщо пісні немає в базі, користувачу прийде повідомлення, що йому варто надіслати аудіофайл для розбиття (рис. 3.4).

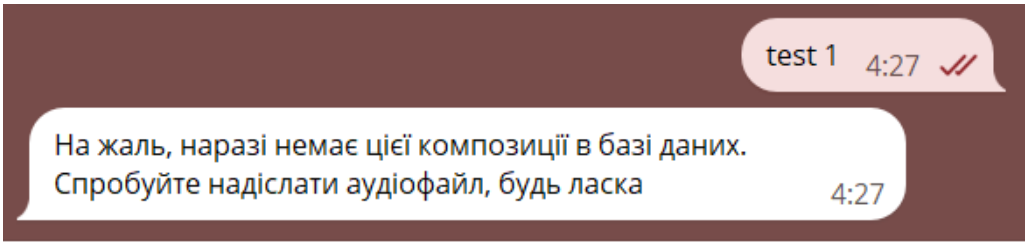

Рисунок 3.4 – Чат з ботом після надсилання запиту на обробку пісні, якої немає в базі даних

Для роботи телеграм-бота важливі команди для розуміння роботи цього бота. Було розроблено команди «/help» та «/info» для отримання додаткової інформації про розробку. Команда «/help» надає користувачам детальні інструкції щодо використання бота, включаючи пояснення основних функцій, доступних команд та приклади їх використання. Команда «/info» надає додаткову інформацію про проєкт, включаючи його цілі, розробників, поточний статус і майбутні оновлення.

Використання цих команд забезпечує кращу взаємодію користувачів з ботом і сприяє їхньому залученню до проєкту (рис. 3.5).

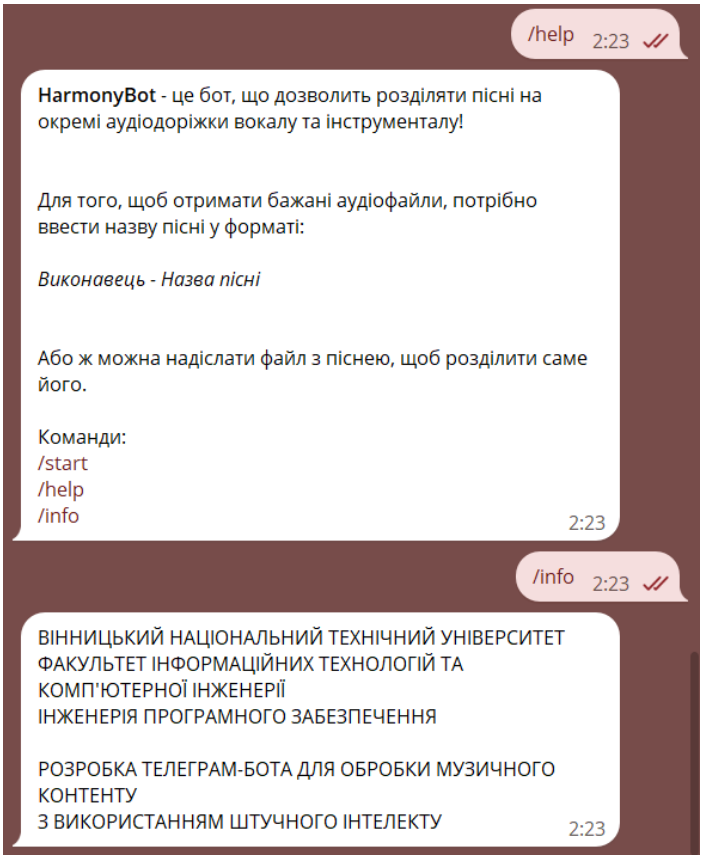

Рисунок 3.5 – Чат з ботом після використання команд «/help» та «/info»

Було додано стилі текстових повідомлень за допомогою вставок HTML [17]. Саме вони дозволяють тексту набувати різних форм, таких як жирний шрифт, курсив, підкреслений текст та інші. Це покращує читабельність та візуальну привабливість повідомлень, дозволяючи користувачам легше

сприймати важливу інформацію. Використання HTML-стилів також дає змогу виділяти ключові моменти та структурувати інформацію більш ефективно, що сприяє кращій взаємодії користувача з ботом (рис. 3.6).

Рисунок 3.6 – HTML-теги для форматування тексту в телеграм-ботах

При надсиланні аудіофайлу, для додаткової перевірки бот запитує назву пісні. Якщо вона вже наявна, завантажує збережені файли (рис. 3.7). Це також дозволить створити менше навантаження на систему бота.

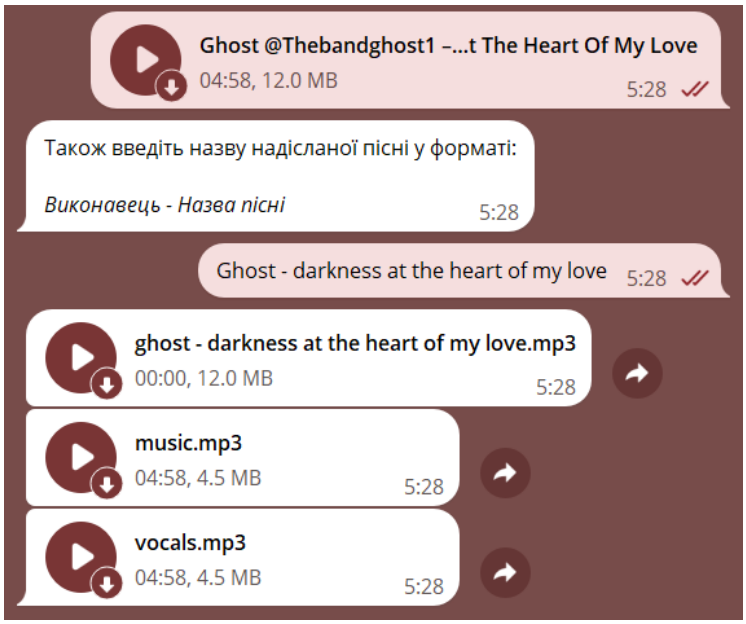

Рисунок 3.7 – Чат з ботом після надсилання аудіофайлу

Також було створено унікальний логотип-аватар для бота, щоб він був впізнаваним – розділена нота, що підходить до тематики та функціоналу бота. Логотип є важливим елементом брендування, який допомагає користувачам миттєво асоціювати бота з його призначенням та функціями (рис. 3.8).

Рисунок 3.8 – Логотип бота

У вікні запуску бота було додано короткий опис – яким функціоналом забезпечує бот користувача (рис. 3.9).

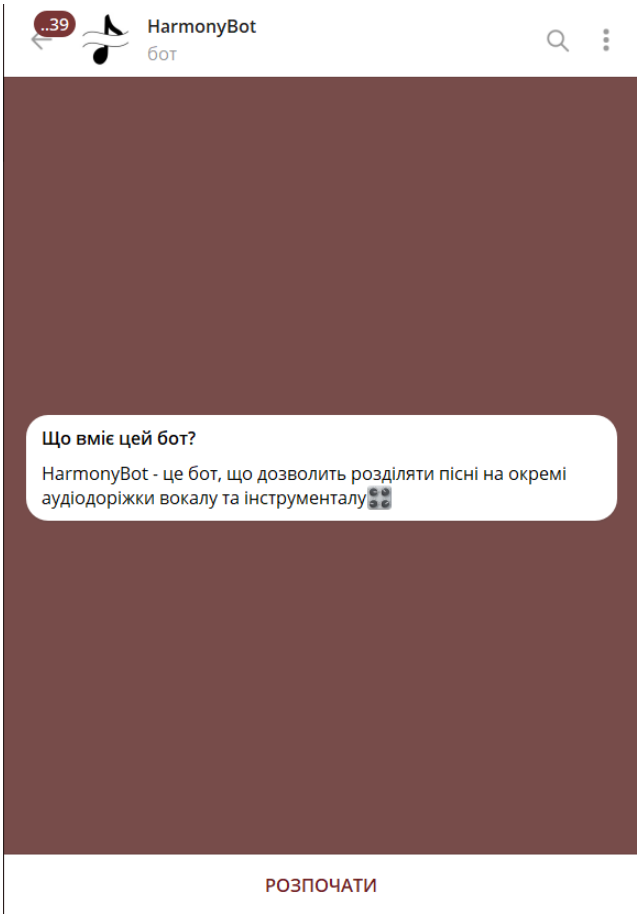

Рисунок 3.9 – Вікно запуску бота

Інформація про бот, яку можна отримати, перейшовши в його профіль, зображена на рисунку 3.10.

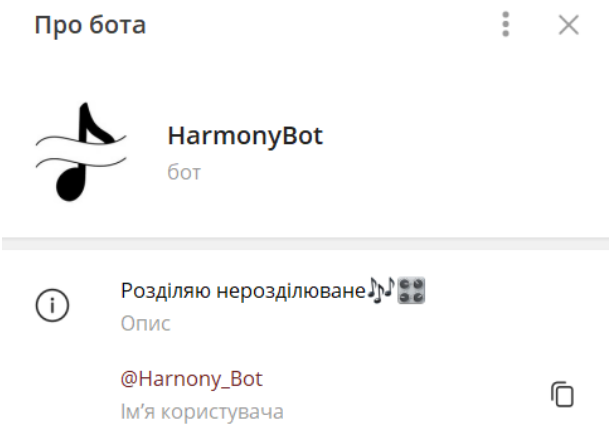

Рисунок 3.10 – Вікно «Про бота»

Розробка графічного інтерфейсу пройшла успішно, завдяки використанню різноманітних інструментів та технологій. За допомогою мови програмування Python та мови розмітки HTML, було створено ефективний та зручний інтерфейс, який забезпечує комфортну взаємодію користувача з програмою. Одним із ключових компонентів інтерфейсу стали команди та створений функціонал бота, забезпечений сервісом Telegram [18].

Інтерфейс бота також включає можливість швидкого доступу до історії оброблених пісень, що дозволяє користувачам легко знаходити та повторно використовувати вже оброблені композиції.

