Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка десктоп-додатку цифрового музичного сервісу з використанням технології Windows Presentation Foundation та гнучких архітектурних моделей

Виконав(ла): студент(ка) IV курсу, групи СП-42
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

		Франчевський М. І.
	(підпис)	(прізвище та ініціали)
Керівник		Петрик М. Р.
	(підпис)	(прізвище та ініціали)
Нормоконтроль		Стоянов Ю. М.
	(підпис)	(прізвище та ініціали)
Завідувач кафедри		Петрик М. Р.
	(підпис)	(прізвище та ініціали)
Рецензент		
	(підпис)	(прізвище та ініціали)

Тернопіль
2023

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет _____
(повна назва факультету)

Кафедра _____
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня _____
(назва освітнього ступеня)

за спеціальністю _____
(шифр і назва спеціальності)

студенту _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____

_____

_____

Керівник роботи _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи _____

_____

_____

4. Зміст роботи (перелік питань, які потрібно розробити)

_____

_____

_____

_____

_____

_____

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)

_____

_____

_____

_____

_____

_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на здобуття освітнього ступеню «бакалавр» за спеціальністю 121 – Інженерія програмного забезпечення. Тернопільський національний технічний університет ім. Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42, 2023 рік.

Пояснювальна записка до кваліфікаційної роботи на здобуття освітнього ступеню «бакалавр» містить: 68 с., 27 рис., 2 додат., 19 бібліогр.

Тема: Розробка десктоп-додатку цифрового музичного сервісу з використанням технології Windows Presentation Foundation та гнучких архітектурних моделей.

Дану кваліфікаційну роботу бакалавра присвячено розробці додатку цифрового музичного сервісу. Використовуючи подійно-орієнтовану архітектуру було створено додаток з застосуванням технології Windows Presentation Foundation у середовищі Visual Studio.

Ключові слова: десктоп-додаток, цифровий музичний сервіс, Microsoft Sql Server, Visual Studio, Windows Presentation Foundation, C#, подійно-орієнтована архітектура.

ANNOTATION

Qualification work for the bachelor's degree in specialty 121 - Software Engineering. Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Software Engineering Department, SP-42, 2023.

Explanatory note to the qualification work for the bachelor's degree contains: 68 p., 27 fig., 2 appendix, 19 ref.

Topic: Development of a desktop application for a digital music service using Windows Presentation Foundation technology and flexible architectural models.

This bachelor's thesis is devoted to the development of a digital music service application. Using event-driven architecture, an application was created with Windows Presentation Foundation technology in the Visual Studio environment.

Keywords: desktop application, digital music service, Microsoft Sql Server, Visual Studio, Windows Presentation Foundation, C#, event-driven architecture.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Стримінг - потокове передавання інформації (медіа).

Стримінговий сервіс – сервіс прослуховування медіа.

Плейлист – список відтворення.

Контент – інформаційне наповнення додатку.

WPF – Windows Presentation Foundation

ПЗ – програмне забезпечення

IBM RSA – Rational Software Architect

ВСТУП

Станом на сьогодні, декілька музичних стримінгових платформ, таких як Spotify, YouTube Music, Apple Music, Amazon Music, є загальновідомими та популярними.

Однак індустрія стримінгу музики постійно розвивається, тому створення нової музичної стримінгової платформи залишається актуальним і зараз. Ця задача вимагає глибокого розуміння музичної індустрії, ліцензійних угод, технологічної інфраструктури, взаємодії користувачів з продуктом і маркетингових стратегій. Дуже важливо враховувати різні фактори, такі як цільова аудиторія, унікальні функції чи ціннісні пропозиції, моделі монетизації та те, як диференціюватись від існуючих платформ.

Однією з потенційних сфер уваги для нової платформи стримінгу музики буде обслуговування конкретних ніш, які недостатньо охопленими нинішніми лідерами ринку, а саме, музику українських виконавців, що може залучити спеціальну базу зацікавлених користувачів.

Окрім того, впровадження інноваційних функцій або підходів, таких як розширені алгоритми рекомендацій, соціальна інтеграція, ексклюзивний контент виконавців, прямі трансляції або віртуальні концерти, допоможе виділити нову платформу серед існуючих конкурентів та залучити більше користувачів.

Варто зазначити, що в індустрії стримінгу музики висока конкуренція, і укладення ліцензійних угод із великими студіями звукозапису та незалежними виконавцями може бути серйозною проблемою. Для успіху нової платформи вкрай важливо створити значний каталог якісної музики, який можна запропонувати користувачам.

Завдяки тому, що індустрія стримінгу музики продовжує безперервно розвиватися, створення нової стримінгової платформи залишається актуальним завданням, залишаючи можливості для інновацій та краще задовольняючи потреби більшої кількості користувачів.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ЦИФРОВОГО МУЗИЧНОГО СЕРВІСУ

1.1 Огляд конкурентів

Лідерами серед музичних стримінгових сервісів можна виділити такі платформи:

- Spotify;
- Apple Music;
- YouTube Music;
- Amazon Music;
- Deezer.

Найбільшим сервісом, беззаперечно, є шведська компанія Spotify (рисунок 1.1), заснована у 2006 році. Даний сервіс здобув широку популярність завдяки тому, що користувач може не купуючи преміум підписку безкоштовно слухати будь-яку пісню, яка йому подобається, періодично отримуючи рекламні оголошення. Окрім того, даний сервіс популяризувало розміщення на ньому не лише музики, а і подкастів, інформації про виконавців, їхні поточні тури та виступи, а також розміщення товарів виконавців.

Рисунок 1.1 – Логотип сервісу Spotify

Згідно з фінансовим звітом, наданим компанією у четвертому кварталі 2022 року, Spotify користується попитом 489 мільйонів активних користувачів по усьому світу, з яких понад 205 мільйонів, тобто близько 41,9% користуються платною підпискою Spotify Premium [1]. Завдяки такій широкій популярності сервісу, прибутки компанії за 2022 рік становлять 11.7 мільярди євро.

Другим за величиною музичним стримінговим сервісом є Apple Music (рисунок 1.2), який станом на 2021 рік використовували 60 мільйонів користувачів зі всього світу.

Рисунок 1.2 – Логотип сервісу Apple Music

Apple Music отримує дохід за рахунок платної підписки за доступ до музики. Фінансові звіти Apple не розбивають конкретні цифри прибутку лише для Apple Music, оскільки це частина більшого сегменту послуг, який включає інші продукти та послуги [2]. У четвертому кварталі 2022 року сегмент послуг отримав понад 19,2 мільярдів доларів доходу, що стало значним зростанням порівняно з попереднім роком.

Сервіс своєю популярністю завдячує:

- широкій користувацькій базі, завдяки розповсюдженості iPhone та інших продуктів Apple;
- інтегрованості в екосистему продукції Apple;
- брендингу і маркетингу компанії;
- ексклюзивному контенту.

