МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

КВАЛІФІКАЦІЙНА БАКАЛАВРСЬКА РОБОТА[1]

Головачук Анастасія Олександрівна
(прізвище, ім'я, по батькові здобувача)

на тему Розробка mp3-плеєру

(повна назва теми)

за матеріалами праць провідних спеціалістів з розробки ПЗ та проектування БД

(повна назва бази дослідження)

науковий керівник к.е.н., доцент _____ Лисенко В. С.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 11.06.2025 № 12
Завідувач кафедри _____
(підпис)
д.т.н., професор Зеленський О.С.
Наук. ступень, вчене звання *Ініціали, прізвище*

Кривий Ріг – 2025

[1] Розробка mp3

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ/факультет	Інформаційних технологій
Кафедра	Інформатики і прикладного програмного забезпечення
Спеціальність	Інженерія програмного забезпечення
Форма навчання	Денна

«ЗАТВЕРДЖУЮ»
Завідувач кафедри _____ Зеленський О.С.
(підпис) (Прізвище, ініціали)
«11» червня 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ[2]

1. Тема роботи «Розробка mp3 плеєру»

Керівник роботи к.е.н., доцент Лисенко В. С.
затверджені наказом закладу вищої освіти від «04» квітня 2025 р. № 222-ст

2. Строк подання здобувачем роботи до «09» червня 2025 р.

3. Зміст кваліфікаційної роботи, об’єкт, предмет та мета дослідження:

Розділ 1. Постановка задачі

Розділ 2. Розробка алгоритму розв’язання задачі

Розділ 3. Розробка бази даних задачі

Розділ 4. Розробка програмного забезпечення

Об’єкт дослідження: mp3 плеєр

Предмет дослідження: програмне забезпечення

Мета кваліфікаційної роботи: створенння mp3 плеєру на мові C++ з використанням OpenGL

5. Дата видачі завдання «04» квітня 2025 р.

[2] Розробка mp3 плеєру

ЗГОДА здобувачки вищої освіти

Державного університету економіки і технологій про перевірку кваліфікаційної роботи на прояви академічного плагіату
та розміщення в Репозитарії Університету

Я, Головачук Анастасія Олександрівна,
підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота
«Розробка mp3 плеєру»
виконана самостійно та не містить академічного плагіату. Я не надавала і не одержувала недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлена. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформована, що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлена.

Дата підпис ініціали, прізвище (власноруч)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів МДР	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	04.04.2025-13.04.2025	
2	Підготовка розділу 2	14.04.2025-26.04.2025	
3.	Підготовка розділу 3	27.04.2025-15.05.2025	
4	Підготовка розділу 4	16.05.2025-08.06.2025	
5	Реєстрація завершеної кваліфікаційної роботи	09.06.2025	Реєстраційний №___ «09»червня 2025 р.
6	Отримання відгуку від наукового керівника	10.06.2025	
7	Подання кваліфікаційної роботи на перегляд завідувачу кафедри	11.06.2025	
8	Отримання зовнішньої рецензії	12.06.2025	
9	Попередній захист кваліфікаційної роботи на кафедрі	13.06.2025	
10	Підготовка до захисту в ЕК	16.06.2025-21.06.2025	

Завдання підготував науковий керівник _____ Лисенко В. С.
(підпис) (прізвище та ініціали)

Завдання одержав _____ Головачук А. О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

на бакалаврську роботу

«Розробка mp3 плеєру»

Головачук А. О.

Кваліфікаційна робота за спеціальністю 121 – Інженерія програмного забезпечення – Державний університет економіки і технологій. – Кривий Ріг, 2025.

У кваліфікаційній роботі розроблено музичний програвач на мові програмування C++ з використанням графічного API OpenGL для побудови користувацького інтерфейсу.

Для обробки та відтворення аудіофайлів використано звукове API OpenAL, що забезпечує якісне звучання та базову взаємодію з аудіо.

Як результат кваліфікаційної роботи створено десктопний музичний програвач, що відповідає базовим вимогам користувача щодо функціональності, зручності встановлення та доступності.

Ключові слова: C++, OPENGL, OPENAL, C#, WINDOWS FORMS, МУЗИЧНИЙ ПРОГРАВАЧ, ІНСТАЛЯТОР, САЙТ-ЛЕНДІНГ.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

C++	Це мова програмування, яка поєднує принципи процедурного та об'єктно-орієнтованого програмування.
API OpenGL	Представляє собою набір інструментів для створення як двовимірної, так і тривимірної графіки, що дозволяє відтворювати різноманітні візуальні образи. Цей інтерфейс забезпечує прямий доступ до можливостей відеокарти, що дає змогу ефективно розподіляти навантаження між центральним і графічним процесорами. Завдяки такому підходу можна реалізувати плавне відображення складних графічних сцен із високим рівнем деталізації в режимі реального часу.
C#	Це сучасна мова програмування, створена компанією Microsoft, що поєднує об’єктно-орієнтований підхід з високим рівнем абстракції. Вона призначена для розробки різноманітних застосунків — від десктопних програм до веб- і мобільних рішень. Мова має вбудовані механізми автоматичного керування пам’яттю, що допомагає уникнути типових помилок, пов’язаних із ресурсами. Також у C# доступний широкий набір бібліотек, які спрощують роботу з графікою, базами даних, мережевими протоколами та іншими компонентами. Завдяки тісній інтеграції з платформою .NET, C# забезпечує ефективне виконання коду і хорошу сумісність між різними системами, що робить її популярним вибором для створення складних і надійних програмних продуктів.
Windows Forms	Це платформа для створення графічних застосунків у середовищі Windows, що дозволяє легко створювати віконні інтерфейси за допомогою різноманітних елементів управління. Завдяки цій технології можна швидко реалізувати функціональні та зручні для користувача додатки. Ця технологія базується на платформі .NET і надає можливість створювати застосунки з графічним інтерфейсом за допомогою візуального конструктора, що значно полегшує процес проєктування без потреби в написанні великої кількості коду.

Інсталятор	Спеціалізована програма, призначена для автоматичного розгортання програмного забезпечення на комп’ютері користувача. Вона виконує копіювання необхідних файлів у відповідні директорії, створює ярлики, реєструє компоненти в системі та, за потреби, встановлює додаткові залежності. Використання інсталятора спрощує процес встановлення програми, забезпечуючи зручність та зменшуючи ймовірність помилок під час ручного налаштування. Такий підхід особливо актуальний для десктопних застосунків, орієнтованих на кінцевого користувача без спеціальних технічних знань.
API OpenAL	Прикладний програмний інтерфейс, призначений для роботи з аудіо у форматі тривимірного простору. Він дозволяє відтворювати, позиціонувати та контролювати звукові ефекти у додатках, забезпечуючи просторове звучання. У даному програмному продукті цей інтерфейс використовується для реалізації програвання аудіофайлів та базової обробки звукових подій.

ЗМІСТ

ВСТУП

У сучасному цифровому середовищі мультимедійні програми, зокрема ті, що дозволяють відтворювати аудіофайли, стали невід'ємною частиною повсякденного життя користувачів. Зростання кількості цифрового контенту та прагнення до комфортного його споживання створюють попит на зручні, функціональні та візуально привабливі музичні програвачі. Саме тому метою даної роботи є створення власного аудіоплеєра, який поєднує в собі простоту використання, гнучкість налаштувань та сучасний дизайн.

У процесі роботи над проєктом передбачається створити застосунок мовою C++, при цьому для побудови візуальної частини буде задіяна бібліотека OpenGL. Це дозволяє забезпечити ефективну візуалізацію в реальному часі та створити унікальний зовнішній вигляд застосунку. Відтворення звукових файлів забезпечується за допомогою бібліотеки OpenAL, що дозволяє реалізувати якісне звучання із підтримкою регулювання гучності, позиціонування джерел у просторі та застосування різноманітних ефектів без істотного навантаження на комп’ютер. Додатково, з метою полегшення інсталяції програми кінцевими користувачами, створено інсталяційний модуль мовою C# із застосуванням Windows Forms, що забезпечує зрозумілий процес встановлення без необхідності вручну переміщувати файли. А для зручного поширення продукту серед користувачів буде розроблена окрема лендінг-сторінка з коротким описом і можливістю завантаження програвача.

Предметом дослідження в роботі є підходи до створення мультимедійного програмного забезпечення з використанням сучасних бібліотек і мов програмування. У центрі дослідження знаходиться повний процес розробки аудіоплеєра — починаючи з дизайну інтерфейсу та реалізації ключових функцій, і завершуючи адаптацією програми для зручного використання користувачами.

РОЗДІЛ 1
ПОСТАНОВКА ЗАДАЧІ

1.1. Основні визначення та характеристика задачі

Розробка mp3 плеєру є актуальним завданням у сфері створення прикладного програмного забезпечення, що поєднує в собі обробку аудіо, створення графічного інтерфейсу та зручність у користуванні. Плеєр має бути функціональним, стабільним, візуально привабливим і простим у використанні.

Основною технологією для створення графічної частини застосунку є OpenGL — графічний інтерфейс програмування, що дозволяє виводити двовимірну та тривимірну графіку. Завдяки роботі з ресурсами відеокарти, він забезпечує високу продуктивність при відображенні інтерфейсу та дозволяє реалізувати плавну візуалізацію елементів керування, анімації, ефекти тощо.

Для обробки звуку у плеєрі застосовується OpenAL — звукове API, що надає інструменти для роботи з аудіо-потоками, налаштуванням гучності, звуковими ефектами та управлінням позиціонуванням джерел звуку. Цей інструмент забезпечує якісне та реалістичне звучання, що є важливою складовою користувацького досвіду.

Таким чином, головне завдання полягає в розробці багатокомпонентного програмного продукту, що об'єднує графічний і звуковий модулі.

1.2. Аналіз існуючих на ринку аналогічних програмних забезпечень

Ринок програм для відтворення музики на комп'ютерах/ноутбуках представлений великою кількістю застосунків, кожен із яких має свої переваги, недоліки та функціональні особливості, переглянемо декілька плеєрів та порівняємо їх між собою. Це допоможе краще зрозуміти потреби користувачів, виявити популярні функції, а також побачити недоліки або обмеження у вже реалізованих продуктах. Вивчивши досвід інших розробників, можна буде уникнути типових помилок і зробити застосунок більш зручним та ефективним.

Аналіз стосуватиметься не лише зовнішнього вигляду інтерфейсу, а й можливостей взаємодії з файлами, якості відтворення звуку, наявності додаткових функцій тощо. Також буде зроблено акцент на технологіях, що використовуються всередині таких програм — це допоможе краще зрозуміти, які інструменти варто застосувати у власній розробці.

Такий аналіз — це перший крок до створення конкурентоспроможного продукту, який відповідатиме сучасним вимогам і запитам користувачів.

Одним із відомих та популярних MP3-плеєрів є AIMP, який має стабільно високу оцінку користувачів — у середньому 4.5 із 5 балів. Цей застосунок доступний для різних операційних систем, зокрема Windows, Linux та macOS, що забезпечує його широке використання серед різних категорій користувачів.

AIMP має зручний процес інсталяції: після запуску інсталяційного файлу з'являється стартове вікно, де можна обрати мову інтерфейсу (рис. 1.1).

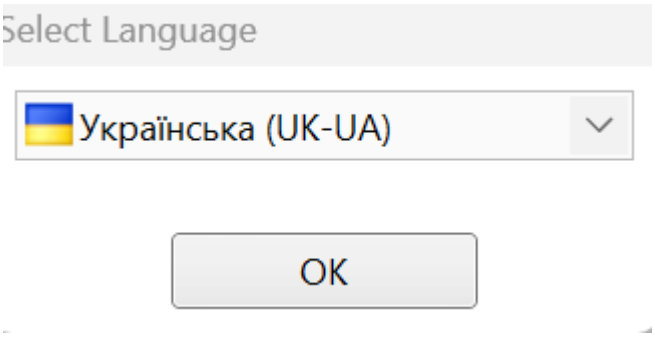

Рис. 1.1. Стартове вікно AIMP з обранням мови інтерфейсу

Далі, після обрання мови з’являється майстер встановлення програми (рис. 1.2).

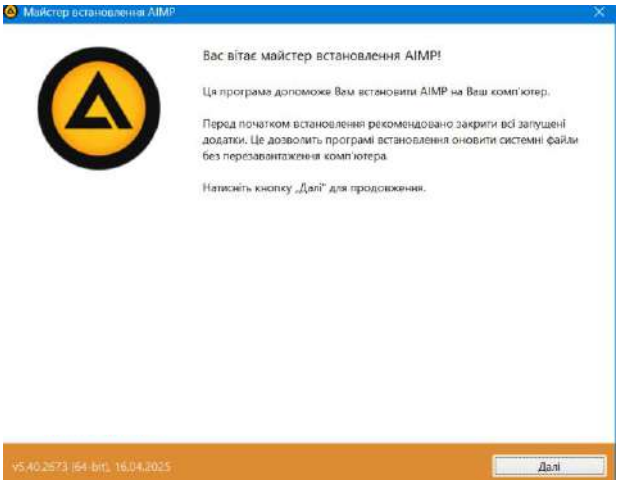

Рис. 1.2. Майстер встановлення AIMP

Після завершення встановлення з’явилося повідомлення про успішне інсталювання програми (рис. 1.3).

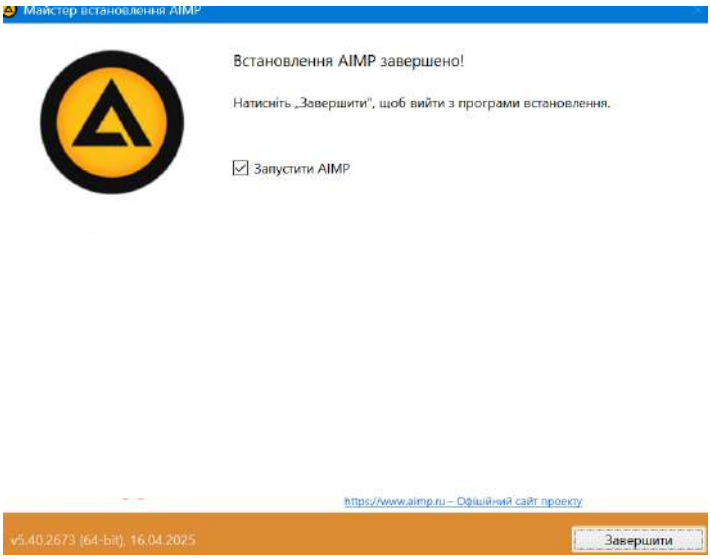

Рис. 1.3. Повідомлення про успішне інсталювання програми

На робочому столі з’явився ярлик аудіо-програвача, який має помітний та мінімалістичний дизайн (рис. 1.4).

Рис. 1.4. Ярлик аудіо-програвача

Після запуску плеєра з’являється вікно з налаштуванням інтерфейсу програвача, можна обрати колір фону (світлий або темний), потім можна обрати сам вигляд програвача (сучасний, звичайний та стандартний) та зовнішній вигляд (застосування акцентного кольору до нижньої панелі, застосування акцентного кольору до рамки вікна та закруглення кутів) (рис. 1.5).

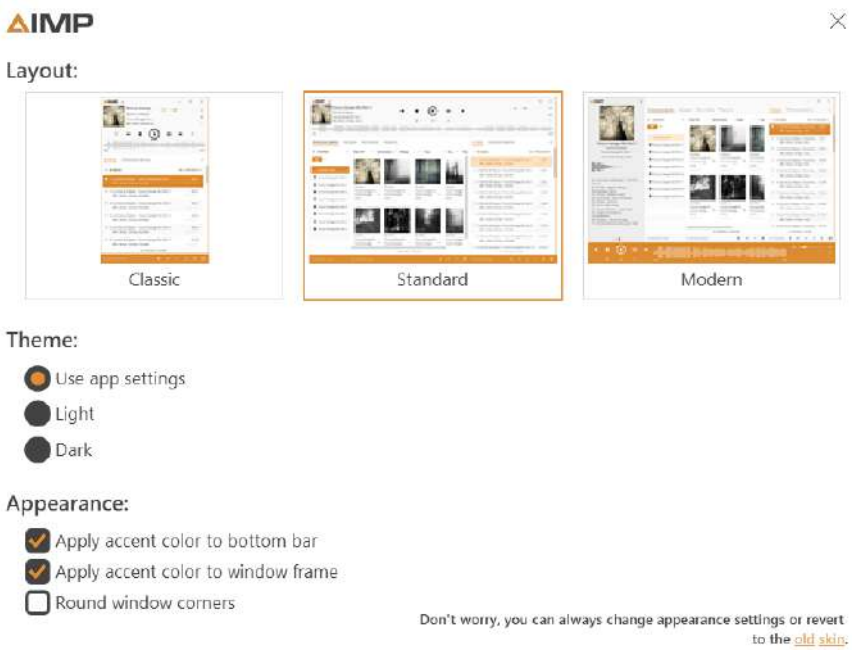

Рис. 1.5. Налаштування плеєру

Після усіх цих налаштування з'являється вікно звичайного плеєру з 4 вкладками: «Локальні файли», «Закладки», «Підкасти», «Мої хмари». Так як музики ще встановленої немає, то в цих вкладках пусто. Після скачування пісень їх треба було перенести до плеєру, бо вони там в списках не з'явилися, я вважаю, це є не дуже зручним. Вгорі над піснями знаходяться заголовки стовпців за якими потім можна буде сортувати пісні (рис. 1.6).