Завдяки ретельному підходу до розробки інтерфейсу, бот став зручним та зрозумілим для користувачів різного рівня підготовки. Це сприяло підвищенню його популярності та збільшенню кількості активних користувачів, що використовують його для обробки музичного контенту. Багатофункціональний інтерфейс, поєднаний із алгоритмами штучного інтелекту, забезпечує високу якість роботи та задоволення потреб користувачів.

3.3 Розробка програмного модулю обробки запитів користувача

Головною задачею програмного застосунку для обробки музичного

контенту з використанням штучного інтелекту є забезпечення зручного та надійного сервісу для розділення аудіофайлів на звукові доріжки за вокалом та інструменталом, тому цей алгоритм є важливою складовою програмного застосунку. Спочатку було розроблено сам власне телеграм-бот за допомогою чат-бота BotFather (рис. 3.11).

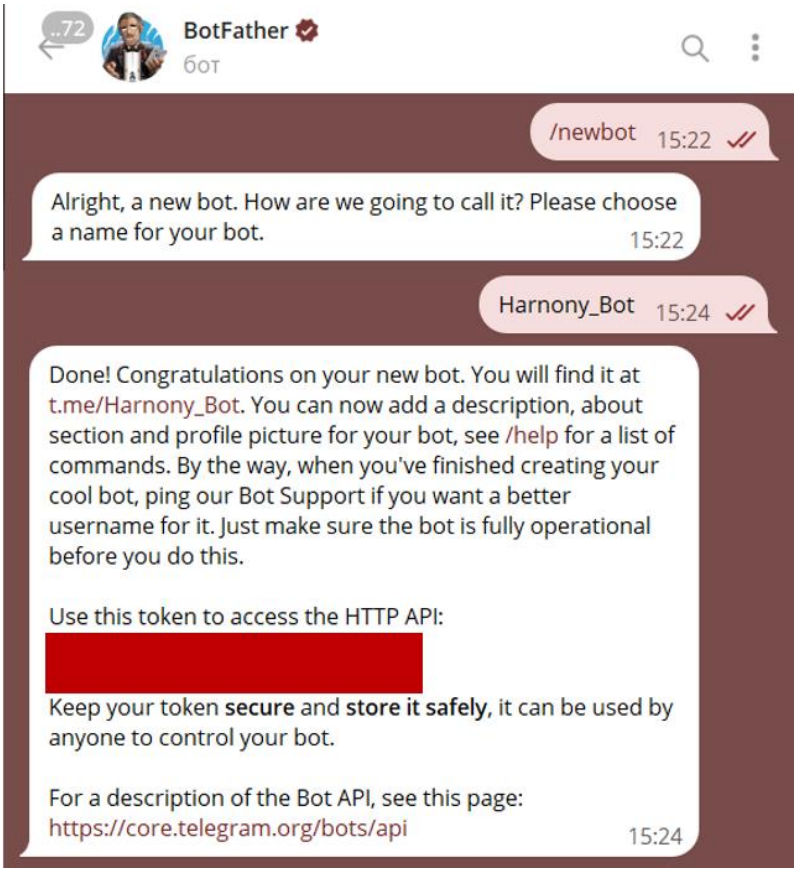

Рисунок 3.11 – Створення бота у BotFather

Після цього бота було імпортовано в проєкт. Для цього було застосовано бібліотеку «telebot» (рис. 3.12).

```
import telebot
import os
import datetime

bot = telebot.TeleBot('token')
```

Рисунок 3.12 – Імпорт бота в код проєкту

Метод start_handler() описує дії бота після запуску команди «/start». Метод створює об’єкт User з елементом chat_id в базу даних для збереження дій користувача. Також надсилає привітання та першу інформацію для користувача з описом роботи бота. Функція start_handler виконує важливу роль у знайомстві нових користувачів з HarmonyBot, вітаючи їх, створюючи записи в базі даних та надаючи чіткі інструкції щодо використання бота. Це сприяє кращому досвіду користувачів та гарантує, що вони розуміють, як використовувати бота для розбиття пісень на звукові доріжки. Метод start_handler() продемонстровано на рисунку 3.13.

```python
# command /start


@bot.message_handler(commands=['start'])
def start_handler(message):
    if not User.select().where(User.chat_id == message.chat.id):
        User.create(
            chat_id=message.chat.id
        )

    bot.send_message(message.chat.id,
                     f"<b>Вітаю у HarmonyBot, {message.chat.first_name + ((' ' + message.chat.last_name) or '')}!</b>",
                     parse_mode='HTML')
    bot.send_message(message.chat.id, "Уведіть назву пісні, яку бажаєте розібрати, у форматі:"
                                      "\n\n<i>Виконавець - Назва пісні</i>", parse_mode='HTML')
```

Рисунок 3.13 – Метод start_handler()

Метод help_handler() описує дії бота після запуску команди «/help». Функція help_handler відіграє важливу роль у наданні користувачам інформації про HarmonyBot та його можливості. Вона чітко описує, як розділяти пісні на звукові доріжки, пояснює формат введення назви пісні, інформує про можливість надіслання файлу та надає список доступних команд. Метод help_handler() продемонстровано на рисунку 3.14.

```python
# command /help


@bot.message_handler(commands=['help'])
def help_handler(message):
    bot.send_message(message.chat.id,
                     f"<b>HarmonyBot</b> - це бот, що дозволить розділяти пісні на окремі аудіодоріжки вокалу та інструменталу!"
                     f"\n\n\nДля того, щоб отримати бажані аудіофайли, потрібно ввести назву пісні у форматі:"
                     "\n\n<i>Виконавець - Назва пісні</i>\n\n\nАбо ж можна надіслати файл з піснею, щоб розділити саме його."
                     "\n\nКоманди:\n/start\n/help\n/info", parse_mode='HTML')
```

Рисунок 3.14 – Метод help_handler()

Метод info_handler() описує дії бота після запуску команди «/info». Функція info_handler надає користувачам чітку та лаконічну інформацію про проєкт бота HarmonyBot, його розробників та його основну функціональність. Це допомагає користувачам краще зрозуміти контекст роботи бота та його цінність. Метод info_handler() продемонстровано на рисунку 3.15.

```
# command /info


@bot.message_handler(commands=['info'])
def info_handler(message):
    bot.send_message(message.chat.id,
                     f"Вінницький національний технічний університет"
                     f"\nфакультет інформаційних технологій та комп'ютерної інженерії"
                     f"\nінженерія програмного забезпечення\n\nРозробка телеграм-бота для обробки музичного
                     parse_mode='HTML')
```

Рисунок 3.15 – Метод info_handler()

Метод set_song_name_handler() описує дії бота після введення тексту користувачем. Функція set_song_name_handler обробляє текстові повідомлення та реагує на відсутність пісні в базі даних HarmonyBot. Вона інформує користувача, пропонує надіслати файл, зберігає інформацію про пошук, а у випадку знаходження раніше завантаженої пісні надсилає користувачеві усі її частини (повний файл, інструментал, вокал). Цей метод створює об’єкт Harmony з елементами singName, is_existent, date і user в базу даних для збереження дій користувача. Після цього метод надсилає інформацію для користувача з обробленими аудіофайлами. Метод set_song_name_handler() продемонстровано на рисунку 3.16.

Метод set_song_handler() описує дії бота після надсилання аудіофайлу користувачем. Цей метод ініціює передачу інформації щодо обробки запиту, доповнюючи передані дані від користувача. Коли користувач надсилає аудіоповідомлення, бот HarmonyBot надсилає йому відповідь з інструкціями щодо форматування назви пісні.

Бот може використовувати цю інформацію для пошуку та обробки пісні

для користувача. Метод set_song_handler() продемонстровано на рисунку 3.17.

```
# send the song name


@bot.message_handler(content_types=['text'])
def set_song_name_handler(message):
    try:
        os.mkdir(f'{dirname}/music/{message.text.lower()}')
        bot.send_message(message.chat.id, "На жаль, наразі немає цієї композиції в базі даних. Спробуйте надіслати аудіофайл, будь ласка")
        os.rmdir(f'{dirname}/music/{message.text.lower()}')
        Harmony.create(
            songName=message.text.lower(),
            is_existent=False,
            user=User.select().where(User.chat_id == message.chat.id),
            date=datetime.date.today()
        )
    except FileExistsError:
        bot.send_audio(message.chat.id, open(f'{dirname}/music/{message.text.lower()}/{message.text.lower()}.mp3', 'rb'))
        bot.send_audio(message.chat.id, open(f'{dirname}/music/{message.text.lower()}/music.mp3', 'rb'))
        bot.send_audio(message.chat.id, open(f'{dirname}/music/{message.text.lower()}/vocals.mp3', 'rb'))
```

Рисунок 3.16 – Метод set_song_name _handler()

Користувач повинен потім надіслати текстове повідомлення з назвою пісні у форматі "Виконавець – Назва пісні".

```
# send audio


@bot.message_handler(content_types=['audio'])
def set_song_handler(message):
    bot.send_message(message.chat.id, "Також введіть назву надісланої пісні у форматі:"
                                      "\n\n<i>Виконавець - Назва пісні</i>", parse_mode='HTML')
```

Рисунок 3.17 – Метод set_song_handler()

Метод to_stereo() описує дії бота для конвертації аудіофайлу, надісланого користувачем. Функція to_stereo() приймає на вхід NumPy-масив, що представляє аудіосигнал, і гарантує, що на виході він матиме два канали (стерео).

Вхідні дані: waveform: це numpy-масив, що представляє аудіодані. він може мати різні форми залежно від кількості каналів:

– моно: форма (n, 1), де n – це кількість зразків у формі сигналу, а 1 – це один канал;

– стерео або багатоканальний: форма (n, d), де n – це кількість зразків, а d – це кількість каналів (наприклад, 2 для стерео, 5 для об'ємного звуку).

Вихідні дані: функція завжди повертає NumPy-масив з формою (N, 2), що представляє стереосигнал з двома каналами.

Функціональність:

1. Перевірка кількості каналів: Спочатку функція перевіряє кількість каналів у вхідному сигналі за допомогою waveform.shape.

2. Перетворення моно в стерео: Якщо сигнал має лише один канал (моно), функція дублює канал за допомогою np.repeat(waveform, 2, axis=-1). Це створює копію одного каналу та розміщує її поруч з ним, фактично створюючи стереосигнал з однаковими лівим і правим каналами.