Іншою великою стримінговою платформою є YouTube Music (рисунок 1.3). Це сервіс від YouTube, який зосереджується на наданні широкого спектру музичного вмісту своїм користувачам. Він пропонує поєднання пісень, музичних відео, живих виступів, каверів і реміксів від різних виконавців. Однак конкретні фінансові відомості про YouTube Music як окрему службу не розголошуються. Натомість його фінансові показники, ймовірно, відображаються як частина загальних фінансових результатів YouTube [3].

Рисунок 1.3 – Логотип сервісу YouTube Music

YouTube, що належить Google (дочірній компанії Alphabet Inc.), отримує дохід з різних джерел, включаючи рекламу, підписки та ліцензування вмісту. Реклама, що відображається на YouTube Music, сприяє прибутку платформи. Крім того, YouTube пропонує платну підписку YouTube Premium, яка включає доступ до YouTube Music без реклами та інші переваги, як-от відтворення в режимі офлайн і доступ до YouTube Originals.

Важливо зазначити, що на фінансові показники YouTube Music впливають різні фактори, як-от залучення користувачів, розмір бази преміум підписників, конкуренція з боку інших стримінгових платформ і зміни в ландшафті музичної індустрії.

Ще одним значним конкурентом є Amazon Music (рисунок 1.4). Це сервіс стримінгу музики від Amazon, однієї з найбільших у світі компаній електронної комерції та технологій.

Рисунок 1.4 – Логотип сервісу Amazon Music

Amazon Music приносить прибуток в основному за рахунок послуг підписки та продажу музики. Компанія не розкриває конкретні цифри доходу окремо для свого музичного підрозділу. Однак Amazon повідомляє про загальний чистий обсяг продажів, який включає дохід від різних сегментів, включаючи Amazon Music.

Окрім вищенаведених, досить популярною стримінговою платформою є Deezer (рисунок 1.5).

Риснок 1.5 – Логотип сервісу Deezer

Сервіс отримує дохід переважно за рахунок підписок і показу реклами. Користувачі можуть вибирати між безкоштовним користуванням з рекламними паузами, та платною преміум підпискою, яка пропонує додаткові функції, та прослуховування музики без реклами. Deezer має значну базу користувачів, станом на 2022 рік у сервісу було понад 9.4 мільйонів користувачів. Загальний дохід компанії, станом на 2022 рік, становив 451.2 мільйони євро [4].

1.2 Обґрунтування вибору напрямку дослідження

Для реалізації даного проєкту, було обрано середовище Visual Studio (рисунок 1.6) – програмне забезпечення, розроблене компанією Microsoft. Visual Studio є одним з найпопулярніших та потужних інтегрованих середовищ розробки (IDE) для розробки програмного забезпечення. Це програмне рішення надає багато можливостей та інструментів для розробки різноманітних додатків, включаючи веб-сайти, мобільні додатки, настільні програми та ігри.

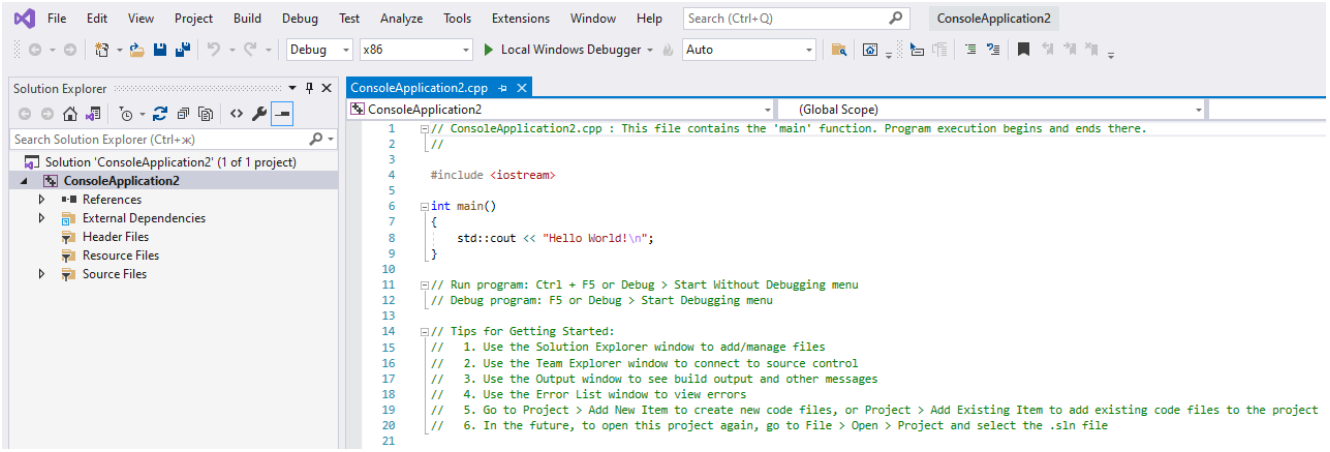

Рисунок 1.6 – Загальний вигляд середовища розробки Visual Studio

Ключові особливості та переваги Visual Studio:

- мовна підтримка;
- розширення;
- відладка;
- інтеграція з іншими інструментами;
- кросплатформенна розробка;
- зручний інтерфейс.

Visual Studio підтримує багато мов програмування, включаючи C, C#, C++, Visual Basic, C++/CLI, JavaScript, TypeScript, F#, тощо.

Це середовище розробки має широкий набір розширень, які дозволяють значно розширити його функціональність для покращення підтримки конкретних технологій, інструментів аналізу коду, систем контролю версій та багато іншого.

Окрім цього, Visual Studio надає потужні засоби відлагодження, що допомагають знаходити та виправляти помилки в коді. Відладка може бути виконана крок за кроком, дозволяючи контролювати виконання програми та переглядати значення змінних.

Також, Visual Studio інтегрується з багатьма іншими інструментами розробки, такими як системи контролю версій (наприклад, Git), тестування, збирання та публікації програмного забезпечення. Це спрощує робочий процес та дозволяє легко використовувати різні інструменти в єдиному інтерфейсі.

Дане середовище дозволяє розробляти кросплатформенні додатки, що працюють на різних операційних системах, таких як Windows, macOS і Linux, що дає можливість створювати мобільні додатки для iOS та Android, веб-додатки, які працюють на різних веб-серверах і браузерах, а також застосунки для різних платформ.

Visual Studio має дружній користувацький інтерфейс, який забезпечує зручну роботу з проєктами та файлами. Інструменти та функціональність цього IDE розташовані в зручних бокових меню та панелях, що полегшує навігацію та використання.

Вибір Visual Studio, як середовища розробки залежав від конкретних потреб та вимог проєкту. Враховуючи його потужність, розширюваність та широку підтримку, Visual Studio було вибрано для розробки застосунку цієї кваліфікаційної роботи.