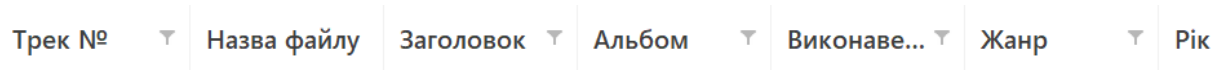

Рис. 1.6. Стовпці для сортування пісень

Також, є можливість групувати пісні за такими запитами: «Без групування», «Нові треки», «Найкращі треки», «Альбом», «Виконавець альбому - Альбом», «Виконавець - Альбом», «Рік – Виконавець альбому - Альбом», «Жанр – Виконавець альбому - Альбом», після створень таких групувань користувачем дуже зручно орієнтуватися при великій кількості пісень та це гарна можливість прослуховувати ті пісні, які саме зараз підходять під настрій та бажання (рис. 1.7).

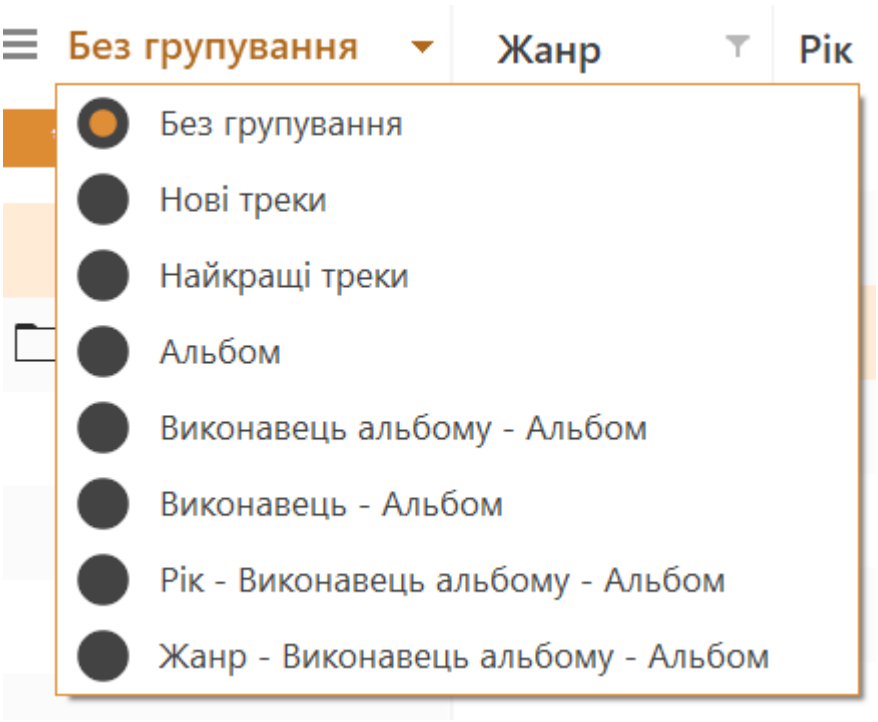

Рис. 1.7. Групування треків в плеєрі

Так як пісень в плеєрі не багато створювати групування немає потреби, але для загального користування це гарне доповнення. Тепер перейдемо до операцій з треками, після натискання на трек правою кнопкою миші. Одразу з'являється вікно з діями до треку. Там є все починаючи з відтворення закінчуючи видаленням треку, також, деякі ці функції можна зробити за допомогою комбінації клавіш, ці комбінації зображені в меню (рис. 1.8).

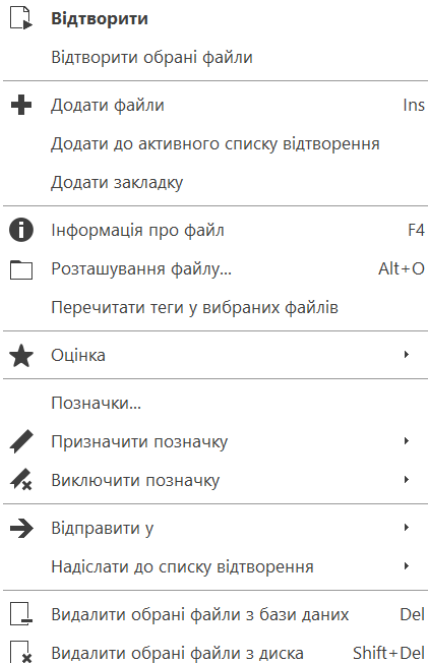

Рис. 1.8. Меню функцій з піснями

Інтерфейс аудіоплеєра містить усі основні елементи керування відтворенням музичних композицій. Кнопки логічно розташовані - основні елементи, які використовують найпоширеніші, централізовані, а додаткові функції згруповані в сторони інтерфейсу, щоб уникнути перевантаження місця зберігання.

Контролер гучності реалізований у вигляді горизонтального повзунка, щоб забезпечити плавні зміни до рівня звуку. Крім того, панель має візуальні показники, які відображають статус гри. Наприклад, повторювана діяльність або тип "перетасування".

Спектральний формат сигналу, показаний у вигляді хвилі, допомагає вам орієнтуватися на структурі стежки, щоб ви могли бачити тишу, динамічні моменти або кінчик обсягу. Це особливо корисно, не слухаючи попереднього композиції (рис 1.9).

Рис. 1.9. Інтерфейс програвача

У програмі доступний вбудований модуль керування звуковими ефектами, який дає змогу тонко налаштувати звучання аудіофайлів. Завдяки цьому можна власноруч змінювати параметри, які представлені там. Кожен ефект представлений окремим повзунком, що дозволяє досягти бажаного звукового результату відповідно до особистих вподобань або специфіки композиції.

Крім того, є функції автоматичного згасання звуку під час пауз або навігації по треку, що забезпечує плавність переходів. Також присутній регулятор балансу між каналами, що допомагає досягти просторової рівноваги звучання. При необхідності всі налаштування можна легко повернути до початкових значень. Такий набір можливостей надає користувачеві гнучкість і комфорт у роботі з аудіо (рис. 1.10).

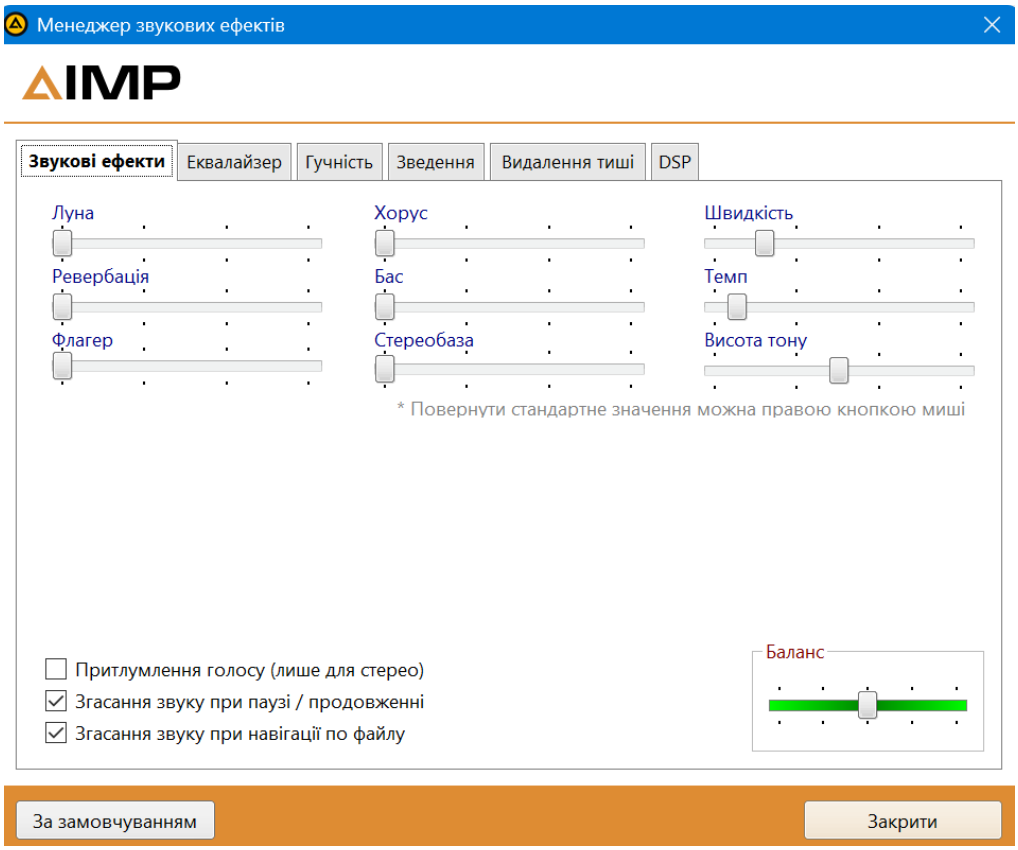

Рис. 1.10. Менеджер звукових ефектів

Легкий у користуванні та розумінні плеєр, тому не дивно, що його оцінка 4.5 із 5.

У рамках подальшого вивчення було обрано програму WinAMP Lite — легку версію популярного аудіоплеєра, сумісну з операційними системами Windows та MacOS. На платформі, де була завантажена програма, користувачі поставили їй максимальну оцінку, що може свідчити про її високу функціональність і комфорт у роботі.

Для початку встановлення програми з’являться вікно з вибором мови: англійська або іспанська, на жаль, вибрати українську не можна, це вже мінус, адже, користувачі які не знають цих мов будуть мати проблеми з користуванням (рис. 1.11).

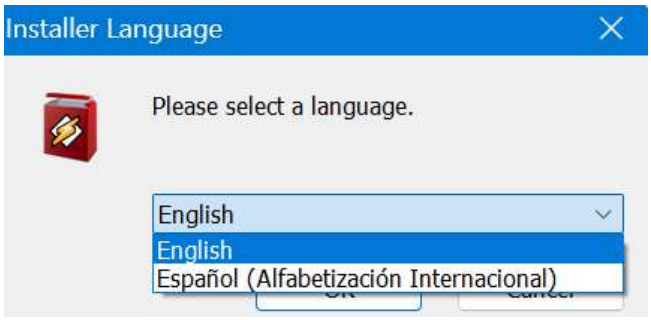

Рис. 1.11. Вибір мови WinAMP Lite

Інсталювання пройшло швидко, на головному екрані з’явивля ярлик плеєру, помітний та мінімалістичний (рис. 1.12).

Рис. 1.12. Ярлик плеєру WinAMP Lite

Вигляд плеєру не є сильно зрозумілим та естетично гарним, також його не можна збільшити щоб в ньому розібратися детально, немає змоги зміни зовнішнього вигляду (рис. 1.13).

Рис. 1.13 MP3-плеєр WinAMP Lite

Щоб додати новий трек до списку відтворення, користувачі повинні натиснути «List OPTS», вибрати «New List» та вручну перенести потрібний аудіофайл. Такий підхід до взаємодії з інтерфейсами менш інтуїтивно зрозумілий, і потребує додаткового часу для початку користування. Стовпців для сортування, як це було в минулому плеєрі немає (рис. 1.14).

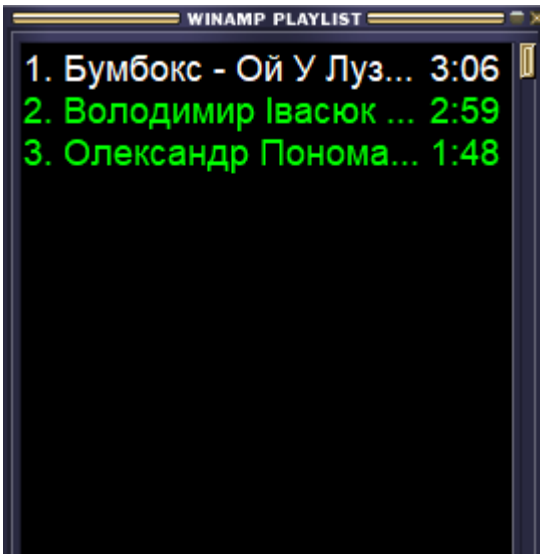

Рис. 1.14. Плейлист WinAMP Lite

У верхній частині програвача розміщено панель, де виводиться назва музичного файлу, тривалість його звучання, а також зазначено ім’я виконавця чи назву альбому. Поряд із цим виводяться основні технічні параметри звукового файлу — зокрема, бітрейт (наприклад, 320 кбіт/с), частота дискретизації (наприклад, 48 кГц) і тип звучання (моно або стерео).

Зліва розміщено графічний індикатор, який візуалізує аудіосигнал у вигляді стовпчикового аналізатора, що змінюється залежно від гучності та частотного діапазону. Під ним розташована шкала регулювання гучності.

У центрі інтерфейсу знаходиться панель керування відтворенням, яка включає стандартні кнопки: перемотування назад, запуск, пауза, зупинка та перемотування вперед. Крім того, присутні кнопки для ввімкнення режимів випадкового відтворення (Shuffle) та повтору треків (Repeat).

Праворуч розташовані елементи доступу до додаткових функцій: еквалайзера (EQ), плейлиста (PL) та налаштувань, що позначені іконкою у вигляді ключа.

Сам інтерфейс програвача є простий у використанні, зручним розміщенням елементів (рис. 1.15).

Рис. 1.15. Програвач WinAMP Lite

Нижче програвача знаходиться еквалайзер для подальшого налаштування відтворюваного треку. Він надає можливість змінювати гучність окремих частот, що дозволяє досягти бажаного звукового ефекту залежно від типу музики або індивідуальних вподобань користувача.

Еквалайзер містить десять повзункових регуляторів, кожен із яких налаштовує певний діапазон частот — від 70 герц до 16 кілогерц. Окремий регулятор PREAMP, розташований ліворуч, дозволяє змінювати загальний рівень підсилення сигналу до його обробки.

У верхній частині панелі присутні кнопки, які дозволяють увімкнути еквалайзер (ON), активувати автоматичне налаштування (AUTO), а також вибрати один із заздалегідь збережених шаблонів налаштувань (PRESETS).

Завдяки зрозумілому і функціональному дизайну, еквалайзер забезпечує зручний інструмент для покращення якості звуку відповідно до потреб. (рис. 1.16).

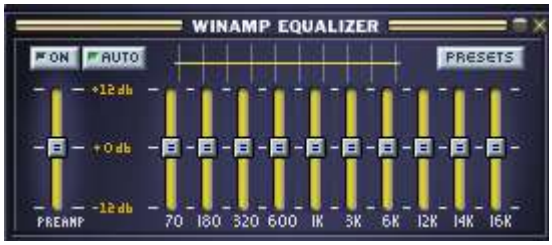

Рис. 1.16. Еквалайзер аудіопрогравача WinAMP Lite

Загалом, якщо не зважати на темне оформлення та дещо тривалий процес додавання композицій до списку відтворення, інтерфейс програми виглядає інтуїтивно зрозумілим і лаконічним — усі основні функції знаходяться під рукою. Саме завдяки простоті використання та зручності управління цей

програвач здобув популярність серед великої кількості користувачів у різних країнах.

У підсумку, одним із найвідоміших і найбільш затребуваних MP3-плеєрів є AIMP, який стабільно отримує високі оцінки від користувачів — у середньому 4.5 з 5. Його перевагою є підтримка різних операційних систем, зокрема Windows, Linux і macOS, що робить програму універсальною для широкого кола користувачів.

AIMP вирізняється простим процесом інсталяції та інтуїтивно зрозумілим налаштуванням інтерфейсу. Користувачі можуть обирати мову, зовнішній вигляд плеєра та змінювати інші параметри інтерфейсу.Завдяки широким можливостям, плеєр дозволяє не просто слухати музику, а й організовувати треки за обраними параметрами, об’єднувати їх у групи та редагувати списки відтворення. Вбудовані звукові ефекти, налаштування гучності, режим перемішування композицій та візуалізація сигналу значно покращують користувацький досвід.

У свою чергу, WinAMP Lite — це спрощена версія популярного програвача, яка теж має позитивні відгуки. Застосунок встановлюється без затримок і сумісний з Windows та macOS, хоча й має певні функціональні обмеження. Зокрема, вона не підтримує українську мову, що може ускладнити її використання деяким користувачам. Інтерфейс плеєра виглядає застарілим, не передбачає змін дизайну та не масштабується. Додавання треків до списку вимагає більше дій, а сортування композицій не реалізовано. Проте основні функції — керування відтворенням, регулювання гучності, відображення технічних параметрів треку — доступні і працюють стабільно.