3. Обрізка багатоканального аудіо: Якщо сигнал має більше двох каналів (об'ємний звук або інші конфігурації), функція обрізає його до перших двох каналів за допомогою waveform[:, :2]. Це зберігає перші два канали (які вважаються лівим і правим) і відкидає всі інші канали.

4. Повернення стереосигналу: Якщо сигнал вже має два канали (стерео), функція просто повертає оригінальний сигнал без будь-яких змін.

Метод to_stereo() продемонстровано на рисунку 3.18.

```python
def to_stereo(waveform: np.ndarray) -> np.ndarray:
    """
    Convert a waveform to stereo by duplicating if mono, or truncating
    if too many channels.

    Parameters:
        waveform (np.ndarray):
            a `(N, d)` numpy array.

    Returns:
        np.ndarray:
            A stereo waveform as a `(N, 1)` numpy array.
    """
    if waveform.shape[1] == 1:
        return np.repeat(waveform, 2, axis=-1)
    if waveform.shape[1] > 2:
        return waveform[:, :2]
    return waveform
```

Рисунок 3.18 – Метод to_stereo()

Ця функція гарантує, що вхідний аудіосигнал буде перетворено у формат з двома каналами (стерео), незалежно від його початкової конфігурації каналів. Вона належним чином обробляє моно- та багатоканальний аудіосигнал для забезпечення узгодженого стереовиходу

3.4 Розробка системи збереження оброблених аудіофайлів за допомогою бази даних

Основна задача розробки системи взаємодії з базою даних полягає в створенні бази даних для збереження надісланої інформації від користувачів та музичних файлів. Ця система повинна забезпечити можливість зчитування, запису та оновлення даних в базі даних з врахуванням потреб користувачів та обмежень безпеки. Вона також має забезпечити оптимальний рівень продуктивності та надійності, щоб забезпечити ефективну роботу системи в будь-який час і в умовах великої кількості користувачів та обсягу даних.

Для цього було використано ORM на Python – Peewee [19]. Ось деякі причини, чому Peewee є популярним вибором для розробників Python, які працюють з базами даних:

1. Простота використання: Peewee має інтуїтивно зрозумілий API, що робить його легким для освоєння навіть початківцям.

2. Гнучкість: Peewee дозволяє вам працювати з базами даних на різних рівнях абстракції. Можливо використовувати його для простих запитів або складніших операцій з даними.

3. Потужність: незважаючи на простоту, Peewee пропонує широкий спектр функціональних можливостей, таких як визначення моделей даних, виконання запитів, міграції баз даних тощо.

4. Підтримка баз даних: Peewee підтримує різні популярні бази даних, що робить його універсальним рішенням для багатьох проектів.

Також було розроблено модуль models.py для створення об’єктів, які будуть використані для додавання даних до БД (рис. 3.19).

Цей код створює базову структуру для Telegram-бота, використовуючи

Peewee та SQLite-базу даних. Він визначає моделі для користувачів та пісень, встановлюючи зв'язок між ними. Сценарій також створює необхідні таблиці в базі даних.

```python
from peewee import SqliteDatabase, Model, CharField, DateField, ForeignKeyField, BooleanField

db = SqliteDatabase('bot.sqlite3')


# class User - for user`s data
class User(Model):
    chat_id = CharField()

    class Meta:
        database = db


# class Harmony - for song`s data
class Harmony(Model):
    songName = CharField()
    is_existent = BooleanField()
    user = ForeignKeyField(User)
    date = DateField()

    class Meta:
        database = db


if __name__ == '__main__':
    db.create_tables([User, Harmony])
```

Рисунок 3.19 – Модуль models.py із створеними об’єктами

Усі запити на пісні, які збережені у базі даних, будуть надіслані за запитом. Записи внесено у базу даних bot.sqlite3 (рис. 3.20).

4 rows | Tx: Auto | DDL

WHERE ORDER BY

	id	songName	is_existent	date	user_id
1	1	test1	0	2024-05-02	1
2	2	sadsvit - силуети	0	2024-05-03	1
3	3	lady gaga - just da…	0	2024-05-03	1
4	4	test2	0	2024-05-03	1

Рисунок 3.20 – База даних bot.sqlite3 із внесеними даними

Отже, всі запити на пісні, які зберігаються в базі даних bot.sqlite3, будуть надсилатися користувачам без створення надмірного навантаження на систему. Це означає, що:

1. Бот буде розумно керувати ресурсами, щоб не перевантажувати систему.

2. Користувачі зможуть отримувати запитувані пісні без затримок чи помилок.

3. Система буде масштабованою, щоб обробляти зростаючу кількість запитів.

Зберігання запитів у базі даних bot.sqlite3 має такі переваги:

1. Збереження історії: Це дозволяє боту відстежувати, які пісні запитували користувачі, навіть якщо вони ще не доступні.

2. Аналіз даних: Ці дані можна використовувати для аналізу популярних запитів та вдосконалення роботи бота.

3. Відновлення запитів: Якщо користувач втратить доступ до свого запиту, його можна відновити з бази даних.

Загалом, надсилання збережених запитів на пісні без створення надмірного навантаження на систему є важливою функцією, яка забезпечить плавну та ефективну роботу HarmonyBot.

3.5 Висновки

У третьому розділі було обґрунтовано вибір мови програмування та технологій для розробки системи. Проаналізувавши потреби програмного застосунку, було обрано мову програмування Python і середовище розробки PyCharm.

Після вибору мови програмування та середовища розробки було розроблено графічний інтерфейс програмного застосунку. Інтерфейс користувача був розроблений таким чином, щоб бути зручним та інтуїтивно зрозумілим. Він використовує елементи керування, які є загальновідомими та легкодоступними для користувачів.

Система складається з декількох програмних модулів, кожен з яких виконує певну функцію. Було описано роботу кожного модуля та його взаємодію з базою даних.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Суть тестування цього застосунку полягає в перевірці його функціональності та надійності через проведення різноманітних видів тестів. Це включає модульне тестування окремих компонентів застосунку, функціональне тестування його як єдиного цілого, перевірку коректності та ефективності роботи графічного інтерфейсу на Python, а також тестування сумісності та продуктивності на різних пристроях та в різних середовищах. Метою тестування є забезпечення високої якості програмного продукту, відповідності його функціональності вимогам користувачів та безпеки його роботи.

Модульне тестування – це процес перевірки окремих компонентів або модулів програмного забезпечення на коректність їх роботи в ізоляції від інших частин системи [20]. У контексті цього застосунку на Python, модульне тестування включає перевірку різних частин коду. Приклад модульного тесту продемонстровано на рисунку 4.1.

```python
import unittest
import numpy as np

class TestStereoConversion(unittest.TestCase):
    """Tests for the to_stereo function"""

def test_mono_to_stereo(self):
    """Tests if mono waveform is converted to stereo correctly"""
    mono_waveform = np.array([[1, 2, 3], [4, 5, 6]])
    stereo_waveform = to_stereo(mono_waveform)
    self.assertEqual(stereo_waveform.shape, (2, 2))
    self.assertTrue(np.array_equal(stereo_waveform, np.repeat(mono_waveform, 2, axis=-1)))

def test_stereo_waveform(self):
    """Tests if stereo waveform is returned unchanged"""
    stereo_waveform = np.array([[1, 2], [3, 4]])
    returned_waveform = to_stereo(stereo_waveform)
    self.assertEqual(returned_waveform.shape, stereo_waveform.shape)
    self.assertTrue(np.array_equal(returned_waveform, stereo_waveform))

def test_multichannel_to_stereo(self):
    """Tests if waveform with more than 2 channels is truncated correctly"""
    multichannel_waveform = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
    stereo_waveform = to_stereo(multichannel_waveform)
    self.assertEqual(stereo_waveform.shape, (2, 2))
    self.assertTrue(np.array_equal(stereo_waveform, multichannel_waveform[:, :2]))


if __name__ == "__main__":
    unittest.main()
```

Рисунок 4.1 – Модульний тест для функції to_stereo()

Цей тест перевіряє, чи правильно функція to_stereo() розбиває композицію на доріжки. Принцип роботи модулю:

1. Імпорт бібліотек: імпортовано unittest для тестування та numpy для роботи з масивами (припускаючи, що to_stereo використовує numpy).

2. Імпорт функції: імпортовано функцію to_stereo з відповідного модуля.

Тестові методи: test_mono_to_stereo: цей тест перевіряє, чи моно-хвиля правильно перетворюється в стерео шляхом дублювання каналів; test_stereo_waveform: цей тест перевіряє, чи стерео-хвиля повертається функцією без змін; test_multichannel_to_stereo: цей тест гарантує, що хвилі з більш ніж 2 каналами обрізаються до стерео шляхом збереження перших двох каналів.

Використано assertEqual для порівняння очікуваних та фактичних форм і assertTrue з np.array_equal для порівняння вмісту масивів. Основний блок: Цей блок гарантує, що тести запускаються лише під час безпосереднього виконання скрипту (не імпортуються як модуль).

Також було розроблено модульний тест на перевірку поведінки функції set_song_name_hendler в двох сценаріях: коли користувач надсилає нову назву пісні та коли надсилає назву вже доданої до бази пісні (рис. 4.2).

```python
def test_new_song_name(mocker):
    # Створити мок-об'єкт для повідомлення
    mock_message = MagicMock()
    mock_message.chat.id = 12345
    mock_message.text = "New Song Name"

    # Створити мок-об'єкт для моделі User
    mock_user_model = mocker.patch('harmonybot.models.User')
    mock_user = MagicMock()
    mock_user_model.select.return_value.get = mock_user

    # Створити мок-об'єкт для моделі Harmony
    mock_harmony_model = mocker.patch('harmonybot.models.Harmony')

    # Створити мок для os.mkdir
    mock_mkdir = mocker.patch('os.mkdir')

    # Створити мок для os.path.exists
    mock_exists = mocker.patch('os.path.exists')

    # Створити мок для datetime.date.today
    mock_today = mocker.patch('datetime.date.today')
    mock_today.return_value = datetime.date(2024, 5, 3)

    # Викликати функцію з мок-повідомленням
    set_song_name_handler(mock_message)
```

Рисунок 4.2 – Модульний тест на створення запиту користувача

Функція mock_mkdir.assert_called_once_with(): перевіряє, чи функція os.mkdir викликалася один раз із правильною адресою папки для нової пісні.