Після цього, було вибрано мову програмування, а саме C# [5], розроблений компанією Microsoft, адже це універсальна мова програмування, яка використовується для розробки широкого спектру додатків для настільних комп’ютерів, веб-додатків, ігор, тощо.

Основні фактори, які вплинули на вибір мови програмування C#:

- Синтаксис і структура.
- Платформа .NET
- Мультиплатформеність.
- Розширена підтримка Visual Studio.

C# пропонує чисту і лаконічну синтаксичну модель, яка полегшує розробку програм, а платформа .NET забезпечує багатофункціональну стандартну бібліотеку, яка пропонує різноманітні класи і функції для роботи з мережами, базами даних, графікою та багатьма іншими аспектами розробки програмного забезпечення. Це дозволяє створювати додатки, які можуть працювати на різних платформах без значних змін у коді.

Окрім цього, мова програмування C# чудово поєднується з Visual Studio, який пропонує розширення для C# з можливістю автоматичного завершення коду, підказок, аналізу помилок і багато іншого.

В цілому, C# є популярною мовою програмування, яка пропонує потужні можливості розробки, широку підтримку і дружній синтаксис, що робить її найбільш привабливим вибором для розробки застосунку даного проєкту.

Оскільки даний проєкт буде реалізовано у вигляді десктопного додатку у середовищі Windows, було вирішено використовувати технологію Windows Presentation Foundation (WPF) [6]. Вона є потужним інструментом для розробки десктопних додатків у середовищі Windows. Основними факторами вибору технології WPF є:

- можливості створення дизайну та інтерфейсу користувача;
- розширені можливості мультимедіа;
- повна інтеграція з середовищем Windows;
- можливості розширення;
- модульність та розділення обов'язків;
- сумісність зі старими версіями Windows.

WPF надає широкий спектр можливостей для створення ефективного та привабливого інтерфейсу користувача. Є можливість використання векторної графіки, стилів, шаблонів, анімацій та інших елементів, для створення сучасного та привабливого дизайну програми.

Дана технологія має вбудовану підтримку мультимедіа, що дозволяє відтворювати аудіо- та відеофайли, використовувати 2D та 3D графіку, а також працювати з мультимедійними пристроями.

Оскільки WPF є частиною платформи .NET Framework, це надає йому значну інтеграцію зі середовищем Windows. Наявні можливості використання класів та бібліотек .NET Framework для роботи зі всіма можливостями операційної системи Windows.

WPF підтримує розширення, що дозволяє створювати власні елементи управління, власні властивості та розширювати можливості фреймворку за допомогою наслідування та розширення класів.

Також, ця технологія дозволяє розділяти логіку додатку (код) та представлення (розмітку та стилі). Це може дозволити розробникам та дизайнерам працювати над проектом паралельно та незалежно один від одного.

Окрім цього, WPF підтримує сумісність зі старими версіями Windows, починаючи з Windows XP, що означає, що розроблені за допомогою цієї технології додатки можуть працювати на різних версіях операційної системи.

Загалом, Windows Presentation Foundation є потужним інструментом для розробки десктопних додатків у середовищі Windows, з великим набором можливостей для створення привабливого інтерфейсу користувача, роботи з мультимедіа та інтеграції з операційною системою, і тому, її було вибрано для реалізації даного проєкту.

1.3 Технічний аспект проблеми

Предметна область стримінгових сервісів містить декілька технічних аспектів, які впливають на їх функціонування та якість надання послуг. Основні технічні виклики пов'язані зі забезпеченням стабільності, швидкості та якості передачі великих обсягів даних в режимі реального часу до великої кількості кінцевих користувачів.

Основні аспекти технічної проблематики стримінгових сервісів:

- пропускна здатність мережі;
- кодеки та компресія;
- масштабованість;
- управління буферизацією;
- мобільність та сумісність;

Щоб забезпечити плавний стрімінг високоякісного відео, необхідна достатня швидкість і пропускна здатність інтернет-з'єднання між серверами стримінгового сервісу та кінцевими користувачами. Недостатня пропускна здатність може призводити до затримок, або низької якості аудіо.

Для ефективної передачі великих обсягів даних стрімінгові сервіси використовують кодеки, що стискають аудіо дані. Вибір кодеків та рівень компресії впливає на якість відео та ресурсоємність для відтворення на пристроях користувачів. Музичні файли мають бути ефективно стиснуті для передачі через мережу з мінімальною втратою якості звуку.

Стримінгові сервіси повинні бути здатні обробляти велику кількість запитів від користувачів одночасно. Масштабованість включає в себе налагодження серверної інфраструктури, використання хмарних рішень та оптимізацію обробки запитів.

Для забезпечення плавного відтворення музики, сервіси музичного стримінгу використовують буферизацію. Це означає, що невеликий фрагмент музики завантажується в кеш пристрою користувача перед відтворенням. Ефективне управління буферизацією може бути складним завданням, оскільки потрібно забезпечити плавні переходи між треками, уникнути затримок чи перерв у відтворенні.

Стримінгові сервіси музики мають бути доступні на різних платформах та пристроях, таких як комп'ютери, смартфони, планшети тощо. Оптимізація та розробка додатків для різних операційних систем, а також забезпечення сумісності з різними пристроями, може бути викликом для розробників стримінгових сервісів.

На сьогоднішній день, одними з найпоширеніших методологій розробки ПЗ є:

- Гнучка методологія;
- DevOps;
- Ощадлива методологія;
- Спіральна методологія;
- RUP (Rational Unified Process);

– Каскадна методологія.

Гнучка методологія (Agile) підходить для проектів, де вимоги можуть змінюватися протягом часу. Agile дозволяє гнучко адаптуватися до змін і швидко реагувати на вимоги користувачів.

DevOps поєднує розробку програмного забезпечення (Dev) з операційною підтримкою (Ops). Цей підхід сприяє автоматизації процесу розробки, тестування, випуску та впровадження програмного забезпечення. DevOps може допомогти забезпечити швидке впровадження та надійність стримінгового сервісу.

Ощадлива методологія (Lean) базується на принципах ефективного використання ресурсів та усунення витрат. Ця методологія може допомогти оптимізувати процес розробки та забезпечити якісний стримінговий сервіс.

Спіральна методологія (Spiral) зосереджена на ітераційному розвитку та оцінці ризиків. Вона передбачає постійне оцінювання ризиків і прийняття рішень залежно від цих оцінок. Spiral є корисною, якщо наявні складні технічні аспекти або великі ризики, які потребують постійного контролю.

RUP базується на уніфікованій мові моделювання (UML) і наголошує на використанні моделей для керування процесом розробки. Вона дотримується поетапного підходу, розділяючи життєвий цикл розробки на серію чітко визначених ітерацій. Кожна ітерація складається з декількох етапів, включаючи аналіз вимог, проектування системи, реалізацію, тестування та розгортання.