Загалом, AIMP забезпечує значно ширший функціонал, більшу зручність у використанні та можливості персоналізації, тому його можна вважати більш сучасним і комфортним вибором серед MP3-плеєрів.

1.3. Формулювання вимог до продукту

Програма повинна підтримувати основні аудіоформати, зокрема MP3 та WAV, що дасть змогу відтворювати файли без необхідності їх конвертації. Користувач повинен мати змогу повноцінно управляти відтворенням: призупиняти музику, змінювати гучність, перемотувати треки та вибирати між різними режимами, зокрема повтором і випадковим порядком.

Окрім цього, важливо, щоб плеєр дозволяв впорядковувати музику за різними ознаками — наприклад, за назвою пісні, виконавцем, жанром або роком випуску. Користувач повинен мати змогу формувати плейлисти, зберігати їх, редагувати, а також легко знаходити потрібні треки через систему пошуку чи фільтрації.

Програма повинна підтримувати роботу на найбільш розповсюджених операційних системах, таких як Windows і MacOS. Серед основних вимог — стабільна та безпомилкова робота, швидка установка, мінімальне навантаження на систему, а також простий і зрозумілий інтерфейс, який буде зручним для користувачів із різним рівнем технічної підготовки. Важливо, щоб програмне забезпечення залишалося надійним і ефективним навіть під час тривалого використання.

ВИСНОВКИ ДО РОЗДІЛУ 1

У першому розділі розглянуто основи створення MP3-плеєра з графічним інтерфейсом і аудіофункціями. AIMP виявився більш гнучким і зручним для користувача, тоді як WinAMP Lite має обмежені можливості та поступається за функціоналом. AAIMP демонструє кращу інтеграцію з потребами користувачів, на відміну від WinAMP Lite, чия функціональність є досить обмеженою. Аналіз допоміг визначити потреби користувачів і сформувати підхід до власного проєкту.

РОЗДІЛ 2
ПРОЕКТУВАННЯ АРХІТЕКТУРИ МР-3 ПЛЕЄРУ

2.1. Структурування алгоритму розв’язання задачі

Розробка MP3-плеєра, передбачає чітке структурування алгоритму, який лежить в основі його функціонування. Це дозволяє забезпечити логічну послідовність дій при реалізації функціоналу, а також створити зручну архітектуру, що полегшує подальший супровід, тестування і масштабування продукту. На цьому етапі визначаються ключові етапи обробки інформації, взаємодія користувача із застосунком та логіка реакції системи на введення даних.

Основна мета цього етапу — побудова алгоритмічної моделі, яка описує весь життєвий цикл взаємодії користувача з MP3-плеєром: від запуску програми до завершення її роботи. Зокрема, потрібно передбачити механізми відкриття та відтворення аудіофайлів, управління списком відтворення, зміни гучності, переходу між треками, а також забезпечити зручний графічний інтерфейс, що дозволить користувачеві інтуїтивно керувати функціями програми.

Алгоритм роботи MP3-плеєра має бути побудований таким чином, щоб забезпечити максимальну продуктивність при мінімальному використанні системних ресурсів. Це особливо важливо з огляду на можливе використання програми на пристроях із середніми або слабкими технічними характеристиками. Тому при розробці архітектури важливо враховувати не лише функціональність, але й ефективність виконання коду, що безпосередньо впливає на швидкість запуску програми, час відкриття аудіофайлів, а також загальну стабільність її роботи.

Після запуску застосунку відкривається головне вікно, у якому користувач може отримати доступ до ключових функцій програми.

Наступним кроком є завантаження музичних файлів, що здійснюється через інтерфейс вибору, який дає змогу обрати один чи кілька MP3-файлів для подальшого прослуховування.

У структурі алгоритму важливу роль відіграє блок управління відтворенням, який повинен забезпечити адекватну реакцію на дії користувача — натискання кнопок «Repeat», «Pause» перемотування треків вперед або назад, зміна гучності. Кожна з цих дій повинна бути коректно оброблена програмною логікою, із відповідним оновленням візуального інтерфейсу (наприклад, зміна іконки кнопки «Repeat» на «Pause» під час відтворення).

Окремо слід виділити реалізацію списку відтворення (playlist), який дозволяє зручно керувати кількома треками. Відповідний функціональний блок має підтримувати операції додавання й видалення аудіотреків, а також забезпечувати їх впорядкування за різними параметрами, такими як назва, тривалість чи дата імпорту. Крім того, він повинен відображати розширену інформацію про кожну композицію, зокрема назву, виконавця й альбом. Важливо, щоб структура списку могла змінюватися без необхідності перезапускати застосунок, тобто у режимі реального часу.

Крім базового функціоналу, алгоритм повинен передбачати обробку виняткових ситуацій, таких як помилкове відкриття файлів іншого формату, відсутність звукового пристрою, збої під час декодування треків або спроба доступу до пошкоджених файлів. Це досягається через реалізацію блоків перевірки помилок (error handling), які виводять повідомлення з поясненням проблеми та, за можливості, пропонують варіанти її вирішення.

На етапі структурування логіки роботи плеєра ефективно впроваджувати підхід, що базується на модульності. Це передбачає окрему реалізацію основних функціональних блоків, таких як інтерфейс користувача, обробка звуку, керування списком треків і система сповіщень, що забезпечує простоту підтримки та розширення проєкту в майбутньому.

Загалом, процес структурування алгоритму створення MP3-плеєра є критично важливим етапом, що визначає ефективність, зручність і надійність

майбутнього програмного продукту. Ретельно продумана алгоритмічна структура дозволяє не лише забезпечити реалізацію необхідного функціоналу, але й формує основу для довготривалої підтримки та розвитку застосунку.

2.2. Алгоритм роботи MP3-плеєра

MP3-плеєр побудовано так, щоб користувач без зайвих зусиль міг слухати улюблену музику. Все починається із запуску програми, коли система завантажує інтерфейс і готує все для роботи — зокрема зчитує з бази даних список треків, які були додпні до плейлисту та створені плейлисти.

Коли користувач додає нові MP3-файли або слухає музику, дані оновлюються в базі — зберігається нова інформація про плейлист (рис. 2.1).

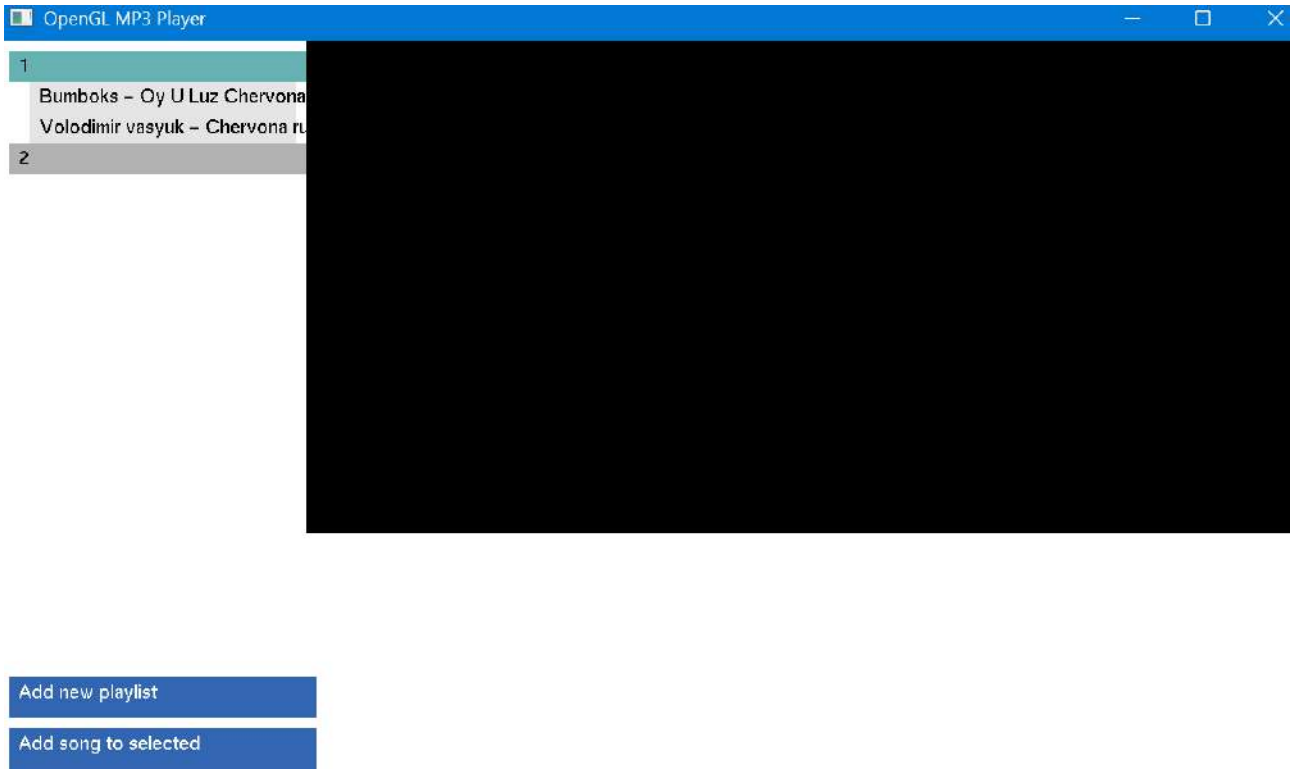

Рис. 2.1. Запуск програми та початковий інтерфейс плеєра

Усі дії — як-то натискання кнопок вправо або влівло для зміни треку чи «Pause», вибір треків та кнопка «Repeat» — одразу обробляються і, за потреби, оновлюються у базі. Це забезпечує зручність: навіть після перезапуску плеєра все залишиться таким, як ви залишили (рис. 2.2).

Рис. 2.2. Кнопки, що вдповідають за відтворення пісень

Далі, дії з додавання пісень до плейлисту та для створення нового плнйлисту відповідають такі кнопки: «Add new playlist» та «Add song to selected» (рис. 2.3).

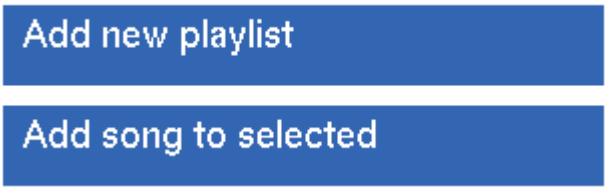

Рис. 2.3. Додавання пісень та створення нового плейлисту

Меню для дій з плейлистом містить все необхідне: виданення, перейменування та додавання до нього музики (рис. 2.4).

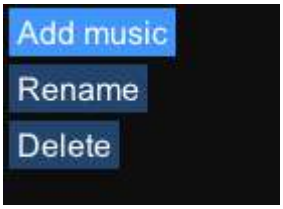

Рис. 2.4. Меню плейлиста

Також своє меню містять і пісні: грати наступною, перейменування та видалення (рис. 2.5).

Рис. 2.5. Меню пісень

Алгоритм роботи MP3-плеєра виглядає наступним чином (рис. 2.6):

Рис. 2.6. Алгоритм роботи MP-3 плеєра

Опис блок-схеми (рис. 2.6) алгоритму роблти MP-3 плеєра:

1. **Початок:** стартова точка виконання програми.
2. **Запуск програми**: ініціалізація програми MP3-плеєра.
3. **Відтворення інтерфейсу та зчитування даних з БД**: відображення графічного інтерфейсу для користувача та завантаження даних про треки та плейлисти з бази даних.
4. **Очікування дій користувача**: програма перебуває в режимі очікування: користувач може взаємодіяти з інтерфейсом.
5. **Обробка подій та оновлення БД**: якщо користувач виконує будь-які дії (додавання пісень, створення нового плейлиста, відалення перейменування, повтор та т.і.), вони обробляються, а зміни фіксуються в базі даних.
6. **Гілки з подіями користувача:** дії з плейлистом (створення, редагування або видплення плейлиста), вибір треку для відтворення (обрання музичного файлу для прослуховування), інші дії користувача (керування відтворенням), редагування інформації про треки.
7. **Завершення**: завершення роботи програми після команди користувача.
8. **Кінець**: кінцева точка алгоритму.

Ця блок-схема демонструє логічну послідовність роботи MP3-плеєра — від початкового запуску до завершення взаємодії. Завдяки такому зображенню легко зрозуміти, що саме відбувається всередині програми та коли саме користувач взаємодіє з інтерфейсом і базою даних.

ВИСНОВКИ ДО РОЗДІЛУ 2

Загалом, можна підсумувати, що проєктування архітектури MP3-плеєра потребує чіткого планування та чіткого поділу на функціональні частини. У цьому розділі було визначено ключові складові програми, описано, як користувач взаємодіє з інтерфейсом, а також окреслено методи роботи з аудіофайлами і організації списку відтворення. Головну увагу приділено

використанню модульної архітектури, яка забезпечує адаптивність і зручність у підтримці програми. Такий підхід гарантує можливість подальшого розвитку програми без необхідності серйозних змін у загальній архітектурі. Запропонований алгоритм розв’язання задачі стане надійною основою для створення ефективного та зручного MP3-плеєра, який відповідатиме сучасним стандартам програмного забезпечення

У розділі також проаналізовано послідовність роботи MP3-плеєра — від моменту запуску до виконання основних функцій. Детально описано, як система реагує на дії користувача: додавання нових композицій, створення й редагування списків відтворення, керування музикою та збереження поточного стану навіть після закриття програми. Це забезпечує стабільність функціонування та зручність використання програми. Завдяки цьому програмний продукт стає інтуїтивно зрозумілим і надійним у повсякденному користуванні.

РОЗДІЛ 3
РОЗРОБКА БАЗИ ДАНИХ ЗАДАЧІ

3.1. Вибір СУБД

У процесі створення програмного забезпечення важливо обрати таку систему управління базами даних (СУБД), яка буде найкраще відповідати потребам конкретного проєкту. Від СУБД яка буде обрана, значною мірою залежить реалізація програмного рішення, зокрема його розвиток, адаптація до зростання даних та зручність експлуатації.

Під час аналізу існуючих рішень було враховано декілька ключових факторів:

- Обсяги даних, з якими буде працювати система.
- Складність запитів, які необхідно буде реалізовувати.
- Технологічне середовище.
- Автономна робота.
- Швидкість розробки та легкість інтеграції СУБД у програму.

Зважаючи на ці аспекти, було вирішено використати SQLite — легку, вбудовану реляційну СУБД, яка широко використовується в мобільних застосунках, настільних програмах, вбудованих системах, а також у прототипах і невеликих проєктах. Одна з ключових переваг полягає в тому, що вся база розміщується у звичайному файлі, що дозволяє обійтися без окремого серверного ПЗ, складних конфігурацій чи адміністративних дій.

SQLite повністю написана на мові C, що дозволяє безпосередньо працювати з нею в проєктах на C або C++. Завдяки цьому значно спрощується процес підключення СУБД до програми, а також зникає потреба у використанні додаткових бібліотек чи адаптерів. Для даного проєкту, реалізованого на C++, це стало вагомою перевагою, адже дозволяє ефективно працювати з базою даних без перевантаження коду сторонніми залежностями.

Ще одним важливим аргументом на користь SQLite стала її відкрита ліцензія (Public Domain), що дозволяє використовувати її безкоштовно навіть у комерційних проєктах.

З технічної точки зору, SQLite забезпечує базову підтримку транзакцій, обмежень цілісності, індексів, а також підтримує стандартний синтаксис SQL, що значно полегшує написання запитів. Для розв’язання завдань, що ставляться перед даною інформаційною системою, цих можливостей цілком достатньо.

Таким чином, використання SQLite дозволяє досягти наступного:

- Спрощення розробки й налагодження системи.
- Зменшення технічних витрат на супровід.
- Швидкий запуск без додаткового налаштування серверного середовища.
- Підвищення портативності застосунку — його можна переносити разом із файлом бази даних без втрати працездатності.

Загалом, SQLite стала оптимальним вибором для реалізації бази даних у межах цього проєкту — як з погляду функціональних можливостей, так і з огляду на простоту використання та відповідність проєктним вимогам.

3.2. Проектування структури бази даних

База даних Open MP3 містить лише 2 таблиці. Почнемо з першої таблиці Таблиця 3.1 «playlist_song» відповідає за зберігання інформації про пісні, що входять в створені плейлисти.

Таблиця 3.1

Опис складових елементів об’єкта «playlist_song»

Назва атрибута	Тип даних	Розмір	Опис
id	INTEGER	4	Унікальний ідентифікатор кожного запису

Продовження табл. 3.1

playlist_id	INTEGER	4	Ідентифікатор плейлиста до якого на лежитт пісня
title	TEXT	до 1 Гб (змінна довжина)	Назва пісні
path	TEXT	до 1 Гб (змінна довжина)	Шлях до пісні

Наступна друга таблиця Таблиця 3.2 «playlists» відповідає за збереження інформації про плейлисти.