Функція bot_send_message_calls: перевіряє, чи функція bot.send_message викликалася один раз для надсилання повідомлення про відсутність файлу.

Функція mock_rmdir.assert_called_once_with(): перевіряє, чи функція os.rmdir викликалася один раз для видалення створеної папки. Функція mock_harmony_model.create.assert_called_once_with(...): перевіряє, чи функція Harmony.create

Після цього було проведено тестування інтерфейсу застосунку. Під час ручного тестування інтерфейсу перевіряється, як працює застосунок. Виконуються певні дії на сайті та перевіряться, чи працює все правильно. Якщо буде виявлено проблему, її записують щоб розробники могли виправити. Такий процес допомагає зробити продукт кращим і забезпечити задоволення користувачів.

Під час ручного спочатку перевіряється головна сторінка застосунку. Після введення даних користувачем, бот надсидає потрібну інформацію залежно від запиту. Важливо зазначити, що успіх пошуку не залежить від регістру літер, написаних в назві, тому, якщо пісня наявна і може бути надіслана, проблем не виникає (рис. 4.3).

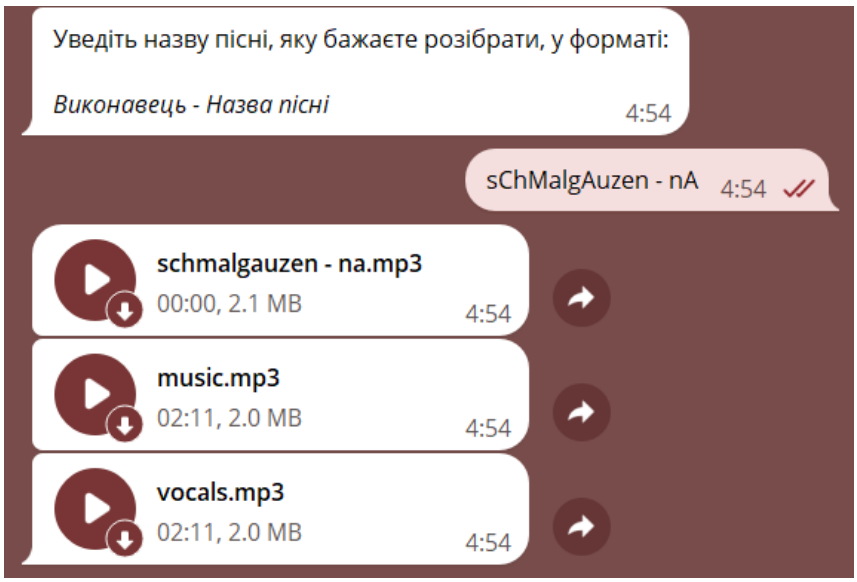

Рисунок 4.3 – Чат з ботом після надсилання запиту

Отже, тестування телеграм-бота включає оцінку функціональності, надійності та продуктивності програмного забезпечення. Виконані тести охоплювали модульне тестування для перевірки коректности роботи окремих методів та ручне тестування інтерфейсу користувача для перевірки у відповідності досвіду користувача задуму.

4.2 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту. Деталі щодо мінімальної та рекомендованої конфігурації серверного комп'ютера можна знайти в таблицях 4.1 та 4.2.

Таблиця 4.1 – Мінімальна конфігурація:

Тип процесора	2-ядерний процесор з частотою 1.5 ГГц
Об'єм оперативної пам'яті	4 ГБ для 64-разрядної системи
Місце на жорсткому диску	2 ГБ
Версія Telegram	Telegram 5.0

Таблиця 4.2 – Рекомендована конфігурація:

Тип процесора	4-ядерний процесор з частотою 2 ГГц
Об'єм оперативної пам'яті	8 ГБ для 64-разрядної системи
Місце на жорсткому диску	4 ГБ
Версія Telegram	Telegram 5.0

Після запуску телеграм-бота користувач може вільно переходити до виконання своїх запитів. Керування ботом відбувається за допомогою команд

«/start», «/info», «/help». Також користувачеві потрібно вводити текстові та аудіо-запити для роботи з ботом. У разі, якщо у користувача виникнуть труднощі з надсиланням запитів, за допомогою команди /help можна додатково ознайомитись із правилами користування ботом (рис. 4.4).

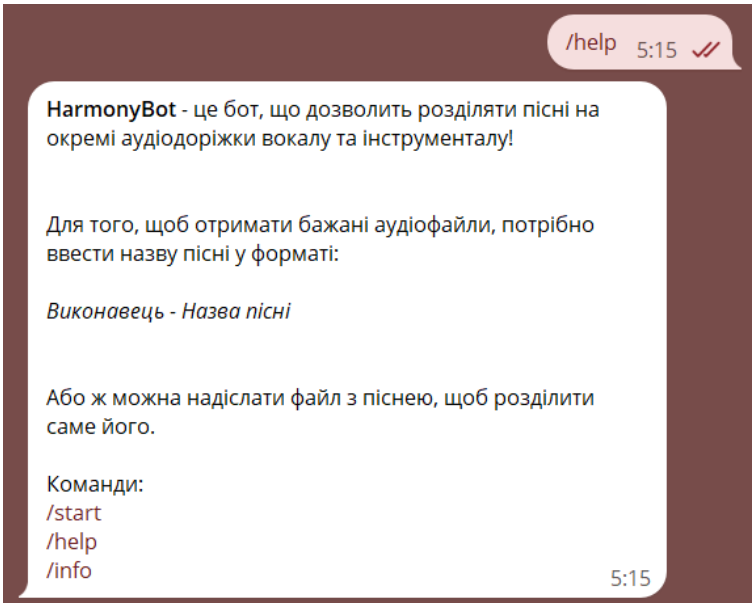

Рисунок 4.4 – Чат з ботом після використання команди «/help»

Підсумовуючи, застосунок дозволяє користувачеві обробляти музичний контент, що сприяє покращенню розуміння створення музики. Шляхом аналізу зібраних даних користувач може без проблем розділяти аудіофайли на звукові доріжки за вокалом та інструменталом.

4.3 Висновки

У четвертому розділі було проведено тестування програмних модулів програмного застосунку для обробки музичного контенту. Було було проведено ручне тестування інтерфейсу користувача та всіх можливостей бота. Було розроблено інструкцію користувача з початкової експлуатації програмного застосунку

ВИСНОВКИ

У цій бакалаврській дипломній роботі було розглянуто та досліджено ряд важливих аспектів розробки телеграм-бота для обробки музичного контенту з використанням штучного інтелекту. Основною метою було створення зручного та надійного сервісу для розділення аудіофайлів на звукові доріжки за вокалом та інструменталом з використанням можливостей штучного інтелекту.

Під час виконання бакалаврської дипломної роботи було проаналізовано стан проблеми на сьогоднішній день. Було розглянуто основні аналоги програмного застосунку. Було визначено їхні недоліки та порівняно з власним програмним застосунком. Базуючись на порівнянні було встановлено основні задачі бакалаврської дипломної роботи.

Під час аналізу технологій розробки було обґрунтовано вибір мови програмування Python у середовищі розробки PyCharm.

У бакалаврській дипломній роботі було зроблено наступне:

– розроблено модуль для обробки та зберігання оброблених файлів користувачів за допомогою штучного інтелекту;

– розроблено графічний інтерфейсу бота, який буде зручним та адаптивним для користувачів різного рівня;

– інтегровано різні модулі в єдиний програмний застосунок.

Було розроблено схему загального алгоритму роботи бота для обробки музичного контенту. Також було розроблено модель системи з використанням UML діаграм.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню. Також було розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Music industry revenue worldwide [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/259979/global-music-industry-revenue/ (дата звернення: 10.03.2024).

2. Best music-making apps in 2024 [Електронний ресурс] – Режим доступу до ресурсу: https://weraveyou.com/tech/best-music-making-apps-2023/ (дата звернення: 10.03.2024).

3. The Best Free Music Production Software for Beginners [Електронний ресурс] – Режим доступу до ресурсу: https://www.makeuseof.com/tag/free-music-production-software/ (дата звернення: 10.03.2024).

4. Шлапак А.В. РОЗРОБКА ТЕЛЕГРАМ-БОТА ДЛЯ ОБРОБКИ МУЗИЧНОГО КОНТЕНТУ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ / А. В. Шлапак Г. О. Черноволик// Матеріали конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)» , Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21153.

5. Music and Artificial Intelligence [Електронний ресурс] – Режим доступу до ресурсу: https://link.springer.com/chapter/10.1007/978-3-030-06170-8_16 (дата звернення: 10.03.2024).

6. Tiia Ketonen. Independent artists in the music industry – Bachelor Thesis – 2018 – 60 ст.

7. Як штучний інтелект створює музику і змінює креативну індустрію [Електронний ресурс] – Режим доступу до ресурсу: https://blog.depositphotos.com/ua/yak-shtuchnyj-intelekt-stvoryuye-muzyku.html (дата звернення: 10.03.2024).

8. Роль музики у вихованні особистості [Електронний ресурс] – Режим доступу до ресурсу: https://www.creativeschool.com.ua/blog/rol-muzyky-u-vyhovanni-osobystosti/ (дата звернення: 13.03.2024).

9. Вебсайт Phonic [Електронний ресурс] – Режим доступу до ресурсу: https://www.phonic.ai/ (дата звернення: 14.03.2024).

10. Вебсайт Lalal.ai [Електронний ресурс] – Режим доступу до ресурсу: : https://www.lalal.ai/ (дата звернення: 14.03.2024).

11. Застосунок Audacity [Електронний ресурс] – Режим доступу до ресурсу: https://www.audacityteam.org/ (дата звернення: 15.03.2024).

12. Застосунок FL Studio [Електронний ресурс] – Режим доступу до ресурсу: https://www.image-line.com/ (дата звернення: 15.03.2024).

13. UI vs. UX Design: What's the Difference? [Електронний ресурс] – Режим доступу до ресурсу: https://www.coursera.org/articles/ui-vs-ux-design (дата звернення: 12.04.2024).

14. What are UML diagrams, and how can you use them? [Електронний ресурс] – Режим доступу до ресурсу: https://miro.com/blog/uml-diagram/ (дата звернення: 12.04.2024).