Каскадна методологія (Waterfall) передбачає лінійний підхід, де кожна фаза проекту виконується послідовно: вимоги, проєктування, реалізація, тестування та впровадження. На початку процесу розробки проводиться аналіз вимог, які повинна мати розроблювана система:

– системні;
– функціональні;
– нефункціональні.

Системні вимоги визначають загальні характеристики та обмеження системи, що розробляється. Це можуть бути вимоги до апаратного забезпечення, операційної системи або мережевої інфраструктури.

Функціональні вимоги описують функції та поведінку системи. Ці вимоги описують, як система повинна працювати, які дії вона повинна виконувати та які результати повинна генерувати.

Нефункціональні вимоги описують якості та характеристики системи, а не її функції. Ці вимоги визначаються з точки зору якості, ефективності, безпеки, надійності, доступності та інших аспектів системи.

На етапі проєктування, відповідно до вимог визначаються компоненти, модулі та інші необхідні елементи. Наступним є етап реалізації, коли починається написання коду на основі визначеної архітектури, розробляються окремі модулі та компоненти, які потім поєднуються в одну систему. Наступний етап – це етап тестування, в ході якого проводяться перевірки якості розробленого ПЗ, та, у разі виявлення помилок, їх виправлення, після чого проводяться повторні тестування. Після успішного проходження етапу тестування, відбувається етап впровадження, коли розроблене програмне забезпечення встановлюється і запускається на цільовому середовищі або інфраструктурі.

Для реалізації даного проєкту було обрано каскадну методологію, оскільки проєкт має чіткі вимоги, і виконуватиметься найефективніше, використовуючи саме цю методологію.

2 ПРОЄКТУВАННЯ ЦИФРОВОГО МУЗИЧНОГО СЕРВІСУ

Для розробки програмної системи цифрового музичного сервісу вибрано подійно-орієнтовану архітектуру [7]. Це архітектурна модель розробки програмного забезпечення, яка наголошує на створенні, виявленні та обробці подій для ініціювання та координації поведінки різних компонентів у системі. У цій архітектурі потік даних і ініціювання дій керуються подіями, а не явним потоком керування.

Ключові компоненти архітектури, керованої подіями:

- події;
- творці подій;
- споживачі подій;
- посередник подій;
- обробка подій.

Події представляють зміни в системі чи її середовищі. Вони можуть генеруватися різними джерелами, такими як дії користувача, зміни стану системи тощо. Події зазвичай є структурованими повідомленнями, які містять відповідні дані та метадані про подію.

Творці подій відповідають за створення та публікацію подій, коли виникають певні умови. Вони ідентифікують і повідомляють систему про важливі події, роблячи їх доступними для використання іншими компонентами.

Споживачі подій підписуються на певні типи подій і відповідно реагують на них. Вони відстежують події та запускають відповідні процеси на основі отриманих даних. Споживачами можуть бути окремі компоненти або служби, які виконують певні завдання або надають певні функції.

Посередник подій діє як посередник між виробниками подій і споживачами. Він отримує опубліковані події від виробників і розповсюджує їх серед зареєстрованих споживачів на основі їхніх підписок. Посередник гарантує, що події доставляються надійно, асинхронно та в правильному порядку, якщо потрібно.

Обробка подій передбачає аналіз і реагування на події в реальному або майже реальному часі. Вона може включати фільтрацію подій, перетворення, агрегацію, кореляцію та комплексну обробку подій для отримання значущої інформації або ініціювання відповідних дій на основі моделей подій або умов.

Переваги архітектури, керованої подіями:

- слабкий зв’язок;
- масштабованість;
- розширюваність;
- відстеження подій.

Архітектура, керована подіями, сприяє слабкому зв’язку між компонентами, оскільки вони спілкуються опосередковано через події. Це підвищує гнучкість і дозволяє компонентам розвиватися без прямих залежностей.

Системи, керовані подіями, можуть ефективно масштабуватися шляхом розподілу обробки подій між кількома споживачами. Можна додавати нові компоненти для обробки збільшених обсягів подій, забезпечуючи кращу продуктивність і швидкість реагування.

Можна легко додати нову функціональність, додавши нових виробників і споживачів подій без зміни існуючих компонентів. Це робить систему більш розширюваною та адаптованою.

Події служать журналом важливих подій у системі, забезпечуючи запис дій і покращуючи пошук несправностей, аудит і аналітику.

Архітектура, керована подіями, широко використовується в різних сферах, таких як керовані подіями мікросервіси, аналітика в реальному часі, тощо. Це дозволяє створювати масштабовані, адаптивні та гнучкі системи, які можуть реагувати на події в режимі реального часу, підтримуючи інтеграцію і робочі процеси.

У зв’язку з проєктуванням десктоп-додатку цифрового музичного сервісу з використанням технології Windows Presentation Foundation та гнучких архітектурних моделей, необхідно визначити та проаналізувати функціональні та нефункціональні вимоги.

У ході проєктування програмної системи було визначено такі функціональні вимоги:

- Пошук та програвання музики. Додаток повинен мати функціонал для пошуку пісень за різними критеріями (назва, виконавець, жанр тощо) та програвання їх.
- Створення та управління плейлистами. Користувач повинен мати можливість створювати власні плейлисти, додавати до них пісні, видаляти та редагувати плейлисти.
- Різні режими програвання. Додаток може підтримувати різні режими програвання, такі як повтор пісні, повтор плейлисту, випадковий порядок програвання пісень тощо.
- Відтворення в фоновому режимі. Додаток повинен підтримувати програвання музики у фоновому режимі для більшої зручності користування продуктом.

Також, виявлено наступні нефункціональні вимоги:

- Продуктивність. Додаток повинен працювати швидко і ефективно навіть при великій кількості пісень та користувачів.
- Надійність. Додаток повинен бути стабільним та не виходити з ладу під час програвання музики або взаємодії з користувачем.
- Безпека. Забезпечення безпеки особистих даних користувачів, шифрування передачі даних та запобігання несанкціонованому доступу.
- Інтерфейс користувача. Інтуїтивно зрозумілий і привабливий інтерфейс користувача, який дозволяє легко орієнтуватися та використовувати функціонал додатку.
- Якість аудіо. Забезпечення високої якості аудіо програвання з мінімальною втратою якості звуку.

– Масштабованість. Здатність додатку працювати з ростом кількості пісень, користувачів та інших ресурсів без значного падіння продуктивності.

2.1. Розробка моделі предметної області цифрового музичного сервісу.

Модель предметної області включає в себе ідентифікацію та опис різних складових системи. У ході проєктування системи було визначено такі основні складові:

– користувачі;
– музичний контент;
– рекомендації;
– пошук;
– спільнота;
– аналітика.

Модель може описувати різні типи користувачів, їх профілі та характеристики, такі як вік, стать, музичні вподобання тощо. Також можна враховувати рівень активності користувачів та їх історію прослуховування.