Таблиця 3.2

Опис складових елементів об’єкта «playlists»

Назва атрибута	Тип даних	Розмір	Опис
id	INTEGER	4	Унікальний ідентифікатор кожного плейлисту
title	TEXT	до 1 Гб (змінна довжина)	Назва плейлиста

Тож, база даних музичного програвача для ПК добре сформована та логічно створена для гарної роботи Open MP3.

ВИСНОВКИ ДО РОЗДІЛУ 3

У цьому розділі було зроблено важливі кроки для створення музичного програвача — обрано зручну систему управління базами даних та продумано її структуру. Після розгляду різних варіантів зупинилися на SQLite. Це легка,

проста у використанні СУБД, яка добре підходить для проєктів такого масштабу. Створена база даних не потребує окремого сервера і працює напряму з файлом, її легко можна підключити до програми.

Було вирішено створити дві таблиці для збереження даних пісень та плейлистів. Це дозволяє зробити легку структуру, яка в свою чергу охоплює необхідне, наприклад, назва треку, плейлиста, шляхи до файлів. Усе просто й логічно.

Обрана база не тільки відповідає технічним потребам, а ще й дозволяє гнучко працювати з даними. Якщо буде необхідність, то можна легко додати нові поля чи функції без великих змін.

У підсумку можна сказати, що вибір СУБД і структура бази даних повністю відповідають цілям проєкту. Це рішення допомогло закласти надійну основу для подальшої розробки та зручної роботи застосунку.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Загальна характеристика програмного забезпечення

Перш за все перед переглядом самого програмного забезпечення, треба почати з опису та перегляду основних класів плеєра, що забезпечують гарну та стабільну роботу програми. Клас «Audio» [додаток А, с.53] відповідає за відтворення аудіофайлів у форматі MP3 в межах мультимедійної програми. Його основне завдання — підготувати аудіофайл до програвання, запустити відтворення, обробити паузу, зупинку або перемотку, а також синхронізувати потік відтворення з буферами і станом програвача.

Після створення об’єкта класу, він приймає і зберігає основну інформацію про аудіофайл: шлях до файлу, його назву, унікальний ідентифікатор. Назва при цьому транслітерується для збереження у базі даних.

Після того, як користувач запускає аудіо, об'єкт ініціалізує спеціальні бібліотеки для роботи з MP3, визначає характеристики файлу (кількість каналів, частоту, тривалість), формує буфери, заповнює їх даними та починає відтворення. Під час фонового режиму запускається окремий потік, який постійно слідкує за станом аудіопотоку та оновлює інформацію про поточну позицію відтворення.

Якщо відтворення зупинено або завершено, ресурсні об'єкти (джерело звуку, буфери тощо) коректно очищаються. У випадку перемотки або перезапуску обробка організована так, щоб уникнути збоїв і пошкоджених даних.

«Audio» клас підтримує зміну назви треку з оновленням у базі даних, а через доступні методи можна дізнатись ідентифікатор треку, шлях до файлу, назву, загальну тривалість, час, який уже було відтворено, а також визначити, чи завершилось відтворення. Об'єкти цього класу можна порівнювати між собою за шляхом до файлу і назвою.

Ініціалізація аудіо, підготовка до відтворення:

```
void Audio::Initialize()
{
    mpg123_init();
    mh = mpg123_new(NULL, NULL);

    if (mpg123_open(mh, path.c_str()) != MPG123_OK) {
        std::cout << "mpg123_open error: " << mpg123_strerror(mh) << std::endl;
        return;
    }

    mpg123_getformat(mh, &rate, &channels, &encoding);
    ALenum format = (channels == 2) ? AL_FORMAT_STEREO16 : AL_FORMAT_MONO16;

    mpg123_format_none(mh);
    mpg123_format(mh, rate, channels, encoding);

    off_t samples = mpg123_length(mh);
    if (samples > 0) {
        duration = static_cast<int>(samples / rate);
    }
    else {
        duration = 0;
    }

    mpg123_seek(mh, 0, SEEK_SET);

    alGenSources(1, &source);
    alGenBuffers(NUM_BUFFERS, buffers);

    unsigned char tempBuffer[AUDIO_BUFFER_SIZE];
    size_t done;
    for (int i = 0; i < NUM_BUFFERS; ++i) {
        if (mpg123_read(mh, tempBuffer, AUDIO_BUFFER_SIZE, &done) == MPG123_OK) {
            alBufferData(buffers[i], format, tempBuffer, done, rate);
        }
    }

    alSourceQueueBuffers(source, NUM_BUFFERS, buffers);
    alSourcePlay(source);
    this->state = AudioState::PLAYING;

    ALint state;
```

```
    alGetSourcei(source, AL_SOURCE_STATE, &state);
    if (state != AL_PLAYING) {
        this->state = AudioState::STOPPED;
        std::cout << "Source state: " << state << std::endl;
        ClearInstance();
    }

    streamingThread = std::thread(&Audio::StreamLoop, this);
}
```

Відтворення аудіо:

```
void Audio::Play()
{
    if (state == AudioState::STOPPED || state == AudioState::INITIAL) {
        Initialize();
    }

    if (streamingThread.joinable()) return;

    alSourcePlay(source);
    state = AudioState::PLAYING;

    streamingThread = std::thread([this]() { StreamLoop(); });
}
```

Зупинка відтворення і очищення ресурсів:

```
void Audio::Stop()
{
    state = AudioState::STOPPED;

    duration = position = 0;
    ClearInstance();
}

void Audio::ClearInstance()
{
    alDeleteSources(1, &source);

    ALint queued = 0;
    alGetSourcei(source, AL_BUFFERS_QUEUED, &queued);
    while (queued--) {
        ALuint buf;
        alSourceUnqueueBuffers(source, 1, &buf);
    }
```

```
    alDeleteBuffers(NUM_BUFFERS, buffers);

    if (streamingThread.joinable()) {
        streamingThread.join();
    }

    if (!mh) {
        return;
    }

    mpg123_close(mh);
    mpg123_delete(mh);
    mpg123_exit();
}
```

Тепер буде розглянуто таблицю з коротким описом усіх змінних та методів класу «Audio» (Таблиця 4.1).

Таблиця 4.1

Опис складових елементів класа «Audio»

Назва	**Тип**	**Опис**
id	int	Ідентифікатор аудіозапису
path	std::string	Шлях до аудіофайлу
title	std::string	Назва треку
state	AudioState	Поточний стан (INITIAL, PLAYING, PAUSED, STOPPED)
duration	int	Загальна тривалість треку у секундах
position	int	Поточна позиція відтворення у секундах
channels	int	Кількість каналів у аудіо (1 — mono, 2 — stereo)
encoding	int	Формат кодування (визначається mpg123)
rate	long	Частота дискретизації
format	ALenum	Формат OpenAL буфера (моно/стерео)
source	ALuint	Ідентифікатор джерела OpenAL
buffers	ALuint[NUM_BUFFERS]	Масив буферів OpenAL
mh	mpg123_handle*	Дескриптор mpg123
streamingThread	std::thread	Потік для потокового відтворення

Продовження таблиці 4.1

restartPending	bool	Прапорець перезапуску потоку
streamMutex	std::mutex	М’ютекс для синхронізації потоків
streamCV	std::condition_variable	Змінна умови для синхронізації
waveformSamples	std::vector<short>	Зразки для візуалізації хвилі звуку
waveformMutex	std::mutex	М’ютекс для доступу до зразків хвилі

Далі, опис класу «Playlist» [додаток Е, с.76] відповідає за створення, збереження та керування списком відтворення музичних файлів. Тоді коли створюється новий об’єкт цього класу, він автоматично додається до бази даних і отримує унікальний ідентифікатор. Якщо ж об’єкт ініціалізується вже з наявним id, тоді вважається, що цей список вже існує в системі, і його не потрібно додатково зберігати.

Створення нового списку з автоматичним збереженням у базі даних та отриманням ID:

```
Playlist::Playlist(std::string title)
    : id(-1), queueEnterIndex(-1)
{
    Db* db = Db::GetInstance("music_db.db");

    sqlite3_stmt* stmt;
    sqlite3_prepare_v2(db->GetDb(), "INSERT INTO playlists (title) VALUES (?);", -1,
&stmt, nullptr);
    sqlite3_bind_text(stmt, 1, title.c_str(), -1, SQLITE_TRANSIENT);

    if (sqlite3_step(stmt) == SQLITE_DONE) {
        int64_t newId = sqlite3_last_insert_rowid(db->GetDb());
        this->id = newId;
        this->title = std::move(title);
    }

    sqlite3_finalize(stmt);
}
```

Ініціалізація з уже наявним ID:

```
Playlist::Playlist(int id, std::string title)
    : id(id), title(title), queueEnterIndex(-1)
{
}
```

До списку можна додавати аудіофайли кількома способами: або просто передаючи шлях і назву файлу, або використовуючи вже створений об’єкт Audio. Отже, якщо аудіофайл уже присутній у списку, система не дозволить додати його повторно — це контролюється за допомогою методу `Include`.

Додавання файлів за шляхом і назвою та додавання вже створеного об’єкта:

```
void Playlist::Add(std::string path, std::string title)
{
    Db* db = Db::GetInstance("music_db.db");

    sqlite3_stmt* stmt;
    sqlite3_prepare_v2(db->GetDb(), "INSERT INTO playlist_songs (playlist_id, title, path) VALUES (?, ?, ?);", -1, &stmt, nullptr);

    sqlite3_bind_int(stmt, 1, id);
    sqlite3_bind_text(stmt, 2, path.c_str(), -1, SQLITE_TRANSIENT);
    sqlite3_bind_text(stmt, 3, title.c_str(), -1, SQLITE_TRANSIENT);

    if (sqlite3_step(stmt) == SQLITE_DONE) {
        int64_t newId = sqlite3_last_insert_rowid(db->GetDb());
        audioList[newId] = std::make_shared<Audio>(newId, std::move(path), std::move(title));
    }

    sqlite3_finalize(stmt);
}
void Playlist::Add(std::shared_ptr<Audio> audio)
{
    if (Include(audio->GetId())) {
        return;
    }

    Db* db = Db::GetInstance("music_db.db");

    sqlite3_stmt* stmt;
    sqlite3_prepare_v2(db->GetDb(), "INSERT INTO playlist_songs (playlist_id, title, path) VALUES (?, ?, ?);", -1, &stmt, nullptr);

    sqlite3_bind_int(stmt, 1, id);
    sqlite3_bind_text(stmt, 2, audio->GetPath().data(), -1, SQLITE_TRANSIENT);
```

```
        sqlite3_bind_text(stmt, 3, audio->GetTitle().data(), -1, SQLITE_TRANSIENT);

        if (sqlite3_step(stmt) == SQLITE_DONE) {
            int64_t newId = sqlite3_last_insert_rowid(db->GetDb());
            audioList[newId] = std::move(audio);
        }

        sqlite3_finalize(stmt);
    }
```

Також у класі реалізовано механізм черги — тимчасового списку аудіофайлів, які потрібно відтворити в першу чергу. Якщо черга порожня, то метод Next повертає наступний елемент зі звичайного списку, або переходить на перший, якщо кінець досягнуто. Аналогічно працює Prev, тільки в зворотному напрямку.

Назву списку можна змінити, і ця зміна автоматично оновлюється в базі даних. Треба звернути увагу на те, що назва проходить через функцію транслітерації, яка, ймовірно, забезпечує збереження її у більш зручному для системи форматі.

Передбачені й методи для видалення: можна прибрати окремий аудіофайл із пам’яті або з бази даних повністю. Якщо ж потрібно стерти весь список — метод Delete видаляє як сам список, так і всі пов’язані з ним пісні.

Видалення одного файлу з пам’яті:

```
    void Playlist::Remove(int id)
    {
        audioList.erase(id);
    }

    void Playlist::Delete(int id)
    {
        Db* db = Db::GetInstance("music_db.db");

        sqlite3_stmt* stmt;
        sqlite3_prepare_v2(db->GetDb(), "DELETE FROM playlist_songs WHERE id = ?", -1,
&stmt, nullptr);
        sqlite3_bind_int(stmt, 1, id);
        sqlite3_step(stmt);
        sqlite3_finalize(stmt);
```

```
    Remove(id);
}
```

Додатково можна отримати заголовок, кількість треків, перевірити наявність конкретного за ідентифікатором, або ж отримати конкретний об’єкт Audio за його назвою чи id. Доступ до списку реалізовано за допомогою ітераторів та перевантажених операторів квадратних дужок.

Ітератори:

```
std::unordered_map<int, std::shared_ptr<Audio>>::iterator Playlist::begin()
{
    return audioList.begin();
}

std::unordered_map<int, std::shared_ptr<Audio>>::iterator Playlist::end()
{
    return audioList.end();
}
```

Загалом цей клас надає повний набір інструментів для роботи з музичними плейлистами, забезпечуючи як інтеграцію з базою даних, так і зручну роботу з даними в оперативній пам’яті. Розглянемо детально клас за допомогою таблиці з коротким описом зміних класу (Таблиця 4.2).

Таблиця 4.2

Опис складових елементів класа «Playlist»

Назва	**Тип**	**Розмір (байт)**	**Призначення**
id	int	4	Ідентифікатор плейліста у базі даних
title	std::string	24+	Назва плейліста
queueEnterIndex	int	4	Індекс елемента, з якого починається черга відтворення

Продовження таблиці 4.2

audioList	std::unordered_map<int, std::shared_ptr<Audio>>	залежить від розміру	Список аудіофайлів у плейлісті (ключ — ідентифікатор, значення — об’єкт Audio)
queue	std::vector<std::shared_ptr<Audio>>	залежить від кількості	Черга відтворення аудіофайлів у плейлісті

Наступний розглянути клас «db» [додаток В, с.62] відповідає за з’єднання з базою даних SQLite і працює за принципом «єдиного екземпляра». Програма створює лише один об’єкт цього класу, і надалі всі операції з базою виконуються через нього.

Ініціалізація єдиного екземпляра класу (Singleton):

```
Db* Db::instance = nullptr;

Db* Db::GetInstance(std::string path) {
    if (instance == nullptr) {
        instance = new Db(path);
    }
    return instance;
}
```

Шлях до файлу бази даних передається під час створення об'єкта класу. Далі викликається метод «Initialize», який відкриває з’єднання. Після встановленого з’єднання, повторне відкриття не відбувається. У випадку помилки при відкритті база не підключається, і в консоль виводиться повідомлення. За умови, якщо об'єкт ще не створено, він створюється з вказаним шляхом.

Конструктор класу та ініціалізація з’єднання з базою даних та метод «Initialize» — відкриття з’єднання з базою, якщо ще не підключено:

```
Db::Db(std::string path)
    : path(std::move(path)) {
    Initialize();
```

```
}
void Db::Initialize() {
    if (connected) {
        return;
    }

    int rc = sqlite3_open(path.c_str(), &db);
    connected = true;

    if (rc != SQLITE_OK) {
        std::cerr << "Cannot open DB: " << sqlite3_errmsg(db) << "\n";
        connected = false;
        return;
    }
}
```

Для виконання SQL-запитів використовується метод «Query». Він перевіряє, чи підключено базу, і намагається виконати запит. Помилка з'являється із описом проблеми.

Метод Query:

```
bool Db::Query(std::string query) {
    Initialize();
    if (!connected) {
        std::cout << "Cannot open DB: " << sqlite3_errmsg(db) << "\n";
        return false;
    }

    const int rc = sqlite3_exec(db, query.c_str(), nullptr, nullptr, nullptr);
    if (rc != SQLITE_OK) {
        std::cerr << "Query failed: " << sqlite3_errmsg(db) << "\n";
        return false;
    }

    return true;
}
```

Також клас має метод «CLose», який закриває підключення, метод IsConnected, що перевіряє стан з’єднання, «GetDb» — повертає вказівник на об’єкт бази даних, і «GetPath», який повертає шлях до файлу бази.

Метод GetDb та метод GetPath:

sqlite3*& Db::GetDb() {

```
    return db;
}
std::string_view Db::GetPath() {
    return path;
}
```

І останнє, розглянемо кортких опис змінних розглянутого класу, що відповідає за усю роботу з базою даних (Таблиця 4.3).

Таблиця 4.3

Опис складових елементів класа «db»

Назва	**Тип**	**Розмір (байт)***	**Призначення**
instance	static Db*	8	Статичний єдиний екземпляр класу (Singleton)
path	std::string	32–40	Шлях до файлу бази даних
db	sqlite3*	8	Вказівник на об'єкт бази SQLite
connected	bool	1	Статус з'єднання з базою

4.2. Опис інсталювання Open MP3 на ПК

Інсталювати музичной плеєр Open MP3 можна завантажити з сайту візитка, на якому детально написано про усі переваги та системні вимоги для коректного встановлення на ПК. Після переходу на сайт можна побачити яскравий банер з назвою та короткою візиткою плеєру, яка привертає увагу користувача (рис. 4.1).

Рис. 4.1. Банер сайту-візитка Open MP3

Прогорнувши трошки нижче йде детальний опис програми (рис. 4.2).