15. Інтерактивний навчальний посібник Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.learnpython.org/ (дата звернення: 12.04.2024).

16. Середовище розробки PyCharm [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/pycharm/ (дата звернення: 12.04.2024).

17. Історія розвитку User Interface [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/user-interface-tse/ (дата звернення: 25.03.2024).

18. Telegram Text Formatting: Tips, Font Tricks, and Shortcuts [Електронний ресурс] – Режим доступу до ресурсу: https://sendpulse.com/blog/telegram-text-formatting (дата звернення: 29.03.2024).

19. Telegram Home page [Електронний ресурс] – Режим доступу до ресурсу: https://telegram.org/ (дата звернення: 29.03.2024).

20. Peewee documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.peewee-orm.com/en/latest/peewee/query_operators.html (дата звернення: 17.04.2024).

21. Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship – Pearson – 2008 – 464 ст.

ДОДАТКИ

ДОДАТОК А – ТЕХНІЧНЕ ЗАВДАННЯ

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 12 » березня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на бакалаврську дипломну роботу «Розробка телеграм-бота для обробки музичного контенту з використанням штучного інтелекту» за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
к.т.н, доц. каф. ПЗ Черноволик Г.О.
«12» березня 2024р.
Виконала:
студентка гр. 4ПІ-20Б Шлапак А.В.
«12» березня 2024р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка телеграм-бота для обробки музичного контенту з використанням штучного інтелекту».

Галузь застосування – сфера музики в професійному та особистому використанні.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від «11» березня 2024 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Мета програмного застосунку для обробки музичного контену полягає в забезпеченні зручного та надійного сервісу для розділення аудіофайлів на звукові доріжки за вокалом та інструменталом з використанням можливостей штучного інтелекту.

Відповідно до мети роботи необхідно виконати наступні завдання:

– визначити вимоги до програмного застосунку;

– розробити модуль для обробки та зберігання оброблених файлів користувачів за допомогою штучного інтелекту;

– розробити графічний інтерфейс бота, який буде зручним та адаптивним для користувачів різного рівня;

– інтегрувати різні модулі в єдиний програмний застосунок, що може бути використаний для обробки музичного контенту;

– провести тестування програмного застосунку.

4. Вихідні дані для проведення БДР

Перелік основних літературних джерел, на основі яких буде виконуватись БДР.

5. Технічні вимоги

Середовище розробки – PyCharm, мова розробки – Python, операційна система – Windows.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БДР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз стану питання та обґрутнування завдання	14.03.2024 – 29.03.2024
2	Аналіз вхідних та вихідних даних системи	30.03.2024 – 07.04.2024
3	Розробка моделі та алгоритмів системи	08.04.2024 – 26.04.2024
4	Розробка інтерфейсу	27.04.2024 – 04.05.2024
5	Розробка програмних модулів системи	05.05.2024 – 14.05.2024
6	Тестування системи	15.05.2024– 20.05.2024
7	Оформлення матеріалів до захисту БДР	21.05.2024 – 30.05.2024

10. Порядок контролю та прийняття

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

ДОДАТОК Б – ПЕРЕВІРКА НА ПЛАГІАТ

ПРОТОКОЛ
ПЕРЕВІРКИ БАКАЛАВРСЬКОЇ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка телеграм-бота для обробки музичного контенту з використанням штучного інтелекту

Тип роботи: БДР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Черноволик Г.О.

Оригінальність	88,33%
Схожість	11,67%

Аналіз звіту подібності

▪ **Запозичення, виявлені у роботі, оформленні коректно і не містять ознак плагіату.**

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цілісності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховання недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г.О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Шлапак А.В.

Керівник роботи Черноволик Г.О.