Модель повинна включати опис різних типів музичного контенту, таких як пісні, альбоми, плейлисти, жанри музики тощо. Кожен тип контенту може мати свої характеристики, такі як назва, автор, тривалість, рейтинг тощо.

Модель може включати алгоритми рекомендацій, які базуються на історії прослуховування користувачів, їх вподобаннях та інших факторах. Рекомендації можуть враховувати популярність контенту, схожість з іншими треками або іншими користувачами, новинки та інші фактори.

Модель повинна включати можливість пошуку музичного контенту за різними критеріями, такими як назва, автор, жанр, тощо. Пошук може бути

реалізований за допомогою текстових запитів користувачів або за допомогою фільтрів.

Модель може включати функціонал, який дозволяє користувачам взаємодіяти між собою, ділитися плейлистами, тощо.

Модель може включати збір та аналіз даних про використання сервісу, таких як час прослуховування, популярність треків, поведінка користувачів тощо. Це дозволяє вдосконалювати рекомендації та вирішувати інші завдання, пов'язані з управлінням сервісом.

Оскільки музичний контент є однією з основних складових моделі цифрового музичного сервісу, виникає необхідність його зберігання в базі даних Microsoft Sql Server.

Для цього, розроблено реляційну базу даних, яка містить три таблиці:

- Songs;
- Playlists;
- PlaylistSongs.

Таблиця «Songs» міститиме окремі пісні, кожна з яких матиме наступні атрибути: номер, назва, автор, тривалість, тощо.

Таблиця «Playlists» представляє список плейлистів, причому кожен з них матиме унікальний ідентифікатор, назву і короткий опис.

Таблиця «PlaylistSongs» служить таблицею з’єднання, яка встановлює зв’язок «багато-до-багатьох» між плейлистами, та піснями. Вона містить посилання на зовнішній ключ до стовпця з унікальним ідентифікатором з таблиці «Playlists» і стовпця номером пісні з таблиці «Songs».

Ця структура бази даних дозволяє пов’язувати кілька пісень із кількома плейлистами, оскільки один плейлист може містити декілька пісень, а пісня може належати до кількох списків відтворення.

2.2. Розробка бізнес-моделі цифрового музичного сервісу.

Діаграма варіантів використання цифрового музичного сервісу, надає візуальне представлення взаємодій між користувачами та системою, фіксуючи ключові функції та поведінку програми, описуючи конкретні дії, які користувачі можуть виконувати в екосистемі програми. Ілюструючи ці випадки використання, діаграма дозволяє розробникам, дизайнерам і зацікавленим сторонам отримати всебічне розуміння того, як програма використовуватиметься та як її функції використовуватимуться в реальних сценаріях. Діаграма варіантів використання пропонує загальний огляд головних функцій програми, що робить її важливим інструментом для розробки та покращення взаємодії з користувачем.

На рисунку 2.1 зображено розроблену UML діаграму варіантів використання для цифрового музичного сервісу, розроблену в середовищі IBM Rational Software Architect [8].

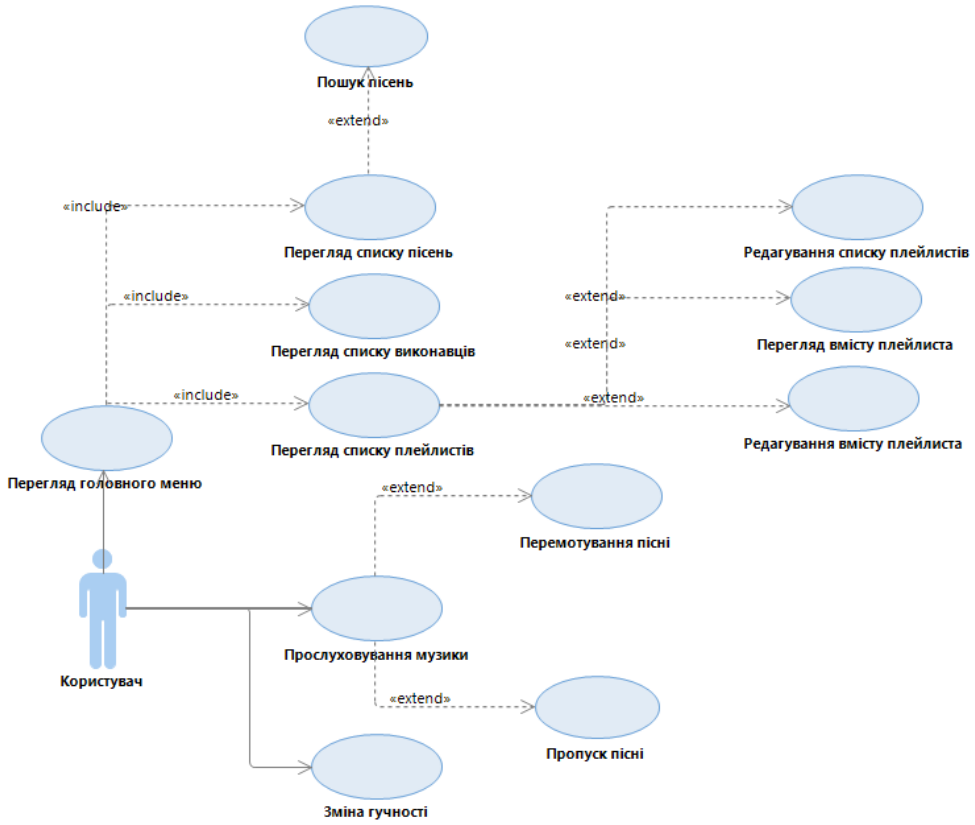

Рисунок 2.1 – Діаграма варіантів використання

Варіант використання «Перегляд головного меню»:

Короткий опис:

Цей варіант використання зображує процес перегляду головного меню у цифровому музичному сервісі, що містить основні функціональні та навігаційні можливості .

Основний потік подій:

1. Користувач відкриває цифровий музичний сервіс на своєму пристрої.
2. Користувач бачить головне меню, яке розташоване у бічній частині екрану.
3. Користувач може натиснути на будь-яку з наявних опцій, щоб отримати доступ до відповідної сторінки або функціоналу.
4. Користувач може навігуватися між різними опціями головного меню, переходити від однієї сторінки до іншої та використовувати усі функції сервісу.

Передумови:

Перед початком виконання цього варіанту використання користувач повинен мати доступ до цифрового музичного сервісу.

Післяумови:

Якщо варіант використання виконано успішно, користувач може переглянути головне меню цифрового музичного сервісу та отримати доступ до основних функцій, таких як пошук пісень, створення плейлистів, перегляд виконавців тощо. Користувач може ефективно навігувати по сервісу та використовувати його функціонал для задоволення своїх музичних потреб.

Варіант використання "Перегляд списку пісень":

Короткий опис:

Даний варіант використання відтворює процес перегляду списку пісень у цифровому музичному сервісі.