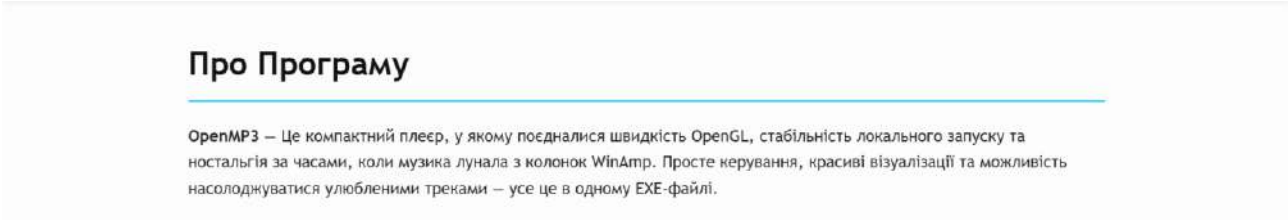

Рис. 4.2. Опис програми Open MP3

Кожен підпункт на сайті-візитці розділений синьою тоненькою лінією, завдяки цьому все на сайти виглядає структуровано та ненагромаджено. Наступний пункт про вміння плеєру розділений блоками, кожна інформація міститься в окремому блоці, що дозволяє чітко побачити функції програвача (рис. 4.3).

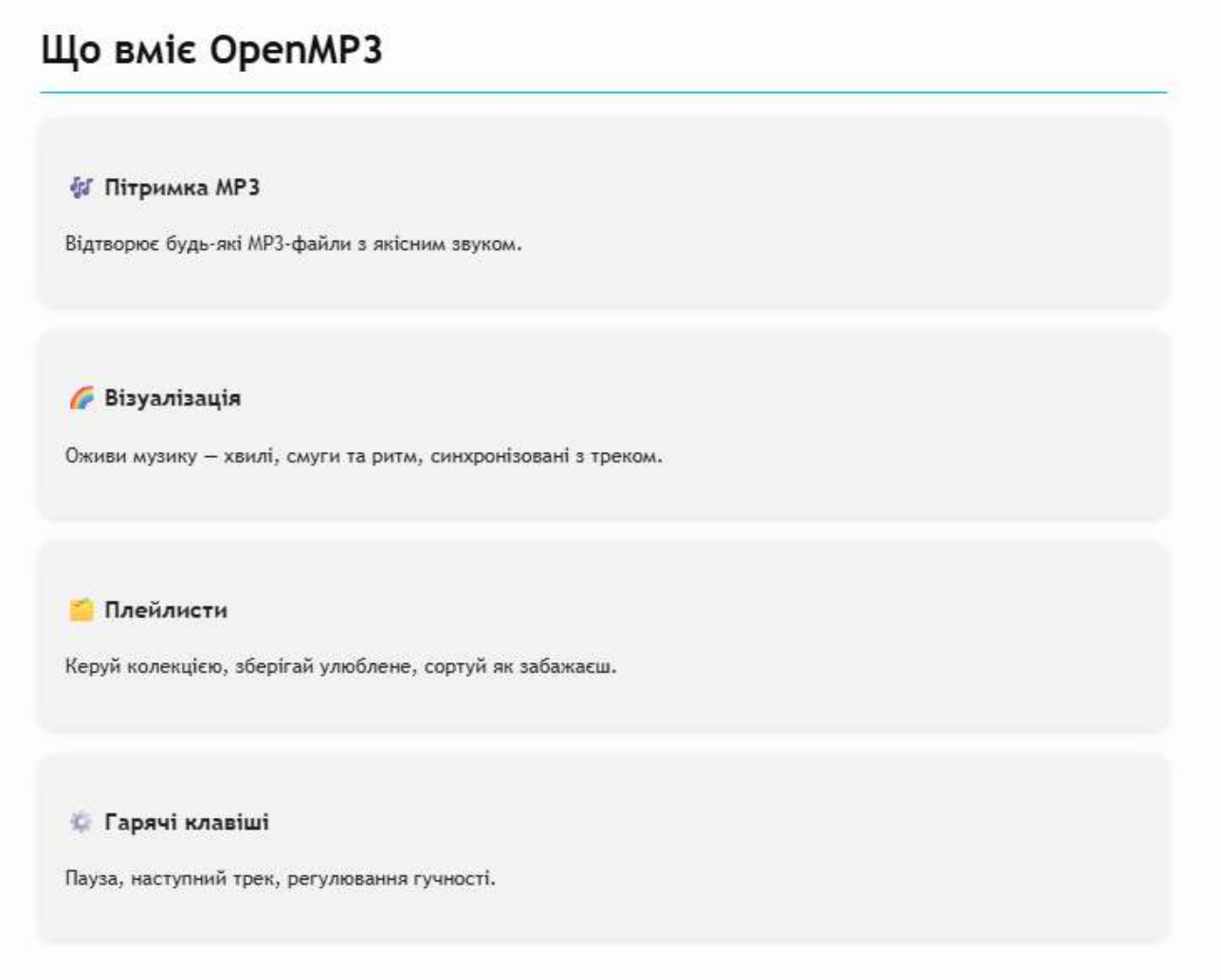

Рис. 4.3. Опис можливостей плеєра

Далі, можна побачити, інтерфейс самого музичного програвача, це надає можливість не в сліпу інсталювати додаток, а одразу побачити, як він буде виглядати під час роботи (рис. 4.4).

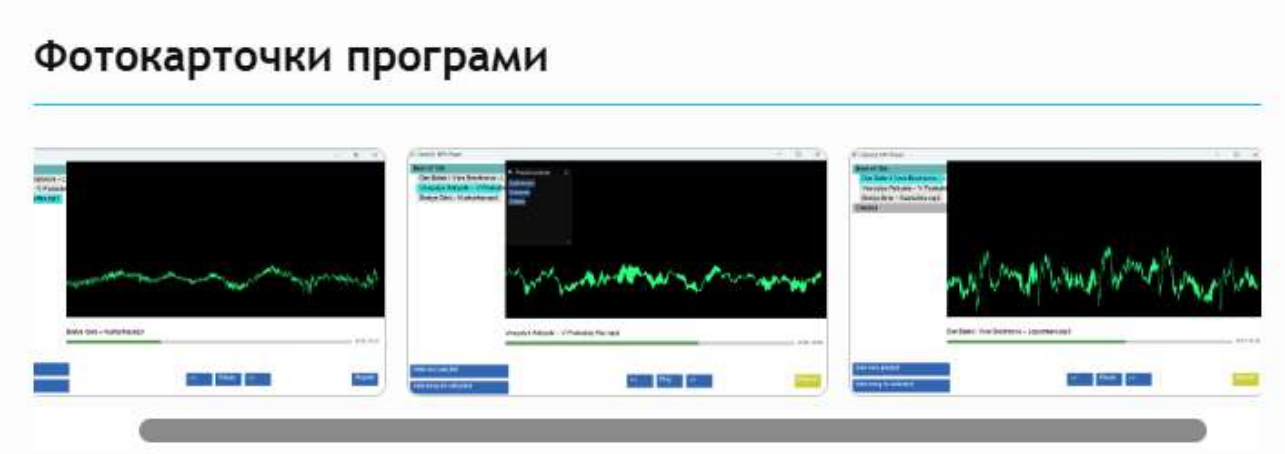

Рис. 4.4. Представлений інтерфейс програвача на сайті-візитка

А вже після всього опису та інтерфейсу йдуть системні вимоги та кнопка для скачування інсталятора. Кнопка виглядає яскравою та привертає увагу користувача для подальшого завантаження (рис. 4.5).

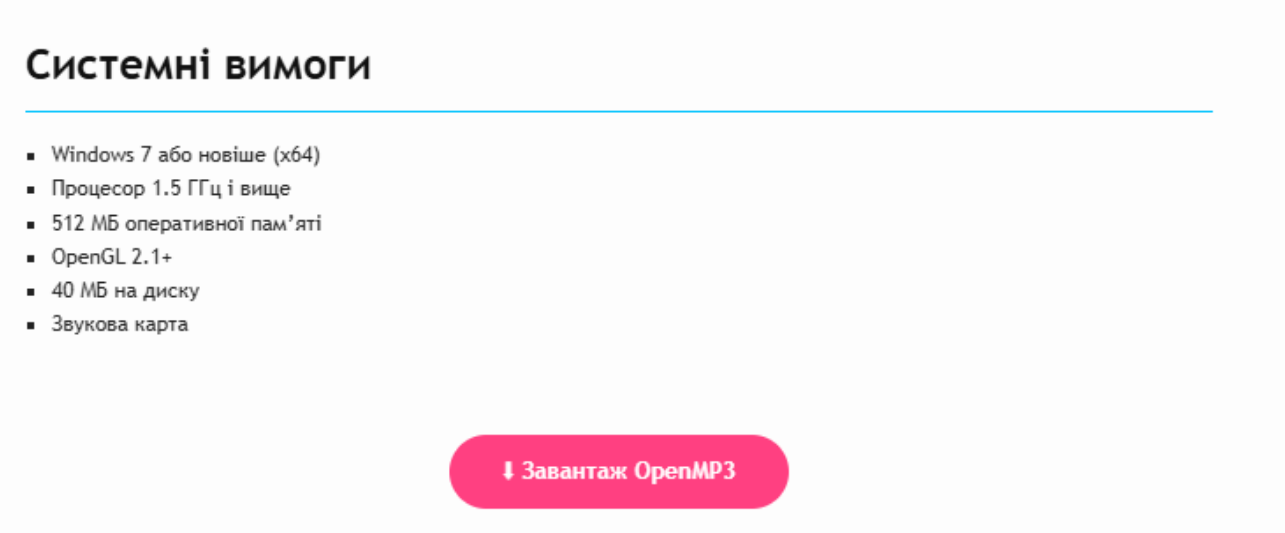

Рис. 4.5. Системні вимоги та кнопка для завантаження інсталятора

Завантаження інсталятора відбувається швидко, одразу після натискання на кнопку для завантаження, інсталятор уже є в завантаженнях. Після натиску на інсталятор з'являється діалогове вікно. Кнопка «Обзор» відповідає за вибір місця, де буде інстальований музичний програвач, кнопка «Установка» відповідає за встановлення додатка після вибору місця (рис. 4.6).

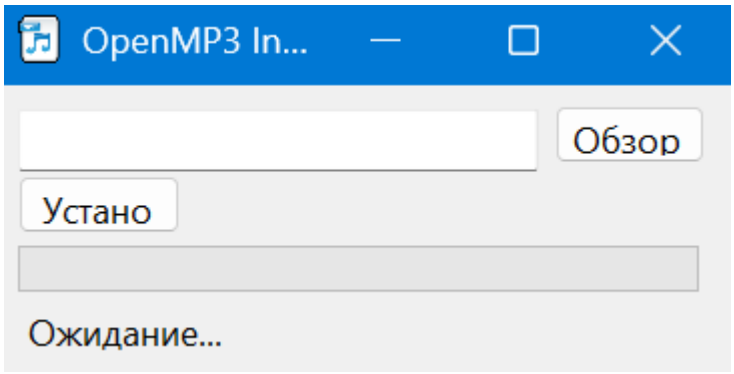

Рис. 4.6. Діалогове вікно

Після успішного інсталювання з’являється діалогове вікно з повідомленням про це (рис. 4.7).

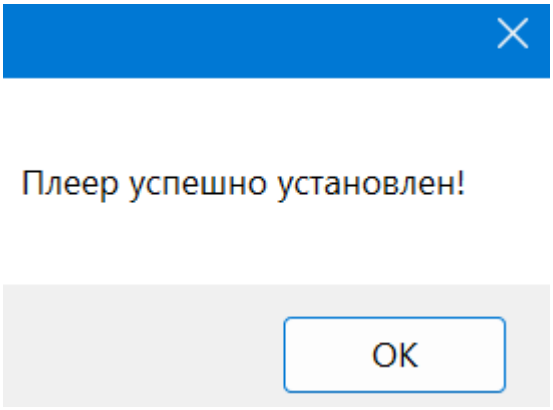

Рис. 4.7. Діалогове вікно про успішне встановлення

Все швидко та якісно працює та що не менш головне, привабливо виглядає.

4.3. Опис музичного плеєра Open MP3

До плеєру поки не додано пісень та не створено ніяких плейлистів. Для додавання пісні до обраного та створення плейлистів є дві відповідні кнопки (рис. 4.8).

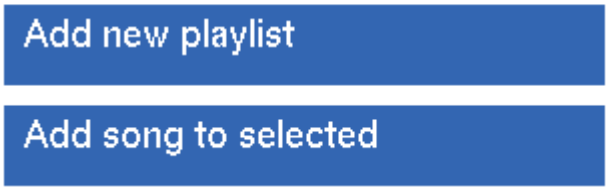

Рис. 4.8. Головні кнопки в плеєрі

При натискані на кнопку для створення плейлисту з’являється віконечко для вписування назви плеєру (рис. 4.9).

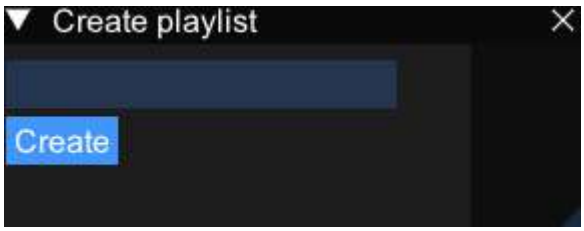

Рис. 4.9. Вікно для написання назви плейлисту

Після створення плейлисту, для додавання пісні треба натиснути на праву кнопку миші, та вибрати потрібну функцію (додати музику, перейменувати або видалити) для обраного плейлисту (рис. 4.10).

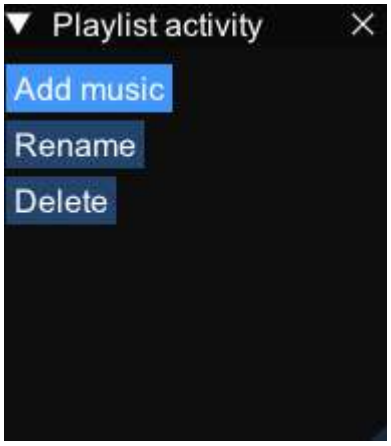

Рис. 4.10. Дії з плейлистом

Вибрано було додати музику, і одразу з’являється вікно з файлами та можливістю обрати потрібні пісні для цього плейлисту. Після обрання пісень внизу під назвою з’явився стовпчик з доданими піснями (рис. 4.11).

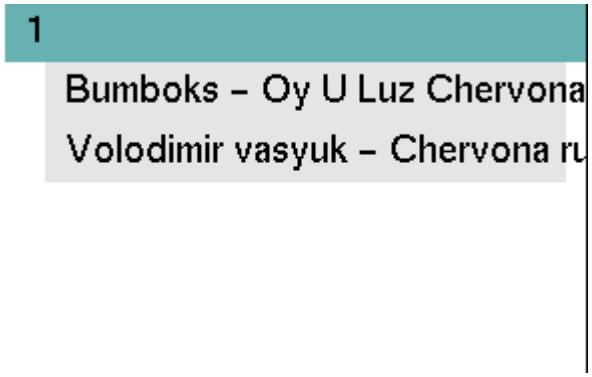

Рис. 4.11. Додані пісні до плейлисту

Тепер можна ввімкнути в програвачі одну з пісень. На чорному фоні візуалізується зелена звукова хвиля, яка динамічно змінюється від гучності та

частоти композиції, виглядає ефектно. Далі, у нижні частині після чорного екрану, розташована панель керування, де можна побачити виконавця та назву пісні, потім йде індикатор часу та тривалість треку. В самому низу музичного програвача знаходяться кнопки для керування: «Пауза», стрілки перемотки композицій та «Повтор» (рис. 4.12).