ДОДАТОК В – ЛІСТИНГ ПРОГРАМИ

```
import telebot
import os
import datetime

from models import User, Harmony

bot = telebot.TeleBot('7075836382:AAHnDBvuf3g--wuPdj9oTbnXA3mBRkUuvCo')
dirname = os.path.dirname(__file__)

# command /start

@bot.message_handler(commands=['start'])
def start_handler(message):
    if not User.select().where(User.chat_id == message.chat.id):
        User.create(
            chat_id=message.chat.id
        )

    bot.send_message(message.chat.id,
                     f"<b>Вітаю у HarmonyBot, {message.chat.first_name + ((' ' + message.chat.last_name) or
")}!</b>",
                     parse_mode='HTML')
    bot.send_message(message.chat.id, "Уведіть назву пісні, яку бажаєте розібрати, у форматі:"
                                      "\n\n<i>Виконавець - Назва пісні</i>", parse_mode='HTML')

# command /help

@bot.message_handler(commands=['help'])
def help_handler(message):
    bot.send_message(message.chat.id,
                     f"<b>HarmonyBot</b> - це бот, що дозволить розділяти пісні на окремі аудіодоріжки
вокалу та інструменталу!"
                     f"\n\n\nДля того, щоб отримати бажані аудіофайли, потрібно ввести назву пісні у
форматі:"
                     "\n\n<i>Виконавець - Назва пісні</i>\n\n\nАбо ж можна надіслати файл з піснею, щоб
розділити саме його."
                     "\n\nКоманди:\n/start\n/help\n/info", parse_mode='HTML')

# command /info

@bot.message_handler(commands=['info'])
def info_handler(message):
    bot.send_message(message.chat.id,
                     f"Вінницький національний технічний університет"
                     f"\nфакультет інформаційних технологій та комп'ютерної інженерії"
                     f"\nінженерія програмного забезпечення\n\nРозробка телеграм-бота для обробки
музичного контенту\nз використанням штучного інтелекту".upper(),
                     parse_mode='HTML')

# send the song name

@bot.message_handler(content_types=['text'])
def set_song_name_handler(message):
```

```
    try:
        os.mkdir(f'{dirname}/music/{message.text.lower()}')
        bot.send_message(message.chat.id, "На жаль, наразі немає цієї композиції в базі даних. Спробуйте
надіслати аудіофайл, будь ласа")
        os.rmdir(f'{dirname}/music/{message.text.lower()}')
        Harmony.create(
            songName=message.text.lower(),
            is_existent=False,
            user=User.select().where(User.chat_id == message.chat.id),
            date=datetime.date.today()
        )
    except FileExistsError:
        bot.send_audio(message.chat.id,
open(f'{dirname}/music/{message.text.lower()}/{message.text.lower()}.mp3', 'rb'))
        bot.send_audio(message.chat.id, open(f'{dirname}/music/{message.text.lower()}/music.mp3', 'rb'))
        bot.send_audio(message.chat.id, open(f'{dirname}/music/{message.text.lower()}/vocals.mp3', 'rb'))

# send audio

@bot.message_handler(content_types=['audio'])
def set_song_handler(message):
    bot.send_message(message.chat.id, "Також введіть назву надісланої пісні у форматі:"
                                      "\n\n<i>Виконавець - Назва пісні</i>", parse_mode='HTML')

bot.infinity_polling()


#!/usr/bin/env python
# coding: utf8

""" This module provides audio data convertion functions. """

# pyright: reportMissingImports=false
# pylint: disable=import-error
import numpy as np
import tensorflow as tf  # type: ignore

from ..utils.tensor import from_float32_to_uint8, from_uint8_to_float32

# pylint: enable=import-error

def to_n_channels(waveform: tf.Tensor, n_channels: int) -> tf.Tensor:
  """
  Convert a waveform to n_channels by removing or duplicating channels if
  needed (in tensorflow).

  Parameters:
    waveform (tf.Tensor):
      Waveform to transform.
    n_channels (int):
      Number of channel to reshape waveform in.

  Returns:
    tf.Tensor:
      Reshaped waveform.
  """
  return tf.cond(
```

```
    tf.shape(waveform)[1] >= n_channels,
    true_fn=lambda: waveform[:, :n_channels],
    false_fn=lambda: tf.tile(waveform, [1, n_channels])[:, :n_channels],
  )

def to_stereo(waveform: np.ndarray) -> np.ndarray:
  """
  Convert a waveform to stereo by duplicating if mono, or truncating
  if too many channels.

  Parameters:
    waveform (np.ndarray):
      a `(N, d)` numpy array.

  Returns:
    np.ndarray:
      A stereo waveform as a `(N, 1)` numpy array.
  """
  if waveform.shape[1] == 1:
    return np.repeat(waveform, 2, axis=-1)
  if waveform.shape[1] > 2:
    return waveform[:, :2]
  return waveform

def gain_to_db(tensor: tf.Tensor, espilon: float = 10e-10) -> tf.Tensor:
  """
  Convert from gain to decibel in tensorflow.

  Parameters:
    tensor (tf.Tensor):
      Tensor to convert
    epsilon (float):
      Operation constant.

  Returns:
    tf.Tensor:
      Converted tensor.
  """
  return 20.0 / np.log(10) * tf.math.log(tf.maximum(tensor, espilon))

def db_to_gain(tensor: tf.Tensor) -> tf.Tensor:
  """
  Convert from decibel to gain in tensorflow.

  Parameters:
    tensor (tf.Tensor):
      Tensor to convert

  Returns:
    tf.Tensor:
      Converted tensor.
  """
  return tf.pow(10.0, (tensor / 20.0))

def spectrogram_to_db_uint(
   spectrogram: tf.Tensor, db_range: float = 100.0, **kwargs
) -> tf.Tensor:
```

```
  """
  Encodes given spectrogram into uint8 using decibel scale.

  Parameters:
    spectrogram (tf.Tensor):
      Spectrogram to be encoded as TF float tensor.
    db_range (float):
      Range in decibel for encoding.

  Returns:
    tf.Tensor:
      Encoded decibel spectrogram as `uint8` tensor.
  """
  db_spectrogram: tf.Tensor = gain_to_db(spectrogram)
  max_db_spectrogram: tf.Tensor = tf.reduce_max(db_spectrogram)
  int_db_spectrogram: tf.Tensor = tf.maximum(
    db_spectrogram, max_db_spectrogram - db_range
  )
  return from_float32_to_uint8(int_db_spectrogram, **kwargs)

def db_uint_spectrogram_to_gain(
  db_uint_spectrogram: tf.Tensor, min_db: tf.Tensor, max_db: tf.Tensor
) -> tf.Tensor:
  """
  Decode spectrogram from uint8 decibel scale.

  Paramters:
    db_uint_spectrogram (tf.Tensor):
      Decibel spectrogram to decode.
    min_db (tf.Tensor):
      Lower bound limit for decoding.
    max_db (tf.Tensor):
      Upper bound limit for decoding.

  Returns:
    tf.Tensor:
      Decoded spectrogram as `float32` tensor.
  """
  db_spectrogram: tf.Tensor = from_uint8_to_float32(
    db_uint_spectrogram, min_db, max_db
  )
  return db_to_gain(db_spectrogram)


from peewee import SqliteDatabase, Model, CharField, DateField, ForeignKeyField, BooleanField

db = SqliteDatabase('bot.sqlite3')

# class User - for user`s data
class User(Model):
   chat_id = CharField()

   class Meta:
      database = db

# class Harmony - for song`s data
class Harmony(Model):
```

```
    songName = CharField()
    is_existent = BooleanField()
    user = ForeignKeyField(User)
    date = DateField()

    class Meta:
        database = db

if __name__ == '__main__':
    db.create_tables([User, Harmony])



from unittest.mock import patch
import telebot

# ... імпортуйте інші необхідні модулі, включаючи функцію set_song_handler
def test_set_song_handler(mocker):
    # Створити мок-об'єкт для повідомлення
    mock_message = MagicMock()
    mock_message.chat.id = 12345
    mock_message.audio = MagicMock()

    # Викликати функцію з мок-повідомленням
    set_song_handler(mock_message)

    # Перевірити, чи було надіслано повідомлення з інструкцією
    bot_send_message_calls = mock_message.bot.send_message.call_args_list
    assert len(bot_send_message_calls) == 1
    assert bot_send_message_calls[0][1] == \
        "Також введіть назву надісланої пісні у форматі:\n\n<i>Виконавець - Назва пісні</i>".upper()

    # Перевірити форматування тексту
    assert bot_send_message_calls[0][2] == 'HTML'



from unittest.mock import patch, MagicMock
import os.path
from datetime import datetime

# ... імпортуйте інші необхідні модулі, включаючи функцію set_song_name_handler

def test_new_song_name(mocker):
    # Створити мок-об'єкт для повідомлення
    mock_message = MagicMock()
    mock_message.chat.id = 12345
    mock_message.text = "New Song Name"

    # Створити мок-об'єкт для моделі User
    mock_user_model = mocker.patch('harmonybot.models.User')
    mock_user = MagicMock()
    mock_user_model.select.return_value.get = mock_user

    # Створити мок-об'єкт для моделі Harmony
    mock_harmony_model = mocker.patch('harmonybot.models.Harmony')
```

```
    # Створити мок для os.mkdir
    mock_mkdir = mocker.patch('os.mkdir')

    # Створити мок для os.path.exists
    mock_exists = mocker.patch('os.path.exists')

    # Створити мок для datetime.date.today
    mock_today = mocker.patch('datetime.date.today')
    mock_today.return_value = datetime.date(2024, 5, 3)

    # Викликати функцію з мок-повідомленням
    set_song_name_handler(mock_message)

    # Перевірити, чи було створено папку
    mock_mkdir.assert_called_once_with(f'{dirname}/music/new_song_name')

    # Перевірити, чи було надіслано повідомлення про відсутність файлу
    bot_send_message_calls = mock_message.bot.send_message.call_args_list
    assert len(bot_send_message_calls) == 1
    assert bot_send_message_calls[0][1].startswith("На жаль, наразі неможливо")

    # Перевірити видалення папки
    mock_rmdir = mocker.patch('os.rmdir')
    mock_rmdir.assert_called_once_with(f'{dirname}/music/new_song_name')

    # Перевірити створення запису Harmony
    mock_harmony_model.create.assert_called_once_with(
        songName="new_song_name",
        is_existent=False,
        user=mock_user,
        date=datetime.date(2024, 5, 3)
    )



import unittest
import numpy as np

class TestStereoConversion(unittest.TestCase):
    """Tests for the to_stereo function"""

def test_mono_to_stereo(self):
    """Tests if mono waveform is converted to stereo correctly"""
    mono_waveform = np.array([[1, 2, 3], [4, 5, 6]])
    stereo_waveform = to_stereo(mono_waveform)
    self.assertEqual(stereo_waveform.shape, (2, 2))
    self.assertTrue(np.array_equal(stereo_waveform, np.repeat(mono_waveform, 2, axis=-1)))

def test_stereo_waveform(self):
    """Tests if stereo waveform is returned unchanged"""
    stereo_waveform = np.array([[1, 2], [3, 4]])
    returned_waveform = to_stereo(stereo_waveform)
    self.assertEqual(returned_waveform.shape, stereo_waveform.shape)
    self.assertTrue(np.array_equal(returned_waveform, stereo_waveform))

def test_multichannel_to_stereo(self):
    """Tests if waveform with more than 2 channels is truncated correctly"""
```

```
    multichannel_waveform = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
    stereo_waveform = to_stereo(multichannel_waveform)
    self.assertEqual(stereo_waveform.shape, (2, 2))
    self.assertTrue(np.array_equal(stereo_waveform, multichannel_waveform[:, :2]))

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


def test_new_song_name(mocker):
    # Створити мок-об'єкт для повідомлення
    mock_message = MagicMock()
    mock_message.chat.id = 12345
    mock_message.text = "New Song Name"

    # Створити мок-об'єкт для моделі User
    mock_user_model = mocker.patch('harmonybot.models.User')
    mock_user = MagicMock()
    mock_user_model.select.return_value.get = mock_user

    # Створити мок-об'єкт для моделі Harmony
    mock_harmony_model = mocker.patch('harmonybot.models.Harmony')

    # Створити мок для os.mkdir
    mock_mkdir = mocker.patch('os.mkdir')

    # Створити мок для os.path.exists
    mock_exists = mocker.patch('os.path.exists')

    # Створити мок для datetime.date.today
    mock_today = mocker.patch('datetime.date.today')
    mock_today.return_value = datetime.date(2024, 5, 3)

def test_send_welcome_message(mocker):
    # Створення мок-об'єкта для чат-повідомлення
    mock_message = mocker.Mock()
    mock_message.chat.id = 12345
    mock_message.chat.first_name = 'John'
    mock_message.chat.last_name = 'Doe'

    # Виклик функції start_handler з мок-повідомленням
    start_handler(mock_message)

    # Перевірка, чи було викликано метод send_message() бота
    mock_message.bot.send_message.assert_called_once()

    # Перевірка тексту вітального повідомлення
    assert mock_message.bot.send_message.call_args[0][1] == \
        "<b>Вітаю у HarmonyBot, John Doe!</b>"

    # Перевірка форматування тексту
    assert mock_message.bot.send_message.call_args[0][2] == 'HTML'

def test_send_song_format_instruction(mocker):
    # Створення мок-об'єкта для чат-повідомлення
    mock_message = mocker.Mock()
```

```
    mock_message.chat.id = 12345

    # Виклик функції start_handler з мок-повідомленням
    start_handler(mock_message)

    # Перевірка, чи було викликано метод send_message() бота
    mock_message.bot.send_message.assert_called_once()

    # Перевірка тексту інструкції
    assert mock_message.bot.send_message.call_args[0][1] == \
        "Уведіть назву пісні, яку бажаєте розібрати, у форматі:\n\n<i>Виконавець - Назва пісні</i>"

    # Перевірка форматування тексту
    assert mock_message.bot.send_message.call_args[0][2] == 'HTML'

from unittest.mock import patch, MagicMock
import os.path
from datetime import datetime

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)

# Stages
START_ROUTES, END_ROUTES = range(2)
# Callback data
ONE, TWO, THREE, FOUR = range(4)


async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Send message on `/start`."""
    # Get user that sent /start and log his name
    user = update.message.from_user
    logger.info("User %s started the conversation.", user.first_name)
    # Build InlineKeyboard where each button has a displayed text
    # and a string as callback_data
    # The keyboard is a list of button rows, where each row is in turn
    # a list (hence `[[...]]`).
    keyboard = [
        [
            InlineKeyboardButton("1", callback_data=str(ONE)),
            InlineKeyboardButton("2", callback_data=str(TWO)),
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    # Send message with text and appended InlineKeyboard
    await update.message.reply_text("Start handler, Choose a route", reply_markup=reply_markup)
    # Tell ConversationHandler that we're in state `FIRST` now
    return START_ROUTES


async def start_over(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
```

```
    """Prompt same text & keyboard as `start` does but not as new message"""
    # Get CallbackQuery from Update
    query = update.callback_query
    # CallbackQueries need to be answered, even if no notification to the user is needed
    # Some clients may have trouble otherwise. See https://core.telegram.org/bots/api#callbackquery
    await query.answer()
    keyboard = [
        [
            InlineKeyboardButton("1", callback_data=str(ONE)),
            InlineKeyboardButton("2", callback_data=str(TWO)),
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    # Instead of sending a new message, edit the message that
    # originated the CallbackQuery. This gives the feeling of an
    # interactive menu.
    await query.edit_message_text(text="Start handler, Choose a route", reply_markup=reply_markup)
    return START_ROUTES


async def one(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Show new choice of buttons"""
    query = update.callback_query
    await query.answer()
    keyboard = [
        [
            InlineKeyboardButton("3", callback_data=str(THREE)),
            InlineKeyboardButton("4", callback_data=str(FOUR)),
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text(
        text="First CallbackQueryHandler, Choose a route", reply_markup=reply_markup
    )
    return START_ROUTES


async def two(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Show new choice of buttons"""
    query = update.callback_query
    await query.answer()
    keyboard = [
        [
            InlineKeyboardButton("1", callback_data=str(ONE)),
            InlineKeyboardButton("3", callback_data=str(THREE)),
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text(
        text="Second CallbackQueryHandler, Choose a route", reply_markup=reply_markup
    )
    return START_ROUTES


async def three(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Show new choice of buttons. This is the end point of the conversation."""
    query = update.callback_query
```

```
    await query.answer()
    keyboard = [
        [
            InlineKeyboardButton("Yes, let's do it again!", callback_data=str(ONE)),
            InlineKeyboardButton("Nah, I've had enough ...", callback_data=str(TWO)),
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text(
        text="Third CallbackQueryHandler. Do want to start over?", reply_markup=reply_markup
    )
    # Transfer to conversation state `SECOND`
    return END_ROUTES


async def four(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Show new choice of buttons"""
    query = update.callback_query
    await query.answer()
    keyboard = [
        [
            InlineKeyboardButton("2", callback_data=str(TWO)),
            InlineKeyboardButton("3", callback_data=str(THREE)),
        ]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await query.edit_message_text(
        text="Fourth CallbackQueryHandler, Choose a route", reply_markup=reply_markup
    )
    return START_ROUTES


async def end(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int:
    """Returns `ConversationHandler.END`, which tells the
    ConversationHandler that the conversation is over.
    """
    query = update.callback_query
    await query.answer()
    await query.edit_message_text(text="See you next time!")
    return ConversationHandler.END


def main() -> None:
    """Run the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # Setup conversation handler with the states FIRST and SECOND
    # Use the pattern parameter to pass CallbackQueries with specific
    # data pattern to the corresponding handlers.
    # ^ means "start of line/string"
    # $ means "end of line/string"
    # So ^ABC$ will only allow 'ABC'
    conv_handler = ConversationHandler(
        entry_points=[CommandHandler("start", start)],
        states={
            START_ROUTES: [
```

```
            CallbackQueryHandler(one, pattern="^" + str(ONE) + "$"),
            CallbackQueryHandler(two, pattern="^" + str(TWO) + "$"),
            CallbackQueryHandler(three, pattern="^" + str(THREE) + "$"),
            CallbackQueryHandler(four, pattern="^" + str(FOUR) + "$"),
        ],
        END_ROUTES: [
            CallbackQueryHandler(start_over, pattern="^" + str(ONE) + "$"),
            CallbackQueryHandler(end, pattern="^" + str(TWO) + "$"),
        ],
    },
    fallbacks=[CommandHandler("start", start)],
)

# Add ConversationHandler to application that will be used for handling updates
application.add_handler(conv_handler)

# Run the bot until the user presses Ctrl-C
application.run_polling(allowed_updates=Update.ALL_TYPES)

def test_new_song_name(mocker):
    # Створити мок-об'єкт для повідомлення
    mock_message = MagicMock()
    mock_message.chat.id = 12345
    mock_message.text = "New Song Name"

    # Створити мок-об'єкт для моделі User
    mock_user_model = mocker.patch('harmonybot.models.User')
    mock_user = MagicMock()
    mock_user_model.select.return_value.get = mock_user

    # Створити мок-об'єкт для моделі Harmony
    mock_harmony_model = mocker.patch('harmonybot.models.Harmony')

    # Створити мок для os.mkdir
    mock_mkdir = mocker.patch('os.mkdir')

    # Створити мок для os.path.exists
    mock_exists = mocker.patch('os.path.exists')

    # Створити мок для datetime.date.today
    mock_today = mocker.patch('datetime.date.today')
    mock_today.return_value = datetime.date(2024, 5, 3)


    # Викликати функцію з мок-повідомленням
    set_song_name_handler(mock_message)

    # Перевірити, чи було створено папку
    mock_mkdir.assert_called_once_with(f'{dirname}/music/new_song_name')

    # Перевірити, чи було надіслано повідомлення про відсутність файлу
    bot_send_message_calls = mock_message.bot.send_message.call_args_list
    assert len(bot_send_message_calls) == 1
    assert bot_send_message_calls[0][1].startswith("На жаль, наразі неможливо")

    # Перевірити видалення папки
    mock_rmdir = mocker.patch('os.rmdir')
```

```
    mock_rmdir.assert_called_once_with(f'{dirname}/music/new_song_name')

    # Перевірити створення запису Harmony
    mock_harmony_model.create.assert_called_once_with(
        songName="new_song_name",
        is_existent=False,
        user=mock_user,
        date=datetime.date(2024, 5, 3)
    )

def test_info_handler(mocker):
    # Створити мок-об'єкт для повідомлення
    mock_message = MagicMock()
    mock_message.chat.id = 12345


# Викликати функцію з мок-повідомленням
info_handler(mock_message)

# Перевірити, чи було надіслано повідомлення з інформацією
bot_send_message_calls = mock_message.bot.send_message.call_args_list
assert len(bot_send_message_calls) == 1
assert bot_send_message_calls[0][1] == \
       "Вінницький національний технічний університет\nфакультет інформаційних технологій та
комп'ютерної інженерії\nінженерія програмного забезпечення\n\nРозробка на
дисципліну\n<i>Проєктний практикум - 2024</i>".upper()

# Перевірити форматування тексту
assert bot_send_message_calls[0][2] == 'HTML'

mport logging
from html import escape
from uuid import uuid4

from telegram import InlineQueryResultArticle, InputTextMessageContent, Update
from telegram.constants import ParseMode
from telegram.ext import Application, CommandHandler, ContextTypes, InlineQueryHandler

# Enable logging
logging.basicConfig(
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s", level=logging.INFO
)
# set higher logging level for httpx to avoid all GET and POST requests being logged
logging.getLogger("httpx").setLevel(logging.WARNING)

logger = logging.getLogger(__name__)


# Define a few command handlers. These usually take the two arguments update and
# context.
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Send a message when the command /start is issued."""
    await update.message.reply_text("Hi!")


async def help_command(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Send a message when the command /help is issued."""
```

```
    await update.message.reply_text("Help!")


async def inline_query(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None:
    """Handle the inline query. This is run when you type: @botusername <query>"""
    query = update.inline_query.query

    if not query:  # empty query should not be handled
        return

    results = [
        InlineQueryResultArticle(
            id=str(uuid4()),
            title="Caps",
            input_message_content=InputTextMessageContent(query.upper()),
        ),
        InlineQueryResultArticle(
            id=str(uuid4()),
            title="Bold",
            input_message_content=InputTextMessageContent(
                f"<b>{escape(query)}</b>", parse_mode=ParseMode.HTML
            ),
        ),
        InlineQueryResultArticle(
            id=str(uuid4()),
            title="Italic",
            input_message_content=InputTextMessageContent(
                f"<i>{escape(query)}</i>", parse_mode=ParseMode.HTML
            ),
        ),
    ]

    await update.inline_query.answer(results)


def main() -> None:
    """Run the bot."""
    # Create the Application and pass it your bot's token.
    application = Application.builder().token("TOKEN").build()

    # on different commands - answer in Telegram
    application.add_handler(CommandHandler("start", start))
    application.add_handler(CommandHandler("help", help_command))

    # on inline queries - show corresponding inline results
    application.add_handler(InlineQueryHandler(inline_query))

    # Run the bot until the user presses Ctrl-C
    application.run_polling(allowed_updates=Update.ALL_TYPES)

def test_set_song_handler(mocker):
    # Створити мок-об'єкт для повідомлення
    mock_message = MagicMock()
    mock_message.chat.id = 12345
    mock_message.audio = MagicMock()
```

```
    # Викликати функцію з мок-повідомленням
    set_song_handler(mock_message)

    # Перевірити, чи було надіслано повідомлення з інструкцією
    bot_send_message_calls = mock_message.bot.send_message.call_args_list
    assert len(bot_send_message_calls) == 1
    assert bot_send_message_calls[0][1] == \
        "Також введіть назву надісланої пісні у форматі:\n\n<i>Виконавець - Назва пісні</i>".upper()

    # Перевірити форматування тексту
    assert bot_send_message_calls[0][2] == 'HTML'
```