Основний потік подій:

1. Користувач відкриває цифровий музичний сервіс на своєму пристрої.
2. Користувач натискає на відповідну кнопку меню.

3. Сервіс відображає список пісень.
4. Користувач може прокручувати список пісень вгору або вниз для перегляду більшої кількості пісень.
5. Користувач може бачити назву пісні, виконавця та тривалість кожної пісні.

Передумови:

Перед початком виконання цього варіанту використання користувач повинен мати доступ до цифрового музичного сервісу, а також, виконавці або плейлисти, пов'язані з певними піснями, повинні бути доступні у базі даних сервісу.

Післяумови:

Якщо варіант використання виконано успішно, користувач може переглядати список пісень, пов'язаних з вибраним виконавцем або плейлистом у цифровому музичному сервісі. Користувач може отримати інформацію про кожну пісню та взаємодіяти з ними, наприклад, відтворювати, додавати до плейлисту або зберігати у свою бібліотеку.

Варіант використання «Пошук пісень»

Короткий опис:

Цей варіант використання окреслює процес пошуку пісень у цифровому музичному сервісі, який дозволяє користувачам знаходити конкретні пісні за назвою, виконавцем, або іншими параметрами.

Основний потік подій:

1. Користувач відкриває цифровий музичний сервіс на своєму пристрої.
2. Користувач відкриває список пісень.
3. Користувач натискає на поле пошуку.
4. Користувач вводить назву пісні, виконавця, або інші ключові слова, пов'язані з треком, який він шукає.
5. Сервіс обробляє введені дані і відображає результати пошуку у вигляді списку пісень, що відповідають введеним критеріям.
6. Користувач може прокручувати список пісень вгору або вниз для перегляду більшої кількості треків.

7. Користувач може натиснути на конкретну пісню з результатів пошуку, щоб отримати детальну інформацію про трек або відтворити його.
8. У разі неуспішного пошуку, коли сервіс не знаходить жодного треку, що відповідає введеним критеріям, користувач може спробувати змінити пошуковий запит або переглянути інші пропозиції, які надає сервіс.

Передумови:

Перед початком виконання цього варіанту використання користувач повинен мати доступ до цифрового музичного сервісу, окрім цього, пісні, виконавці, або плейлисти, які користувач шукає, повинні бути доступні у базі даних сервісу.

Післяумови:

Якщо варіант використання виконано успішно, користувач може шукати конкретні пісні за різними параметрами у цифровому музичному сервісі. Він отримує список результатів пошуку, який може прогортати та взаємодіяти з ними.

Варіант використання «Перегляд списку виконавців»

Короткий опис:

Варіант використання змальовує процес перегляду списку виконавців у цифровому музичному сервісі, що дозволяє користувачам знаходити та ознайомлюватися з різними музичними виконавцями.

Основний потік подій:

1. Користувач відкриває цифровий музичний сервіс на своєму пристрої.
2. Користувач переходить до сторінки, яка присвячена перегляду виконавців, або використовує функцію пошуку для знаходження конкретного виконавця.
3. Сервіс відображає список виконавців, доступних у базі даних, які можна переглядати.
4. Користувач може прокручувати список вгору або вниз для перегляду більшої кількості виконавців.
5. Користувач бачить назви виконавців та може вибрати конкретного виконавця, натиснувши на його назву або зображення.

6. Користувач може переглядати пісні та плейлисти виконавця, які доступні у базі даних сервісу.

Передумови:

Перед початком виконання цього варіанту використання користувач повинен мати доступ до цифрового музичного сервісу. Крім того, виконавці та їх музика повинні бути доступні у базі даних сервісу.

Післяумови:

Якщо варіант використання виконано успішно, користувач може переглядати список виконавців у цифровому музичному сервісі, його пісні та плейлисти. Цей варіант використання дозволяє користувачам знайти нових виконавців та відкрити для себе нову музику.

Варіант використання «Перегляд списку плейлистів»

Короткий опис:

Даний варіант використання описує процес перегляду списку плейлистів у цифровому музичному сервісі, що дозволяє користувачам знаходити та переглядати різноманітні музичні плейлисти, створені собою або самим сервісом.

Основний потік подій:

1. Користувач відкриває цифровий музичний сервіс на своєму пристрої.
2. Користувач переходить до сторінки, яка присвячена перегляду плейлистів.
3. Сервіс відображає список плейлистів, які доступні у базі даних.
4. Користувач може прокручувати список вгору або вниз для перегляду більшої кількості плейлистів.
5. Користувач бачить назви та опис плейлистів та може вибрати конкретний плейлист, натиснувши на його назву або зображення.
6. Сервіс відображає інформацію про вибраний плейлист.
7. Користувач може прогортати список треків, що входять до плейлисту, та переглядати їх деталі, такі як назва, виконавця та тривалість.
8. Користувач може натиснути на конкретний пісні, щоб відтворити його або взаємодіяти з ним.

Передумови:

Перед початком виконання цього варіанту використання користувач повинен мати доступ до цифрового музичного сервісу та плейлисти повинні бути доступні у базі даних сервісу.

Післяумови:

Якщо варіант використання виконано успішно, користувач може переглядати список плейлистів у цифровому музичному сервісі. Він може ознайомлюватися з інформацією про кожен плейлист, переглядати його пісні та взаємодіяти з ними. Цей варіант використання дозволяє користувачам знайти музичні плейлисти за своїми уподобаннями та насолодитися їх вмістом.

Варіант використання «Редагування списку плейлистів»

Короткий опис:

Даний варіант використання показує процес редагування списку плейлистів у цифровому музичному сервісі, що дозволяє користувачам створювати, редагувати та управляти власними музичними плейлистами.

Основний потік подій:

1. Користувач відкриває цифровий музичний сервіс на своєму пристрої.
2. Користувач переходить до розділу плейлистів.
3. Сервіс відображає список плейлистів, які доступні у базі даних.
4. Користувач може створити новий плейлист, натиснувши на відповідну кнопку.
5. Користувач може видалити існуючий плейлист, натиснувши на відповідну кнопку
6. Користувач має можливість зберігати зроблені зміни та оновлювати плейлист.

Передумови:

Перед початком виконання цього варіанту використання користувач повинен мати доступ до цифрового музичного сервісу.

Післяумови:

Якщо варіант використання виконано успішно, користувач може створювати або видаляти свої плейлисти у цифровому музичному сервісі. Цей варіант використання дозволяє користувачам налаштовувати свої власні музичні колекції та насолоджуватися персоналізованим музичним досвідом.

Варіант використання «Перегляд вмісту плейлиста»

Короткий опис:

Даний варіант використання зображує процес перегляду вмісту плейлиста у цифровому музичному сервісі, що дозволяє користувачам переглядати пісні, що входять до певного плейлиста.