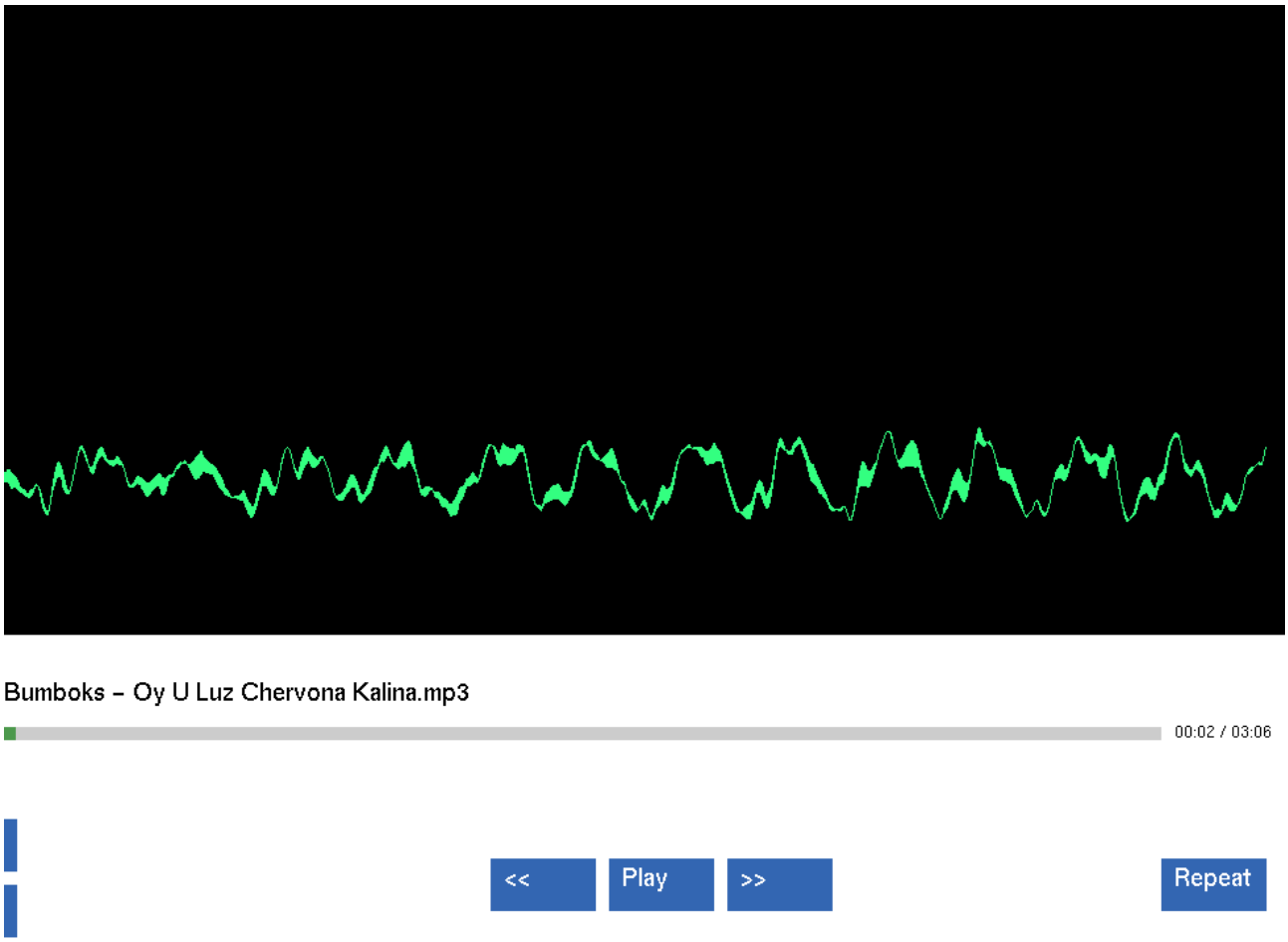

Рис. 4.12. Інтерфейс плеєра в роботі

Також можна зазначити, що пісні також мають свої функції для керування ним: видалення, перейменування та поставити в чергу наступною (рис. 4.13).

Рис. 4.13. Меню для керування піснями

Плеєр має стабільну роботу, гарне звучання. Не менш важливо те, що музичний програвач має зрозумілий та простий інтерфейс.

ВИСНОВКИ ДО РОЗДІЛУ 4

У цьому розділі було детально розглянуто структуру та логіку роботи ключових складових програмного забезпечення — класів Audio, Playlist та Db. Клас `Audio` призначений для відтворення аудіофайлів, обробки буферів, управління станом програвача та синхронізації потоків. Його реалізація забезпечує стабільну роботу з MP3-файлами та інтеграцію з OpenAL.

Клас Playlist реалізує зручний механізм керування списками відтворення, включно з додаванням і видаленням треків, підтримкою черги та взаємодією з базою даних. Особливу увагу приділено унікальності аудіофайлів у списку та гнучкій роботі з об’єктами Audio.

Окремо було проаналізовано клас Db, що реалізує підключення до бази даних SQLite за шаблоном Singleton. Його функціональність забезпечує стабільне збереження даних і ефективне виконання SQL-запитів.

Загалом, структура програмного забезпечення побудована логічно, з чітким розподілом обов’язків між класами, що створює передумови для надійної та масштабованої роботи програми.

Також було розглянуто процес встановлення та функціонування музичного програвача Open MP3. Завантаження інсталятора для подальшого встановлення плеєру на викликає ніких трудновів, все відбувається швидко та зрозуміло.

Інтерфейс плеєра продуманий до дрібниць — він поєднує в собі зручність, привабливий вигляд і необхідний функціонал. Користувач має можливість створювати власні плейлисти, додавати до них пісні та насолоджуватись якісним відтворенням музики. Завдяки простоті, швидкій роботі та зрозумілому дизайну Open MP3 можна взяти за приклад, як гарне програмне забезпечення.

ВИСНОВКИ

Пройдено повний цикл створення програмного продукту — від ідеї до реалізації зручного застосунку для прослуховування музики. Було обрано сучасні та ефективні інструменти: мову програмування C++ для логіки програми, бібліотеки OpenGL — для графічного інтерфейсу, а також OpenAL — для якісного аудіозвучання. Такий вибір дав змогу поєднати гнучкість у дизайні з високою продуктивністю програвача.

Особливу увагу приділено зручності користувача. Інтерфейс плеєра вийшов зрозумілим і простим у користуванні, а навігація — зручною та логічно побудованою. Передбачено основні функції, які очікує сучасний користувач: відтворення треків, керування гучністю, формування плейлистів, перемикання між піснями тощо. Реалізовано навіть інсталяційний модуль, що дозволяє без зайвих складнощів встановити програму на комп’ютер.

Для збереження інформації про музику та плейлисти використано просту, але надійну базу даних на основі SQLite. Це рішення добре себе зарекомендувало як практичне для невеликих проєктів, не потребує окремого сервера й легко підключається до основної програми.

Окрім технічного боку, було подбано і про візуальне представлення проєкту — створено лендінг-сторінку, де потенційні користувачі можуть ознайомитися з можливостями плеєра та швидко його завантажити.

У результаті можна стверджувати, що всі поставлені завдання успішно виконано. Результатом стала цілісна, зручна у використанні та технічно продумана програма. Вона може стати як гарною основою для подальших удосконалень, так і готовим продуктом для щоденного використання. Здобутий під час роботи досвід є цінним і стане у пригоді для майбутніх професійних проєктів у галузі програмування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Зеленський А. С., Лисенко В. С., Баран С. В. Методичні вказівки до виконання лабораторних та індивідуальних робіт на основі прикладів розробки програмного забезпечення у Visual C++ 6.0 / Криворізький економічний інститут ДВНЗ «КНЕУ імені Вадима Гетьмана». — Кривий Ріг: КЕІ, 2007. — 63 с.
2. Зеленський А. С., Лисенко В. С., Баран С. В. Методичні вказівки для самостійного вивчення роботи з базами даних на Visual C++ з використанням об’єктів ActiveX Data Object (ADO) / Криворізький економічний інститут ДВНЗ «КНЕУ імені Вадима Гетьмана». — Кривий Ріг: КЕІ, 2008. — 54 с.
3. Бондар І. О. Основи розробки програмних застосунків: навчальний посібник / І. О. Бондар. — Львів: ЛНУ імені Івана Франка, 2020. — 128 с.
4. Глінка І. Ю. Проєктування інтерфейсів користувача: теорія та практика / І. Ю. Глінка. — Київ: Ліра-К, 2018. — 144 с.
5. Соммервіл І. Інженерія програмного забезпечення / Пер. з англ. — 10-те видання. — К.: Вільямс, 2018. — 848 с.
6. Чепурний С. В. Розробка програмного забезпечення: методичні рекомендації до дипломного проєктування / С. В. Чепурний. — Харків: ХНУРЕ, 2019. — 56 с.
7. Кірсанов С. М. Основи алгоритмізації та програмування: навчальний посібник. — Вінниця: ВНТУ, 2017. — 212 с.
8. Дейкстра Е. Дисципліна програмування / Е. Дейкстра. — М.: Мир, 1984. — 234 с.
9. Таненбаум А. С. Архітектура комп’ютерних систем / А. С. Таненбаум. — К.: Діалектика, 2016. — 752 с.
10. Кнут Д. Мистецтво програмування. Т. 1: Основні алгоритми / Д. Кнут. — К.: Вільямс, 2019. — 624 с.

ДОДАТКИ

Додаток А

Лістинг програмного коду класу «Audio»

```
#include "Audio.h"
#include <vector>
#include <iostream>
#include "AudioController.h"
#include "translit.h"
#include "db.h"

void Audio::Initialize()
{
    mpg123_init();
    mh = mpg123_new(NULL, NULL);

    if (mpg123_open(mh, path.c_str()) != MPG123_OK) {
        std::cout << "mpg123_open error: " << mpg123_strerror(mh) << std::endl;
        return;
    }

    mpg123_getformat(mh, &rate, &channels, &encoding);
    ALenum format = (channels == 2) ? AL_FORMAT_STEREO16 : AL_FORMAT_MONO16;

    mpg123_format_none(mh);
    mpg123_format(mh, rate, channels, encoding);

    off_t samples = mpg123_length(mh);

    if (samples > 0) {
        duration = static_cast<int>(samples / rate);
    }
    else {
        duration = 0;
    }

    mpg123_seek(mh, 0, SEEK_SET);

    alGenSources(1, &source);

    alGenBuffers(NUM_BUFFERS, buffers);

    unsigned char tempBuffer[AUDIO_BUFFER_SIZE];
    size_t done;
    for (int i = 0; i < NUM_BUFFERS; ++i) {
        if (mpg123_read(mh, tempBuffer, AUDIO_BUFFER_SIZE, &done) == MPG123_OK) {
            alBufferData(buffers[i], format, tempBuffer, done, rate);
        }
    }

    alSourceQueueBuffers(source, NUM_BUFFERS, buffers);
    alSourcePlay(source);
    this->state = AudioState::PLAYING;

    ALint state;
    alGetSourcei(source, AL_SOURCE_STATE, &state);
    if (state != AL_PLAYING) {
        this->state = AudioState::STOPPED;

std::cout << "Source state: " << state << std::endl;
        ClearInstance();
    }
```

```
    streamingThread = std::thread(&Audio::StreamLoop, this);
}

void Audio::ClearInstance()
{
       alDeleteSources(1, &source);

    ALint queued = 0;
    alGetSourcei(source, AL_BUFFERS_QUEUED, &queued);
    while (queued--) {
        ALuint buf;
        alSourceUnqueueBuffers(source, 1, &buf);
    }

    alDeleteBuffers(NUM_BUFFERS, buffers);

    if (streamingThread.joinable()) {
        streamingThread.join();
    }

    if (!mh) {
        return;
    }

    mpg123_close(mh);
    mpg123_delete(mh);
    mpg123_exit();
}

void Audio::StreamLoop()
{
    unsigned char tempBuffer[AUDIO_BUFFER_SIZE];
    size_t done;
    while (state != AudioState::STOPPED && state != AudioState::INITIAL) {
        {
            std::unique_lock<std::mutex> lock(streamMutex);
            if (restartPending) {
                restartPending = false;
                streamCV.notify_one(); // сказать SeekTo, что мы сбросили старый поток
                continue;              // НЕ выходим! Просто продолжаем цикл.
            }
        }

        if (state == AudioState::PAUSED) {
            std::this_thread::sleep_for(std::chrono::milliseconds(100));
            continue;
        }
        ALint processed = 0;
        alGetSourcei(source, AL_BUFFERS_PROCESSED, &processed);

        while (processed-- > 0) {
            ALuint buf;
            alSourceUnqueueBuffers(source, 1, &buf);
            if (mpg123_read(mh, tempBuffer, AUDIO_BUFFER_SIZE, &done) == MPG123_OK) {
                //Audio
                alBufferData(buf, (channels == 2 ? AL_FORMAT_STEREO16 : AL_FORMAT_MONO16),
                    tempBuffer, done, rate);
                alSourceQueueBuffers(source, 1, &buf);
                off_t currentFrame = mpg123_tell(mh);
                position = static_cast<int>(currentFrame / rate);

                //Waves
```

```
                std::lock_guard<std::mutex> lock(waveformMutex);
                int16_t* samples = reinterpret_cast<int16_t*>(tempBuffer);
                size_t count = done / sizeof(int16_t);

                waveformSamples.insert(waveformSamples.end(), samples, samples + count);
                const size_t maxSamples = 2048;
                if (waveformSamples.size() > maxSamples)
                    waveformSamples.erase(waveformSamples.begin(), waveformSamples.end() -
maxSamples);
            }

        }

        ALint state;
        alGetSourcei(source, AL_SOURCE_STATE, &state);
        if (state != AL_PLAYING) {
            alSourcePlay(source);
        }

        if (state == AL_STOPPED) {
            position = duration;
        }

        //std::this_thread::sleep_for(std::chrono::milliseconds(100));
    }
}

Audio::Audio(int id, std::string path, std::string title)
       : id(id), path(path.c_str()), title(translit(title)),
         state(AudioState::INITIAL), duration(0), position(0),
         channels(0), encoding(0), format(0), rate(0),
         source(NULL), mh(NULL)
{
    //Initialize();
}

std::vector<short> Audio::GetWaveformSamples()
{
    return waveformSamples;
}

void Audio::SetTitle(std::string title)
{
    Db* db = Db::GetInstance("music_db.db");

    sqlite3_stmt* stmt;
    sqlite3_prepare_v2(db->GetDb(), "UPDATE playlist_songs SET path = ? WHERE id = ?", -1,
&stmt, nullptr);

    this->title = translit(title);

    sqlite3_bind_text(stmt, 1, this->title.c_str(), -1, SQLITE_TRANSIENT);
    sqlite3_bind_int(stmt, 2, id);

    sqlite3_step(stmt);
    sqlite3_finalize(stmt);
}

void Audio::Play()

{
    if (state == AudioState::STOPPED || state == AudioState::INITIAL) {
        Initialize();
```

```
    }

    if (streamingThread.joinable()) return;

       alSourcePlay(source);
       state = AudioState::PLAYING;

    streamingThread = std::thread([this]() { StreamLoop(); });
}

void Audio::Pause()
{
       if (state == AudioState::PLAYING) {
        state = AudioState::PAUSED;
              return alSourcePause(source);
       }

       if (state == AudioState::PAUSED) {
        state = AudioState::PLAYING;
              return alSourcePlay(source);
       }
}
void Audio::Stop()
{
    state = AudioState::STOPPED;

    duration = position = 0;
       ClearInstance();
}
void Audio::Restart()
{
    if (state == AudioState::PLAYING) {
        SeekTo(0);
        return;
    }

    Stop();
    Play();
}

void Audio::SeekTo(float positionRatio)
{
    if (duration <= 0) {
        return;
    }

    std::unique_lock<std::mutex> lock(streamMutex);

    restartPending = true;
    streamCV.wait(lock, [this]() { return !restartPending; });

    float positionSeconds = positionRatio * static_cast<float>(duration);
    off_t sampleOffset = static_cast<off_t>(positionSeconds * rate);

    // 1. Стоп и очистка источника
    alSourceStop(source);

    ALint queued = 0;
    alGetSourcei(source, AL_BUFFERS_QUEUED, &queued);

while (queued-- > 0) {
        ALuint buf;
        alSourceUnqueueBuffers(source, 1, &buf);
```

```
    }

    // 2. Реальное перемещение
    off_t actual = mpg123_seek(mh, sampleOffset, SEEK_SET);
    if (actual < 0) {
        std::cerr << "Seek failed." << std::endl;
        return;
    }

    position = static_cast<int>(actual / rate);

    // 3. Безопасная перезагрузка буферов (с фильтрацией битых данных)
    for (int i = 0; i < NUM_BUFFERS; ++i) {
        unsigned char buffer[AUDIO_BUFFER_SIZE];
        size_t done = 0;
        int tries = 0;

        while (tries++ < 10) {  // максимум 10 попыток пропустить битые фреймы
            int err = mpg123_read(mh, buffer, AUDIO_BUFFER_SIZE, &done);
            if (err == MPG123_OK || (err == MPG123_DONE && done > 0)) {
                alBufferData(buffers[i], format, buffer, done, rate);
                alSourceQueueBuffers(source, 1, &buffers[i]);
                break;
            }
            else if (err == MPG123_ERR || err == MPG123_BAD_BUFFER) {
                std::cerr << "Warning: Skipped corrupt frame (" << mpg123_strerror(mh) <<
")" << std::endl;
                continue;
            }
            else {
                break;
            }
        }
    }

    // 6. Запускаем снова
    alSourcePlay(source);
    state = AudioState::PLAYING;

    if (!streamingThread.joinable()) {
        streamingThread = std::thread([this]() { StreamLoop(); });
    }
}

int Audio::GetId()
{
    return id;
}

std::string_view Audio::GetPath()
{
    return path;
}

std::string_view Audio::GetTitle()
{
    return title;
}
int Audio::GetPlayedTime()
{
       return position;
}
int Audio::GetDuration()
```