Лістинг модуля models.py

```
from peewee import SqliteDatabase, Model, CharField, DateField, ForeignKeyField, BooleanField

db = SqliteDatabase('bot.sqlite3')


# class User - for user`s data
class User(Model):
    chat_id = CharField()

    class Meta:
        database = db


# class Harmony - for song`s data
class Harmony(Model):
    songName = CharField()
    is_existent = BooleanField()
    user = ForeignKeyField(User)
    date = DateField()

    class Meta:
        database = db

from unittest.mock import patch

# ... імпортуйте ваші моделі User та Harmony


def test_create_user():
    user = User(chat_id=12345)
    user.save()  # Збереження об'єкта в базу даних (потрібен mock для save())

    # Перевірка атрибутів
    assert user.chat_id == 12345



@patch('harmonybot.models.User.objects.delete')  # Mock видалення об'єктів
def test_cleanup_users(mock_delete):
    # Створити декілька об'єктів User
    User.objects.create(chat_id=11111)
    User.objects.create(chat_id=22222)
```

```
    # Викликати видалення всіх об'єктів User
    User.objects.all().delete()

    # Перевірити, чи викликалася функція видалення
    mock_delete.assert_called_once()



def test_create_harmony():
    # Створити mock-об'єкт для моделі User
    mock_user = MagicMock()

    harmony = Harmony(
        songName="Test Song",
        is_existent=False,
        user=mock_user,
        date=datetime.date.today()
    )
    harmony.save()  # Збереження об'єкта в базу даних (потрібен mock для save())

    # Перевірка атрибутів
    assert harmony.songName == "Test Song"
    assert harmony.is_existent is False
    assert harmony.user == mock_user
    assert harmony.date == datetime.date.today()


if __name__ == '__main__':
    db.create_tables([User, Harmony])
```

ДОДАТОК Г – ІЛЮСТРАТИВНА ЧАСТИНА

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ТЕЛЕГРАМ-БОТА ДЛЯ ОБРОБКИ МУЗИЧНОГО КОНТЕНТУ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