Основний потік подій:

1. Користувач відкриває цифровий музичний сервіс на своєму пристрої.
2. Користувач переходить до розділу плейлистів.
3. Сервіс відображає список плейлистів, доступних користувачу або плейлисти, які користувач вже створив.
4. Користувач вибирає конкретний плейлист, натискаючи на його назву або зображення.
5. Сервіс відображає вміст обраного плейлиста, включаючи назви пісень, виконавців та додаткові деталі.
6. Користувач може прокручувати список треків вгору або вниз для перегляду більшої кількості треків.
7. Користувач може бачити назву пісні, виконавця та тривалість кожної пісні.
8. Користувач може натиснути на конкретний трек, щоб відтворити його.
9. Користувач може повернутися до списку плейлистів або продовжити переглядати вміст інших плейлистів.

Передумови:

Перед початком виконання цього варіанту використання користувач повинен мати доступ до цифрового музичного сервісу, а плейлисти, які користувач бажає переглянути, повинні бути доступні у базі даних сервісу.

Післяумови:

Якщо варіант використання виконано успішно, користувач може переглядати вміст плейлиста, включаючи пісні, що входять до нього, та отримувати додаткову інформацію про них. Цей варіант використання дозволяє користувачам ознайомитися з музичним вмістом плейлистів.

Варіант використання «Редагування вмісту плейлиста»

Короткий опис:

Даний варіант використання змальовує процес редагування вмісту плейлиста у цифровому музичному сервісі, що дозволяє користувачам додавати або видаляти пісні в плейлисті за їхніми вподобаннями.

Основний потік подій:

1. Користувач відкриває цифровий музичний сервіс на своєму пристрої.
2. Користувач переходить до розділу плейлистів.
3. Сервіс відображає список плейлистів, доступних користувачу або плейлисти, які користувач вже створив.
4. Користувач вибирає плейлист, який бажає редагувати, натискаючи на його назву або зображення.
5. Сервіс відображає вміст обраного плейлиста, включаючи назви пісень та виконавців.
6. Користувач може вибрати певну пісню, яку бажає додати до плейлиста, шляхом натискання відповідної іконки.
7. Користувач може видалити пісню з плейлиста, натиснувши на відповідну опцію або іконку.
8. Користувач може зберегти зміни, натиснувши на відповідну кнопку або опцію.

Передумови:

Перед початком виконання цього варіанту використання користувач повинен мати доступ до цифрового музичного сервісу, окрім цього, плейлисти, які користувач бажає редагувати, повинні бути доступні у базі даних сервісу.

Післяумови:

Якщо варіант використання виконано успішно, користувач може редагувати вміст плейлиста відповідно до своїх вподобань. Користувач може додавати або видаляти пісні у плейлисті. Цей варіант використання дозволяє користувачам створювати та налаштовувати власні плейлисти з улюбленими піснями, створюючи персоналізований музичний досвід.

Варіант використання «Прослуховування музики»

Короткий опис:

Даний варіант використання показує процес прослуховування музики у цифровому музичному сервісі, що дозволяє користувачам відтворювати бажані пісні на своєму пристрої.

Основний потік подій:

1. Користувач відкриває цифровий музичний сервіс на своєму пристрої.
2. Користувач вибирає потрібний розділ, наприклад, "Популярне", "Пісні", "Плейлисти" або використовує функцію пошуку для знаходження певної пісні чи виконавця.
3. Користувач вибирає конкретну пісню, яку бажає прослухати.
4. Сервіс починає відтворювати музику, починаючи з першої пісні.
5. Користувач може відтворювати музику в фоновому режимі, дозволяючи йому продовжувати використовувати інші програми або функції свого пристрою під час прослуховування.
6. Під час прослуховування пісні користувач може бачити інформацію про виконавця, назву, тривалість та інші додаткові деталі.
7. Користувач може переключатися між різними виконавцями або плейлистами, продовжуючи прослуховування вибраної музики.

Передумови:

Перед початком використання цього варіанту користувач повинен мати доступ до цифрового музичного сервісу. Також, відтворювана музика повинна бути доступна у базі даних сервісу.

Післяумови:

Якщо варіант використання виконано успішно, користувач може насолоджуватися музикою за допомогою музичного сервісу.

Варіант використання «Перемотування пісні»

Короткий опис:

Даний варіант використання описує процес перемотування пісні у цифровому музичному сервісі, що дозволяє користувачам пересуватися до певного моменту у пісні, якій вони прослуховують.

Основний потік подій:

1. Користувач відкриває цифровий музичний сервіс на своєму пристрої та вибирає пісню, яку він бажає прослухати.
2. Користувач може перетягнути бігунець програвача вперед або назад, щоб переміститься до конкретного моменту в пісні.
3. Після перемотування пісні до бажаного моменту, програвач продовжує відтворювати пісню з нового місця.
4. Користувач може повторити цей процес і перемотувати пісню у різних напрямках, щоб знайти потрібний момент.

Передумови:

Перед використанням цього варіанту користувач повинен мати доступ до цифрового музичного сервісу. Крім того, пісня, яку користувач хоче перемотати, повинна бути доступною у базі даних сервісу.

Післяумови:

Якщо варіант використання виконано успішно, користувач може легко перемотати пісню у музичному сервісі. Він може швидко переміститься до певного моменту в пісні, щоб прослухати улюблену частину або пропустити непотрібну.

Варіант використання «Пропуск пісні»

Короткий опис:

Даний варіант використання описує процес пропуску пісні у цифровому музичному сервісі, що дозволяє користувачам перейти до наступної пісні без прослуховування поточної до кінця.

Основний потік подій:

1. Користувач відкриває цифровий музичний сервіс на своєму пристрої та вибирає пісню, яку він бажає прослухати.
2. Після початку відтворення поточної пісні, користувач бачить панель управління програвача, на якій є кнопка пропуску пісні.
3. Користувач натискає кнопки пропуску пісні, щоб перейти до наступної або минулої пісні в черзі.
4. Програвач автоматично переходить до наступної або минулої пісні і розпочинає її відтворення.

Передумови:

Перед використанням цього варіанту користувач повинен мати доступ до цифрового музичного сервісу, а також пісня, яку користувач прослуховує, повинна мати наступну у списку.

Післяумови:

Якщо варіант використання виконано успішно, користувач може легко перейти до наступної або попередньої пісні у черзі без прослуховування поточної пісні до кінця. Цей варіант дозволяє швидко пропускати пісні, які користувач не бажає слухати, і продовжувати прослуховування музики без перерв.

Варіант використання «Зміна гучності»

Короткий опис:

Даний варіант використання відтворює процес зміни гучності у цифровому музичному сервісі, що дозволяє користувачам налаштовувати гучність звуку під час прослуховування музики.

Основний потік подій:

Користувач відкриває цифровий музичний сервіс на своєму пристрої та вибирає пісню, яку він бажає прослухати.

Після початку відтворення музики, користувач бачить панель управління програвача, на якій знаходиться регулятор гучності.