Продовження додатку А

```
{
    return duration;
}
bool Audio::Ended()
{
    return position >= duration;
}
const AudioState& Audio::GetState()
{
       return state;
}
bool Audio::operator==(const Audio& other) const
{
    return path == other.path && title == other.title;
}
```

Додаток Б

Лістинг програмного коду класу «AudioController»

```
#include "AudioController.h"
#include <windows.h>
#include <iostream>
#include <vector>

#include <GL/glew.h>
#include <GL/glut.h>

std::string AudioController::audioRoot = "/";

std::string AudioController::GetExecutableDir()
{
    char buffer[MAX_PATH];
    GetModuleFileNameA(NULL, buffer, MAX_PATH);
    std::string path(buffer);
    return path.substr(0, path.find_last_of("\\/"));
}

void AudioController::Initialize()
{
    device = alcOpenDevice(NULL);
    context = alcCreateContext(device, NULL);
    alcMakeContextCurrent(context);

    audioRoot = GetExecutableDir() + "\\Audio\\";
}

void AudioController::ClearInstance()
{
    if (currentAudio) {
              currentAudio->Stop();
    }
}

AudioController::AudioController()
{
    Initialize();
}

std::weak_ptr<Audio> AudioController::GetCurrentAudio()
{
    return currentAudio;
}

std::string AudioController::GetAudioRoot()
{
    return audioRoot;
}

void AudioController::DrawAudio()
{
    std::vector<short> waveformSamples = currentAudio->GetWaveformSamples();
    std::lock_guard<std::mutex> lock(waveformMutex);
    if (waveformSamples.empty()) return;

    glPushMatrix();
    glTranslatef(300, 720 / 2, 0);
    glColor3f(0.2f, 1.0f, 0.5f);
```

Продовження додатку Б

```
glBegin(GL_LINE_STRIP);

    size_t count = waveformSamples.size();
    float xStep = 960.0f / (float)count;

    for (size_t i = 0; i < count; ++i) {
        float x = i * xStep;
        float y = (waveformSamples[i] / 32768.0f) * 100.0f;
        glVertex2f(x, y);
    }

    glEnd();
    glPopMatrix();
}

void AudioController::DrawTitle(float x, float y)
{
    if (!currentAudio) {
        return;
    }

    glColor3f(0, 0, 0);
    glRasterPos2f(x, y);
    for (char c : currentAudio->GetTitle()) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, c);
    }
}

void AudioController::DrawProgressBar(float x, float y, float w, float h)
{
    float playedTime = currentAudio->GetPlayedTime();
    float duration = currentAudio->GetDuration();
    float percent = duration > 0 ? playedTime / duration : 0;

    // Полоса фона
    glColor3f(0.8f, 0.8f, 0.8f);
    glBegin(GL_QUADS);
    glVertex2f(x, y);
    glVertex2f(x + w, y);
    glVertex2f(x + w, y + h);
    glVertex2f(x, y + h);
    glEnd();

    // Полоса прогресса
    glColor3f(0.3f, 0.6f, 0.3f);
    glBegin(GL_QUADS);
    glVertex2f(x, y);
    glVertex2f(x + w * percent, y);
    glVertex2f(x + w * percent, y + h);
    glVertex2f(x, y + h);
    glEnd();

    int minsPlayed = playedTime / 60;
    int secsPlayed = ((int)playedTime) % 60;
    int minsTotal = duration / 60;
    int secsTotal = ((int)duration) % 60;

    char buf[32];
    sprintf(buf, "%02d:%02d / %02d:%02d", minsPlayed, secsPlayed, minsTotal, secsTotal);
    glColor3f(0, 0, 0);
    glRasterPos2f(x + w + 10, y + h - 3);
    for (int i = 0; buf[i]; i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, buf[i]);
```

```
}

void AudioController::Play(std::shared_ptr<Audio> audio)
{
    if (currentAudio && currentAudio->GetState() != AudioState::STOPPED) {
        ClearInstance();
    }
    currentAudio = std::move(audio);
    currentAudio->Play();
}

void AudioController::Pause()
{
    if (!currentAudio) {
        return;
    }

    currentAudio->Pause();
}

void AudioController::Stop()
{
    ClearInstance();
}

void AudioController::Kill()
{
    ClearInstance();
    alcDestroyContext(context);
    alcCloseDevice(device);
}
```

Додаток В

Лістинг програмного коду класу «db»

```
#include "db.h"
#include <iostream>

Db* Db::instance = nullptr;

Db::Db(std::string path)
	: path(std::move(path))
{
	Initialize();
}

void Db::Initialize()
{
	if (connected) {
		return;
	}

	int rc = sqlite3_open(path.c_str(), &db);
	connected = true;

	if (rc != SQLITE_OK) {
		std::cerr << "Cannot open DB: " << sqlite3_errmsg(db) << "\n";
		connected = false;
		return;
	}
}

Db* Db::GetInstance(std::string path)
{
	if (instance == nullptr) {
		instance = new Db(path);
	}
	return instance;
}

bool Db::Query(std::string query)
{
	Initialize();
	if (!connected) {
		std::cout << "Cannot open DB: " << sqlite3_errmsg(db) << "\n";
		return false;
	}

	const int rc = sqlite3_exec(db, query.c_str(), nullptr, nullptr, nullptr);

	if (rc != SQLITE_OK) {
		std::cerr << "Query failed: " << sqlite3_errmsg(db) << "\n";
		return false;
	}

	return true;
}

void Db::CLose()
{
	sqlite3_close(db);
	connected = false;
}
```

Продовження додатку В

```
sqlite3*& Db::GetDb()
{
	return db;
}

std::string_view Db::GetPath()
{
	return path;
}

bool Db::IsConnected()
{
	return connected;
}
```

Додаток Г

Лістинг програмного коду класу «Dialogs»

```
#include "Dialogs.h"
#include "encoding.h"

#include <Windows.h>

std::vector<std::string> OpenMultipleFilesDialog()
{
    wchar_t* buffer = new wchar_t[8192];
    buffer[0] = L'\0';

    OPENFILENAMEW ofnW = { 0 };
    ofnW.lStructSize = sizeof(ofnW);
    ofnW.hwndOwner = NULL;
    ofnW.lpstrFilter = L"Audio Files\0*.mp3;*.wav\0All Files\0*.*\0";
    ofnW.lpstrFile = buffer;
    ofnW.nMaxFile = 8192;
    ofnW.Flags = OFN_ALLOWMULTISELECT | OFN_EXPLORER;

    std::vector<std::string> result;

    if (GetOpenFileNameW(&ofnW)) {
        wchar_t* ptr = buffer;
        std::wstring folder = ptr;
        ptr += folder.size() + 1;

        if (*ptr == '\0') {
            result.push_back(UTF16ToUTF8(folder));
        }
        else {
            while (*ptr) {
                std::wstring file = ptr;
                std::wstring full = folder + L"\\" + file;
                result.push_back(UTF16ToUTF8(full));
                ptr += file.size() + 1;
            }
        }
    }

    delete[] buffer;
    return result;
}

std::string GetFileNameFromPath(const std::string& path)
{
    size_t pos = path.find_last_of("/\\");
    if (pos != std::string::npos)
        return path.substr(pos + 1);
    return path;
}
```

Додаток Д

Лістинг програмного коду класу «OpenMP3»

```
#define GLEW_STATIC
#include <GL/glew.h>
#include <GLFW/glfw3.h>
#include <GL/glut.h>

#include <imgui.h>
#include <imgui_impl_glfw.h>
#include <imgui_impl_opengl3.h>

#include <windows.h>
#include <commdlg.h>
#include <iostream>
#include <vector>
#include <cmath>
#include <string>

#include "AudioController.h"
#include "PlaylistsController.h"
#include "Dialogs.h"
#include "encoding.h"

#include "Gui/GuiCreateDialog.h"
#include "Gui/GuiPlaylistActivity.h"
#include "Gui/GuiMusicActivity.h"

GLuint windowWidth = 1280, windowHeight = 720;

std::string UTF8ToCP1251(const std::string& utf8) {
    int size = MultiByteToWideChar(CP_UTF8, 0, utf8.c_str(), -1, nullptr, 0);
    std::wstring wstr(size, 0);
    MultiByteToWideChar(CP_UTF8, 0, utf8.c_str(), -1, &wstr[0], size);

    size = WideCharToMultiByte(CP_ACP, 0, wstr.c_str(), -1, nullptr, 0, nullptr, nullptr);
    std::string cp1251(size, 0);
    WideCharToMultiByte(CP_ACP, 0, wstr.c_str(), -1, &cp1251[0], size, nullptr, nullptr);

    return cp1251;
}
```

Продовження додатку Д

```
bool pointInRect(double x, double y, float rx, float ry, float rw, float rh) {
    return x >= rx && x <= rx + rw && y >= ry && y <= ry + rh;
}

void drawButton(float x, float y, float w, float h, const char* label, glm::vec3 color =
glm::vec3(0.2f, 0.4f, 0.7f)) {
    glColor3f(color.x, color.y, color.z);
    glBegin(GL_QUADS);
    glVertex2f(x, y);
    glVertex2f(x + w, y);
    glVertex2f(x + w, y + h);
    glVertex2f(x, y + h);
    glEnd();

    glColor3f(1, 1, 1);
    glRasterPos2f(x + 10, y + h / 2);
    for (int i = 0; label[i]; ++i)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, label[i]);
}

void drawPlayButton(bool playing) {
    float x = 350, y = 250, w = 100, h = 100;
    glColor3f(playing ? 0.0f : 0.8f, 0.8f, 0.2f);
    glBegin(GL_QUADS);
    glVertex2f(x, y);
    glVertex2f(x + w, y);
    glVertex2f(x + w, y + h);
    glVertex2f(x, y + h);
    glEnd();
}

std::string OpenFileDialog() {
    char filename[MAX_PATH] = "";
    OPENFILENAMEA ofn = {};
    ofn.lStructSize = sizeof(ofn);
    ofn.hwndOwner = NULL;
    ofn.lpstrFilter = "MP3 Files\0*.mp3\0All Files\0*.*\0";
    ofn.lpstrFile = filename;
    ofn.nMaxFile = MAX_PATH;
    ofn.Flags = OFN_EXPLORER | OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;

    if (GetOpenFileNameA(&ofn)) {
```

```
        return filename;
    }
    return "";
}

void drawSidebar(PlaylistsController* playlists, int& expandedId, int scrollOffset, int selectedId, std::shared_ptr<Audio> currentAudio) {
    float x = 10, y = 10 - scrollOffset;
    float itemHeight = 30;

    int i = 0;
    for (auto& playlist : *playlists) {
        if (!playlist.second) continue;
        const int& playlistId = playlist.second->GetId();

        glColor3f((selectedId == playlistId) ? 0.4f : 0.7f, 0.7f, 0.7f);
        glBegin(GL_QUADS);
        glVertex2f(x, y);
        glVertex2f(x + 300, y);
        glVertex2f(x + 300, y + itemHeight);
        glVertex2f(x, y + itemHeight);
        glEnd();

        glColor3f(0, 0, 0);
        glRasterPos2f(x + 10, y + 20);
        for (char c : UTF8ToCP1251(std::string(playlist.second->GetTitle())))
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, c);

        y += itemHeight;

        if (expandedId == playlistId) {
            for (auto& audio : *playlist.second) {
                bool audioPlaying = (currentAudio && currentAudio == audio.second);
                glColor3f(audioPlaying ? 0.3f : 0.9f, 0.9f, 0.9f);
                glBegin(GL_QUADS);
                glVertex2f(x + 20, y);
                glVertex2f(x + 280, y);
                glVertex2f(x + 280, y + itemHeight);
                glVertex2f(x + 20, y + itemHeight);
                glEnd();

                glColor3f(0, 0, 0);
```

Продовження додатку Д

```
                glRasterPos2f(x + 30, y + 20);
                for (char c : std::string(audio.second->GetTitle()))
                    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, c);

                y += itemHeight;
            }
        }

        i++;
    }

    drawButton(10, windowHeight - 50, 300, 40, "Add song to selected");
    drawButton(10, windowHeight - 100, 300, 40, "Add new playlist");
}

int main(int argc, char** argv) {
    if (!glfwInit()) return -1;
    GLFWwindow* window = glfwCreateWindow(windowWidth, windowHeight, "OpenGL MP3 Player",
NULL, NULL);
    glfwMakeContextCurrent(window);

    glutInit(&argc, argv);
    glewInit();

    setlocale(LC_ALL, "ru");
    SetConsoleOutputCP(CP_UTF8);
    SetConsoleCP(CP_UTF8);

    ImGui::CreateContext();
    ImGuiIO& io = ImGui::GetIO();

    io.Fonts->Clear();

    ImFontConfig font_cfg;
    font_cfg.OversampleH = 3;
    font_cfg.OversampleV = 1;
    font_cfg.PixelSnapH = true;

    ImFont* myFont = io.Fonts->AddFontFromFileTTF(
        "C:\\Windows\\Fonts\\Arial.ttf",
        18.0f,
```

Продовження додатку Д

```
    &font_cfg,
    io.Fonts->GetGlyphRangesCyrillic()
);
io.FontDefault = myFont;

ImGui_ImplGlfw_InitForOpenGL(window, true);
ImGui_ImplOpenGL3_Init("#version 130");

ImGui_ImplOpenGL3_DestroyFontsTexture();
ImGui_ImplOpenGL3_CreateFontsTexture();

ImGui::StyleColorsDark();

std::unique_ptr<AudioController> audio = std::make_unique<AudioController>();
std::unique_ptr<PlaylistsController> playlists = std::make_unique<PlaylistsController>();

std::unique_ptr<GuiCreateDialog> guiCreateDialog;

std::unique_ptr<GuiPlaylistActivity> guiPlaylistActivity;

std::unique_ptr<GuiMusicActivity> guiAudioActivity;

int scrollOffset = 0;

int expandedPlaylist = -1;
int selectedPlaylist = -1;

int playedPlaylist = -1;

bool repeat = false;
bool skipKeyPress = false;

bool open = true;

while (!glfwWindowShouldClose(window)) {
    glClearColor(1, 1, 1, 1);
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0, windowWidth, windowHeight, 0, -1, 1);
    glMatrixMode(GL_MODELVIEW);
```

Продовження додатку Д

```
        glLoadIdentity();

        std::shared_ptr<Audio> currentAudio = audio->GetCurrentAudio().lock();
        drawSidebar(playlists.get(),   expandedPlaylist,   scrollOffset,   selectedPlaylist,
currentAudio);

        // Чёрный фон под визуализацией
        glColor3f(0, 0, 0);
        glBegin(GL_QUADS);
        glVertex2f(300, 0);
        glVertex2f(1280, 0);
        glVertex2f(1280, 480);
        glVertex2f(300, 480);
        glEnd();

        if (currentAudio) {
            audio->DrawAudio();
            audio->DrawTitle(300, 530);
            audio->DrawProgressBar(300, 550, 880, 10);

            if (currentAudio->Ended()) {
                if (!repeat) {
                    currentAudio->Stop();
                    audio->Play(playlists->operator[](playedPlaylist)->Next(currentAudio-
>GetId()));
                }

                if (repeat) {
                    currentAudio->Restart();
                    repeat = false;
                }
            }

            drawButton(670, 650, 80, 40, "<<");
            drawButton(760,  650,  80,  40,  currentAudio  &&  currentAudio->GetState()  ==
AudioState::PLAYING ? "Pause" : "Play");
            drawButton(850, 650, 80, 40, ">>");

            drawButton(1180, 650, 80, 40, "Repeat", repeat ? glm::vec3(0.8f, 0.8f, 0.2f) :
glm::vec3(0.2f, 0.4f, 0.7f));
        }
```

Продовження додатку Д

```
        if (guiPlaylistActivity) {
                    guiPlaylistActivity->Update();
            if (guiPlaylistActivity->Complated()) {
                        guiPlaylistActivity.reset();
                        skipKeyPress = true;
            }
              }

        if (guiAudioActivity) {
                    guiAudioActivity->Update();
            if (guiAudioActivity->Complated()) {
                        guiAudioActivity.reset();
                        skipKeyPress = true;
                    }
        }

        if (guiCreateDialog) {
            guiCreateDialog->Update();
            if (guiCreateDialog->Complated()) {
                playlists->Add(std::make_unique<Playlist>(std::move(guiCreateDialog-
>GetResponse())));
                        guiCreateDialog.reset();
                skipKeyPress = true;
                    }
        }

        glfwSwapBuffers(window);
        glfwPollEvents();

        if    (glfwGetMouseButton(window,    GLFW_MOUSE_BUTTON_LEFT)    ==    GLFW_PRESS    &&
!skipKeyPress) {
            double mx, my;
            glfwGetCursorPos(window, &mx, &my);

            if (currentAudio && pointInRect(mx, my, 760, 650, 80, 40)) { //Pause
                audio->Pause();
            }
            else if (currentAudio && pointInRect(mx, my, 670, 650, 80, 40)) { //Prev
                audio->Play(playlists->operator[](playedPlaylist)->Prev(currentAudio-
>GetId()));
            }
```