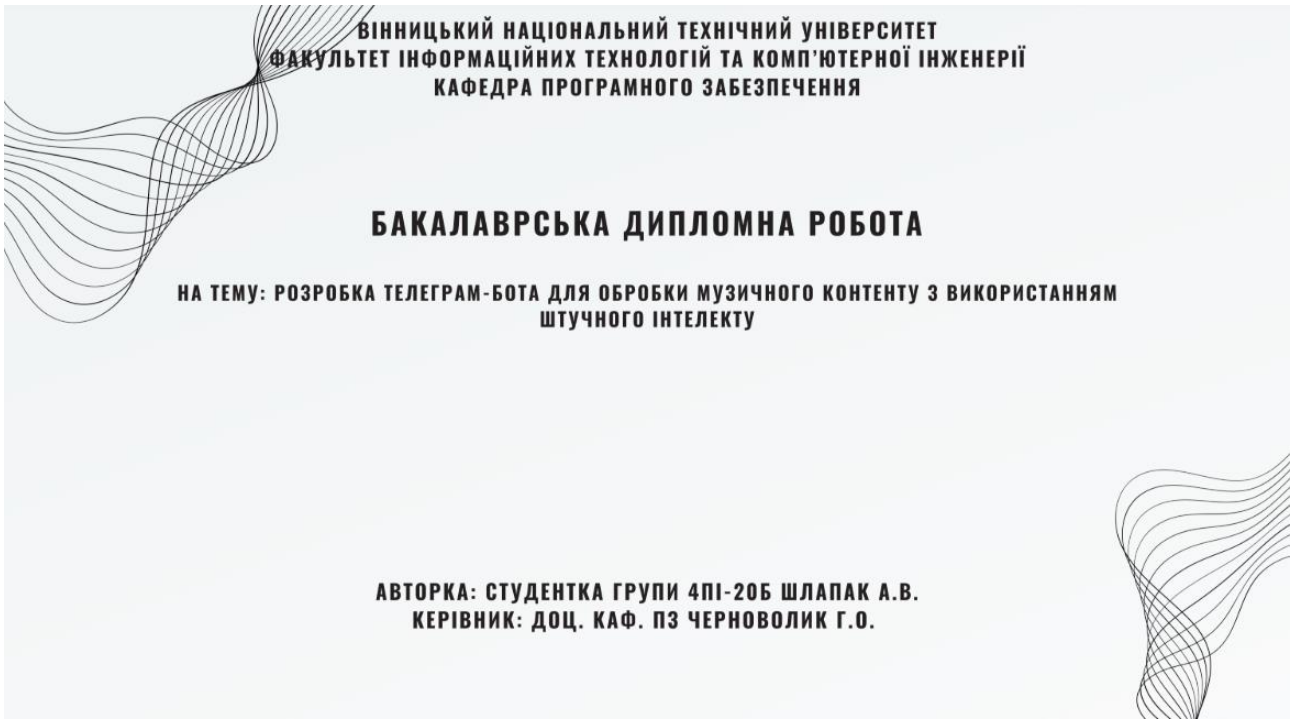

Рисунок Г.1 – Титульний слайд

Рисунок Г.2 – Актуальність теми

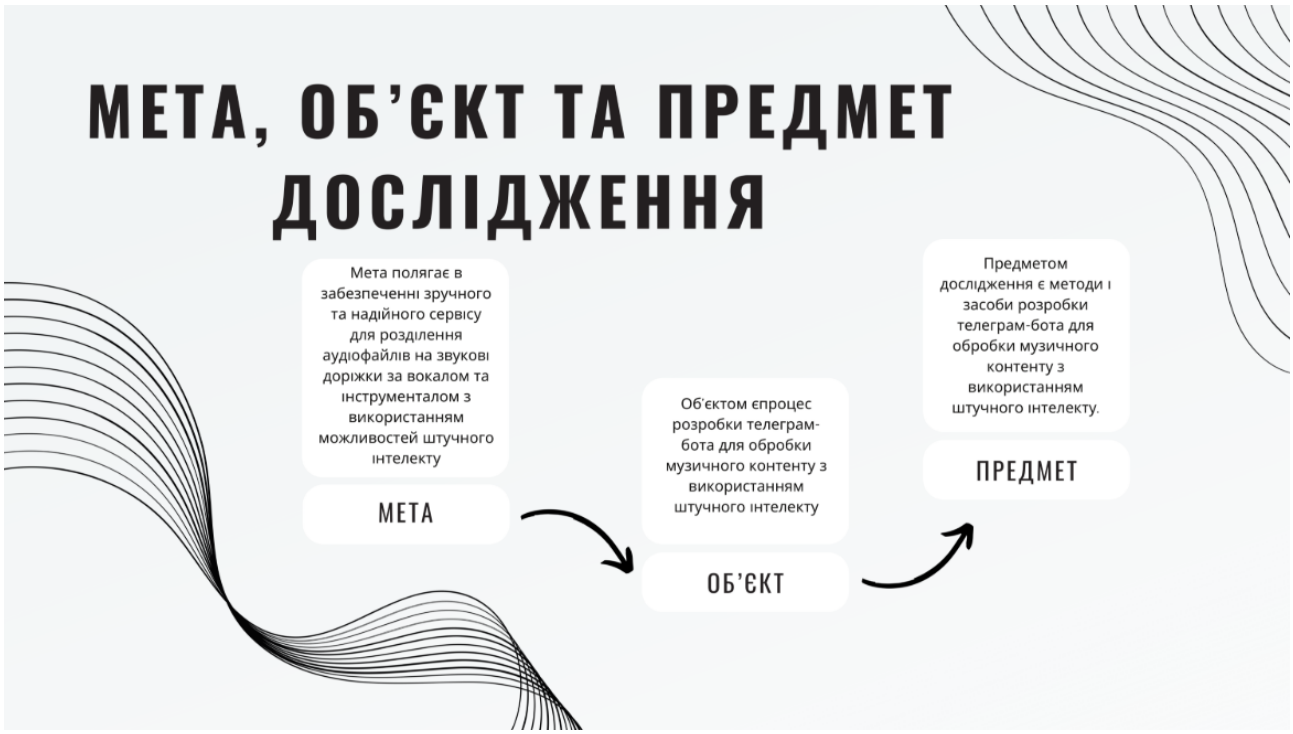

Рисунок Г.3 – Мета, об'єкт та предмет дослідження

Рисунок Г.4 – Новизна отриманих результатів

ПОРІВНЯЛЬНИЙ АНАЛІЗ АНАЛОГІВ

Характеристика	Phonic	Lalal.ai	Audacity	FL Studio	HarmonyBot
Інтерфейс українською мовою	-	-	-	-	+
Вільне безкоштовне користування	+	-	+	-	+
Автоматичне розбиття інструменталу на звукові доріжки	+	+	-	-	+
Можливість користуватись вільно у месенджері Telegram	-	-	-	-	+
Загальна оцінка	50%	25%	25%	0%	100%

Рисунок Г.5 – Порівняльний аналіз аналогів

АЛГОРИТМИ ЗБЕРЕЖЕННЯ ОБРОБЛЕНИХ АУДІОФАЙЛІВ

Блок-схема описує алгоритм роботи Telegram-бота:

Початок: Користувач запускає програму.
Цикл роботи:
Очікування на повідомлення: Бот очікує на введення користувача (текст або команда).
Обробка запиту: Бот аналізує введений текст або команду.
Перевірка наявності інформації: Бот шукає відповідну інформацію в системі.
Інформація знайдена: Бот надає інформацію користувачеві (текст, аудіо).
Інформація відсутня: Бот повідомляє про відсутність інформації та пропонує спробувати знову або ввести інший запит.
Збереження запиту: Бот додає новий запит до бази даних для навчання та покращення роботи.
Кінець: Користувач може вийти з бота або продовжити роботу.

Рисунок Г.6 – Алгоритми збереження оброблених аудіофайлів

Рисунок Г.7 – Алгоритм обробки запитів користувачів

Рисунок Г.8 – Розробка інтерфейсу

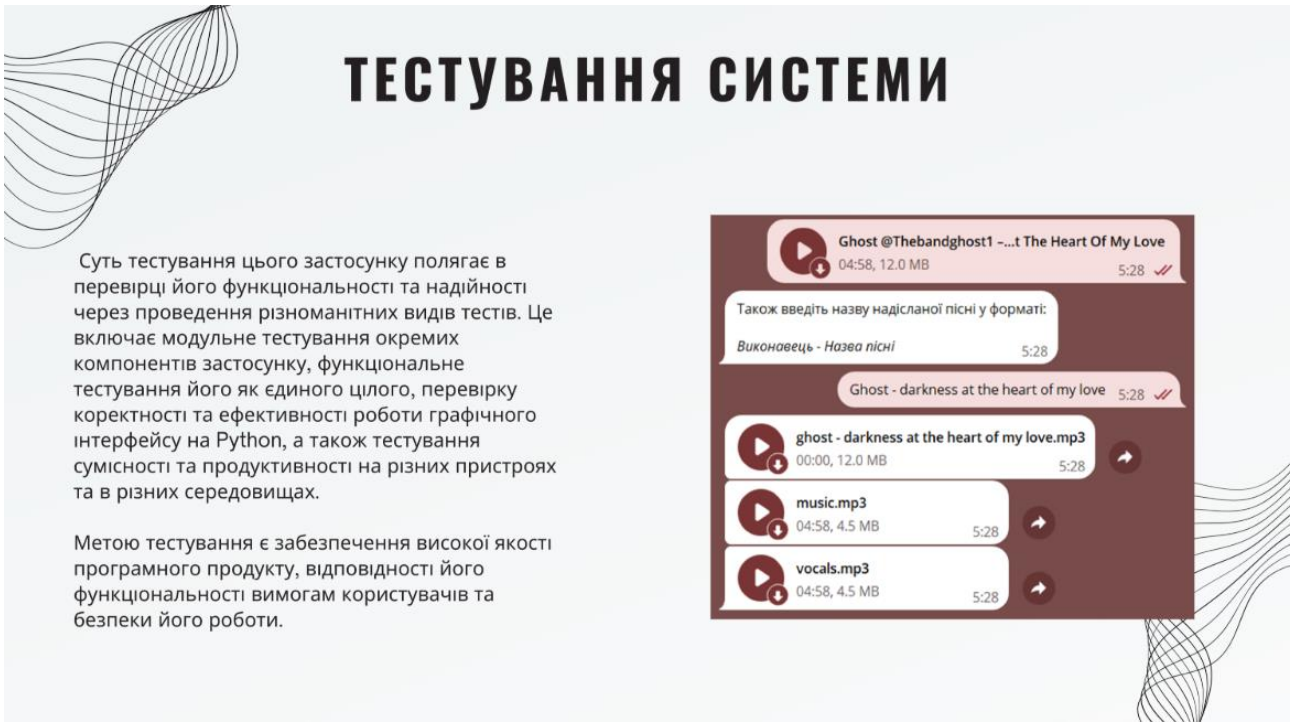

Рисунок Г.9 – Тестування системи

Рисунок Г.10 – Висновки