Продовження додатку Д

```
            else if (currentAudio && pointInRect(mx, my, 850, 650, 80, 40)) { //Next
                audio->Play(playlists->operator[](playedPlaylist)->Next(currentAudio-
>GetId()));
            }
            else if (currentAudio && pointInRect(mx, my, 1180, 650, 80, 40)) { //Repeat
                repeat = !repeat;
            }
            else if (pointInRect(mx, my, 10, windowHeight - 100, 300, 40)) { // Add new
playlist
                if (guiCreateDialog) {
                                guiCreateDialog.reset();
                }

                            guiCreateDialog    =    std::make_unique<GuiCreateDialog>("Create
playlist");
                guiCreateDialog->Show();
            }
            else if (pointInRect(mx, my, 10, windowHeight - 50, 300, 40)) { //Add new track
                std::vector<std::string> paths = OpenMultipleFilesDialog();
                for (const auto& path : paths) {
                    std::string name = GetFileNameFromPath(path);
                    playlists->operator[](selectedPlaylist)->Add(path, std::move(name));
                }
            }
            else if (pointInRect(mx, my, 300, 550, 880, 10)) {
                float position = (mx - 300.0f) / 880.0f;
                if (std::shared_ptr<Audio> currentAudio = audio->GetCurrentAudio().lock()) {
                    currentAudio->SeekTo(position);
                }
            }
            else { //Select
                float y = 10 - scrollOffset;
                int i = 0;
                bool selectMusic = false;

                for (auto& playlist : *playlists) {
                    const int& playlistId = playlist.second->GetId();
                    if (pointInRect(mx, my, 10, y, 300, 30)) { //Select playlist
                        if (expandedPlaylist == playlistId) {
                            expandedPlaylist = -1;
                            selectedPlaylist = -1;
```

Продовження додатку Д

```
                    }
                    else {
                        expandedPlaylist = playlistId;
                        selectedPlaylist = playlistId;
                    }
                    break;
                }
                y += 30;

                if (expandedPlaylist == playlistId) {  //Select song
                    for (auto& audioEntry : *playlist.second) {
                        if (pointInRect(mx, my, 30, y, 250, 30)) {
                            audio->Play(audioEntry.second);
                            selectMusic = true;
                            playedPlaylist = playlistId;
                            break;
                        }
                        y += 30;
                    }
                }

                if (selectMusic) {
                    break;
                }

                i++;
            }
        }

        while (glfwGetMouseButton(window, GLFW_MOUSE_BUTTON_LEFT) == GLFW_PRESS)
            glfwPollEvents();
    }

    if (glfwGetMouseButton(window, GLFW_MOUSE_BUTTON_RIGHT) == GLFW_PRESS &&
!skipKeyPress) {
        double mx, my;
        glfwGetCursorPos(window, &mx, &my);

        float y = 10 - scrollOffset;
        int i = 0;
        bool selectMusic = false;
```

Продовження додатку Д

```
            for (auto& playlist : *playlists) {
                const int& playlistId = playlist.second->GetId();
                if (pointInRect(mx, my, 10, y, 300, 30)) { //Playlist menu
                    if (guiPlaylistActivity) {
                        guiPlaylistActivity.reset();
                    }

                    guiPlaylistActivity
                        =      std::make_unique<GuiPlaylistActivity>(glm::vec2(300,      my),
playlist.second.get(), playlists.get());

                    guiPlaylistActivity->Show();
                    break;
                }
                y += 30;

                if (expandedPlaylist == playlistId) {  //Song menu
                    for (auto& audioEntry : *playlist.second) {
                        if (pointInRect(mx, my, 30, y, 250, 30)) {
                            if (guiAudioActivity) {
                                                guiAudioActivity.reset();
                            }

                                                guiAudioActivity
                                                =
std::make_unique<GuiMusicActivity>(glm::vec2(300,  my),  playlists->operator[](playedPlaylist
== -1 ? selectedPlaylist : playedPlaylist), audioEntry.second);
                            guiAudioActivity->Show();

                            selectMusic = true;
                            break;
                        }
                        y += 30;
                    }
                }

                if (selectMusic) {
                    break;
                }

                i++;
            }
```

Продовження додатку Д

```
        }

        if (glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_PRESS) {
            scrollOffset += 10;
        }
        else if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_PRESS) {
            scrollOffset = std::max(0, scrollOffset - 10);
        }

        skipKeyPress = false;
    }

    audio->Kill();
    glfwTerminate();
    return 0;
}
```

Додаток Е

Лістинг програмного коду класу «Playlist»

```
#include "Playlist.h"
#include "db.h"
#include <iostream>

#include "translit.h"

Playlist::Playlist(std::string title)
	: id(-1), queueEnterIndex(-1)
{
	Db* db = Db::GetInstance("music_db.db");

	sqlite3_stmt* stmt;
	sqlite3_prepare_v2(db->GetDb(), "INSERT INTO playlists (title) VALUES (?);", -1,
&stmt, nullptr);

	sqlite3_bind_text(stmt, 1, title.c_str(), -1, SQLITE_TRANSIENT);

	if (sqlite3_step(stmt) == SQLITE_DONE) {
		int64_t newId = sqlite3_last_insert_rowid(db->GetDb());

		this->id = newId;
		this->title = std::move(title);
	}

	sqlite3_finalize(stmt);
}

Playlist::Playlist(int id, std::string title)
	: id(id), title(title), queueEnterIndex(-1)
{
}

Playlist::Playlist(
	int id,
	std::string title,
	std::unordered_map<
		int,
		std::shared_ptr<Audio>
	> audioList
) : id(id), title(title), queueEnterIndex(-1)
{
	this->audioList.swap(audioList);
}

void Playlist::Add(std::string path, std::string title)
{
	Db* db = Db::GetInstance("music_db.db");

	sqlite3_stmt* stmt;
	sqlite3_prepare_v2(db->GetDb(), "INSERT INTO playlist_songs (playlist_id, title,
path) VALUES (?, ?, ?);", -1, &stmt, nullptr);

	sqlite3_bind_int(stmt, 1, id);
	sqlite3_bind_text(stmt, 2, path.c_str(), -1, SQLITE_TRANSIENT);
	sqlite3_bind_text(stmt, 3, title.c_str(), -1, SQLITE_TRANSIENT);

	if (sqlite3_step(stmt) == SQLITE_DONE) {
		int64_t newId = sqlite3_last_insert_rowid(db->GetDb());
```

Продовження додатку Е

```
			audioList[newId] = std::make_shared<Audio>(newId, std::move(path),
std::move(title));
		}

		sqlite3_finalize(stmt);
}

void Playlist::Add(std::shared_ptr<Audio> audio)
{
		if (Include(audio->GetId())) {
			return;
		}

		Db* db = Db::GetInstance("music_db.db");

		sqlite3_stmt* stmt;
		sqlite3_prepare_v2(db->GetDb(), "INSERT INTO playlist_songs (playlist_id, title,
path) VALUES (?, ?, ?);", -1, &stmt, nullptr);

		sqlite3_bind_int(stmt, 1, id);
		sqlite3_bind_text(stmt, 2, audio->GetPath().data(), -1, SQLITE_TRANSIENT);
		sqlite3_bind_text(stmt, 3, audio->GetTitle().data(), -1, SQLITE_TRANSIENT);

		if (sqlite3_step(stmt) == SQLITE_DONE) {
			int64_t newId = sqlite3_last_insert_rowid(db->GetDb());
			audioList[newId] = std::move(audio);
		}

		sqlite3_finalize(stmt);
}

void Playlist::Add(int id, std::shared_ptr<Audio> audio)
{
		audioList[id] = std::move(audio);
}

void Playlist::AddToQueue(std::shared_ptr<Audio> audio)
{
		queue.insert(queue.begin(), audio);
}

std::shared_ptr<Audio> Playlist::Next(int currentId)
{
		if (!queue.empty()) {
			if (queueEnterIndex == -1) {
				queueEnterIndex = currentId;
			}

			std::shared_ptr<Audio> audio = queue.front();
			queue.erase(queue.begin());
			return audio;
		}

		if (queueEnterIndex != -1) {
			currentId = queueEnterIndex;
			queueEnterIndex = -1;
		}

		auto it = audioList.find(currentId);

		if (it != audioList.end()) {
			++it;
```

Продовження додатку Е

```
			if (it != audioList.end()) {
				return it->second;
			}
		}

		it = audioList.begin();
		return it->second;
}

std::shared_ptr<Audio> Playlist::Prev(int currentId)
{
		auto it = audioList.find(currentId);

		if (it != audioList.begin()) {
			--it;

			if (it != audioList.end()) {
				return it->second;
			}

			it = --audioList.end();
		}

		return it->second;
}

void Playlist::SetTitle(std::string title)
{
		this->title = std::move(title);

		Db* db = Db::GetInstance("music_db.db");

		sqlite3_stmt* stmt;
		sqlite3_prepare_v2(db->GetDb(), "UPDATE playlists SET title = ? WHERE id = ?", -1,
&stmt, nullptr);


		sqlite3_bind_text(stmt, 1, translit(this->title).c_str(), -1, SQLITE_TRANSIENT);
		sqlite3_bind_int(stmt, 2, id);

		sqlite3_step(stmt);
		sqlite3_finalize(stmt);
}

void Playlist::Remove(int id)
{
		audioList.erase(id);
}

void Playlist::Remove(std::string_view title)
{
		for (auto& audio : audioList) {
			if (audio.second->GetTitle() == title) {
				audioList.erase(audio.first);
				return;
			}
		}
}

void Playlist::Remove(Audio* audio)
{
		for (auto& item : audioList) {
```

Продовження додатку Е

```
			if (*item.second == *audio) {
				audioList.erase(item.first);
				return;
			}
		}
}

void Playlist::Delete()
{
		Db* db = Db::GetInstance("music_db.db");

		sqlite3_stmt* stmt;
		sqlite3_prepare_v2(db->GetDb(), "DELETE FROM playlist_songs WHERE playlist_id = ?", -
1, &stmt, nullptr);

		for (auto& audio : audioList) {
			sqlite3_bind_int(stmt, 1, audio.first);
			if (sqlite3_step(stmt) != SQLITE_DONE) {
				std::cout << "Error while deleting playlist song " << audio.second-
>GetId() << std::endl
					<< "Error: " << sqlite3_errmsg(db->GetDb()) << std::endl;
			}

			sqlite3_reset(stmt);
		}

		sqlite3_prepare_v2(db->GetDb(), "DELETE FROM playlists WHERE id = ?", -1, &stmt,
nullptr);
		sqlite3_bind_int(stmt, 1, id);
		if (sqlite3_step(stmt) != SQLITE_DONE) {
			std::cout << "Error while deleting playlist " << id << std::endl
				<< "Error: " << sqlite3_errmsg(db->GetDb()) << std::endl;
		}

		sqlite3_finalize(stmt);
}

void Playlist::Delete(int id)
{
		Db* db = Db::GetInstance("music_db.db");

		sqlite3_stmt* stmt;
		sqlite3_prepare_v2(db->GetDb(), "DELETE FROM playlist_songs WHERE id = ?", -1, &stmt,
nullptr);

		sqlite3_bind_int(stmt, 1, id);
		if (sqlite3_step(stmt) != SQLITE_DONE) {
			std::cout << "Error while deleting playlist song " << id << std::endl
				<< "Error: " << sqlite3_errmsg(db->GetDb()) << std::endl;
		}

		sqlite3_reset(stmt);

		sqlite3_finalize(stmt);

		Remove(id);
}

const int Playlist::GetAudioId(std::string_view title)
{
		for (auto &item : audioList) {
			if (item.second->GetTitle() == title) {
```

Продовження додатку Е

```
                        return item.first;
                }
        }
        return -1;
}

const int Playlist::GetAudioId(Audio* audio)
{
        for (auto& item : audioList) {
                if (*item.second == *audio) {
                        return item.first;
                }
        }
        return -1;
}

std::string_view Playlist::GetTitle()
{
        return title;
}

const int& Playlist::GetId()
{
        return id;
}

const int Playlist::Size()
{
        return audioList.size();
}

const bool Playlist::Include(int id)
{
        return audioList.empty() ? false : audioList.find(id) != audioList.end();
}

std::unordered_map<int, std::shared_ptr<Audio>>::iterator Playlist::begin()
{
        return audioList.begin();
}

std::unordered_map<int, std::shared_ptr<Audio>>::iterator Playlist::end()
{
        return audioList.end();
}

std::shared_ptr<Audio> Playlist::operator[](int id)
{
        auto it = audioList.find(id);
        if (it == audioList.end()) {
                return nullptr;
        }
        return it->second;
}

std::shared_ptr<Audio>Playlist::operator[](std::string_view title)
{
        int id = GetAudioId(title);
        if (id == -1) {
                return nullptr;
        }
        return (*this)[id];}
```

Додаток Ж

Лістинг програмного коду класу «PlaylistsController»

```
#include "PlaylistsController.h"
#include "db.h"

#include <iostream>

void PlaylistsController::GetPlaylistsFromDb()
{
	setlocale(LC_ALL, "ru");

	Db* db = Db::GetInstance("music_db.db");

	sqlite3_stmt* stmt;

	sqlite3_prepare_v2(db->GetDb(), "SELECT * FROM playlists", -1, &stmt, nullptr);

	while (sqlite3_step(stmt) == SQLITE_ROW) {
		int id = sqlite3_column_int(stmt, 0);
		std::string title = reinterpret_cast<const char*>(
			sqlite3_column_text(stmt, 1)
			);
		this->playlists[id] = std::make_unique<Playlist>(
			id,
			std::move(title)
		);
	}

	std::vector<std::thread> threads;

	for (auto& playlist : this->playlists) {
		sqlite3_prepare_v2(db->GetDb(), "SELECT id, title, path FROM playlist_songs
WHERE playlist_id = ?", -1, &stmt, nullptr);
		sqlite3_bind_int(stmt, 1, playlist.second->GetId());

		while (sqlite3_step(stmt) == SQLITE_ROW) {
			int id = sqlite3_column_int(stmt, 0);
			std::string path = reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 1));
			std::string title = reinterpret_cast<const
char*>(sqlite3_column_text(stmt, 2));
			playlist.second->Add(id, std::make_unique<Audio>(id, path, title));
		}
	}
}

void PlaylistsController::Initialize()
{
	Db* db = Db::GetInstance("music_db.db");

	if (!db->Query(R"(
	 CREATE TABLE IF NOT EXISTS playlists (
	     id INTEGER PRIMARY KEY AUTOINCREMENT,
	     title TEXT NOT NULL
	 );
    )")) {
		return;
	}

	if (!db->Query(R"(
```

Продовження додатку Ж

```
		CREATE TABLE IF NOT EXISTS playlist_songs (
			id INTEGER PRIMARY KEY AUTOINCREMENT,
			playlist_id INTEGER NOT NULL,
			title TEXT NOT NULL,
			path TEXT NOT NULL
		);)")
	) {
		return;
	}

	GetPlaylistsFromDb();

	if (playlists.empty()) {
		sqlite3_stmt* stmt;
		sqlite3_prepare_v2(db->GetDb(), "INSERT INTO playlists (title) VALUE(?);", -1,
&stmt, nullptr);

		sqlite3_bind_text(stmt, 1, "Default playlist", -1, SQLITE_TRANSIENT);

		if (sqlite3_step(stmt) == SQLITE_DONE) {
			int64_t newId = sqlite3_last_insert_rowid(db->GetDb());
			this->playlists[newId] = std::make_unique<Playlist>(newId, "Default
playlist");
		}
		sqlite3_finalize(stmt);
	}
}

Playlist* PlaylistsController::GetPlaylistByTitle(std::string_view title)
{
	for (auto& playlist : this->playlists) {
		if (playlist.second->GetTitle() == title) {
			return playlist.second.get();
		}
	}
	return nullptr;
}

PlaylistsController::PlaylistsController()
{
	Initialize();
}

void PlaylistsController::Add(
	std::unique_ptr<Playlist> playlist
)
{
	playlists[playlist->GetId()] = std::move(playlist);
}

void PlaylistsController::Remove(int id)
{
	this->playlists.erase(id);
}

void PlaylistsController::Remove(std::string_view title)
{
	Playlist* playlist = GetPlaylistByTitle(title);
	if (playlist != nullptr) {
		Remove(playlist->GetId());
	}
}
```

Продовження додатку Ж

```
const int& PlaylistsController::Size()
{
	return this->playlists.size();
}

std::unordered_map<int, std::unique_ptr<Playlist>>::iterator PlaylistsController::begin()
{
	return this->playlists.begin();
}

std::unordered_map<int, std::unique_ptr<Playlist>>::iterator PlaylistsController::end()
{
	return this->playlists.end();
}

Playlist* PlaylistsController::operator[](int id)
{
	return this->playlists[id].get();
}

Playlist* PlaylistsController::operator[](std::string_view title)
{
	return GetPlaylistByTitle(title);
}
```