Користувач може змінити гучність, переміщуючи регулятор вправо або вліво. Вправо збільшує гучність, вліво зменшує.

Зміни гучності відбуваються в реальному часі, дозволяючи користувачеві налаштовувати звук під свої потреби.

Передумови:

Перед використанням цього варіанту користувач повинен мати доступ до цифрового музичного сервісу.

Післяумови:

Якщо варіант використання виконано успішно, користувач може змінювати гучність звуку під час прослуховування музики у музичному сервісі. Цей варіант дозволяє користувачам контролювати рівень звуку, щоб налаштувати його на свій смак, або врахувати оточуючі умови.

Для зображення послідовності виконання головного варіанту використання цифрового музичного сервісу, а саме «Прослуховування музики», у середовищі IBM RSA було створено діаграму послідовності (рисунок 2.2).

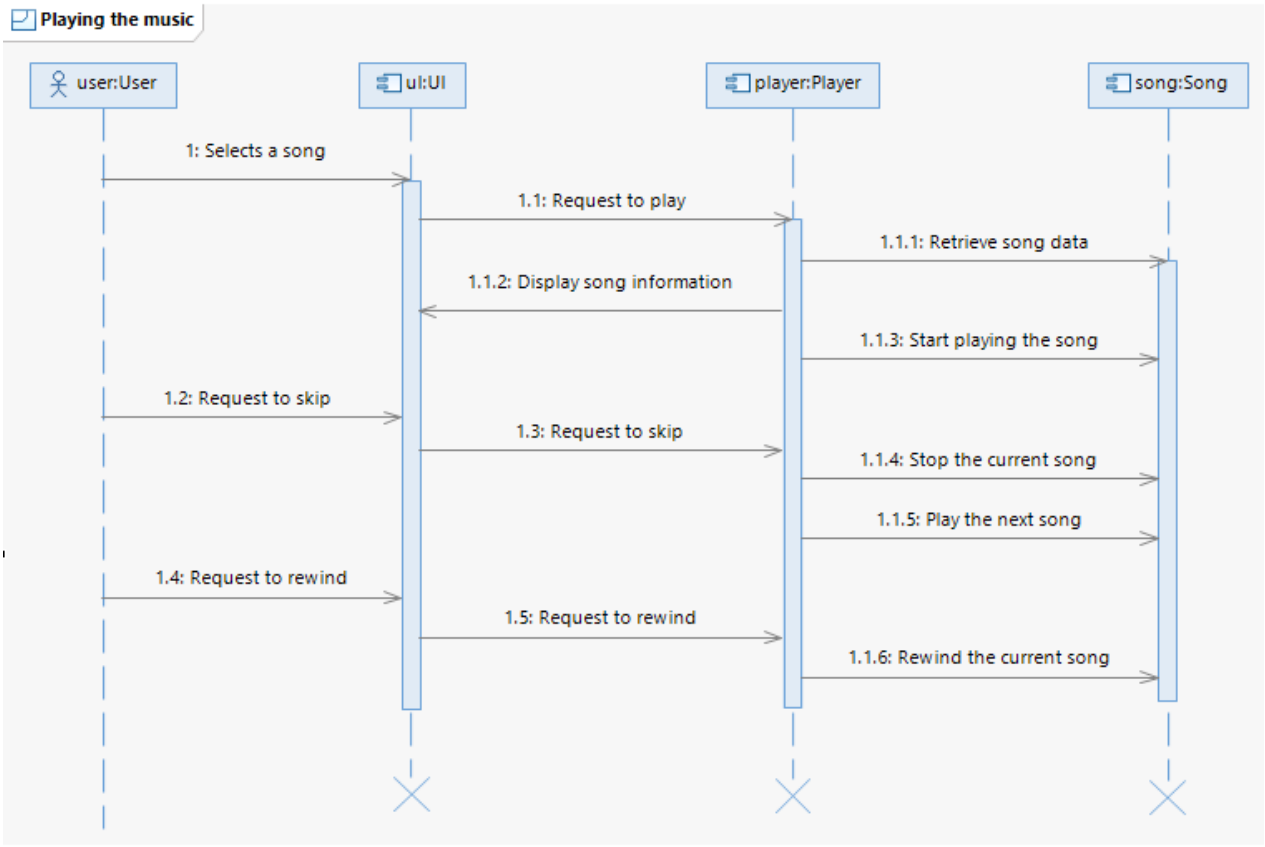

Рисунок 2.2 – Діаграма послідовності головного варіанту використання

На цій діаграмі зображено чотири об’єкти:

- User – представляє користувача та його дії, а саме вибір пісні, запит на пропуск пісні та запит на перемотування пісні.
- UI – користувацький інтерфейс системи. Він містить різні візуальні елементи, такі як кнопки та вікна, за допомогою яких користувач взаємодіє з системою.
- Player – компонент системи, який відповідає за внутрішню логіку: обробку запитів користувача, отримання інформації з бази даних, та її відтворення за допомогою користувацького інтерфейсу.
- Song – представляє музичний контент, котрий зберігається у розробленій базі даних.

2.3. Проєктування архітектури

Проєктування якісної архітектури відіграє вирішальну роль у забезпеченні безперебійної та приємної роботи користувача. Це охоплює базову структуру, компоненти та взаємодію, які забезпечують відтворення, керування та організацію музичних файлів. Добре продумана архітектура музичного сервісу не лише забезпечує ефективну функціональність, але й полегшує розширюваність, масштабованість та обслуговування.

Проєктування архітектури музичного сервісу передбачає прийняття ключових рішень щодо організації компонентів, зберігання даних, користувацького інтерфейсу та системної інтеграції. Метою проектування є створення надійної та гнучкої структури, здатної виконувати різні завдання, такі як перегляд списку музики, відтворення аудіофайлів, керування плейлистами та надання зручних елементів керування.

Дизайн інтерфейсу користувача відіграє ключову роль у забезпеченні візуально привабливого та інтуїтивно зрозумілого зовнішнього вигляду програми

для користувачів. Архітектура повинна включати відповідні компоненти користувацького інтерфейсу, такі як вікна, елементи керування та навігаційні меню, щоб забезпечити безперешкодну взаємодію з музичним плеєром. Крім того, слід звернути увагу на швидкість реагування, доступність і можливості налаштування, щоб задовольнити різноманітні уподобання користувачів.

Таким чином, проєктування архітектури музичного сервісу вимагає ретельного врахування різних факторів, таких як, управління даними, дизайн користувацького інтерфейсу, розширюваність і масштабованість. Добре продумана архітектура не лише сприяє ефективному відтворенню музики, але й дозволяє легко організовувати, розширювати та масштабувати розроблену систему.

З цією метою, у середовищі IBM RSA було створено діаграму класів для розроблюваного цифрового музичного сервісу, зображену на рисунку 2.3.

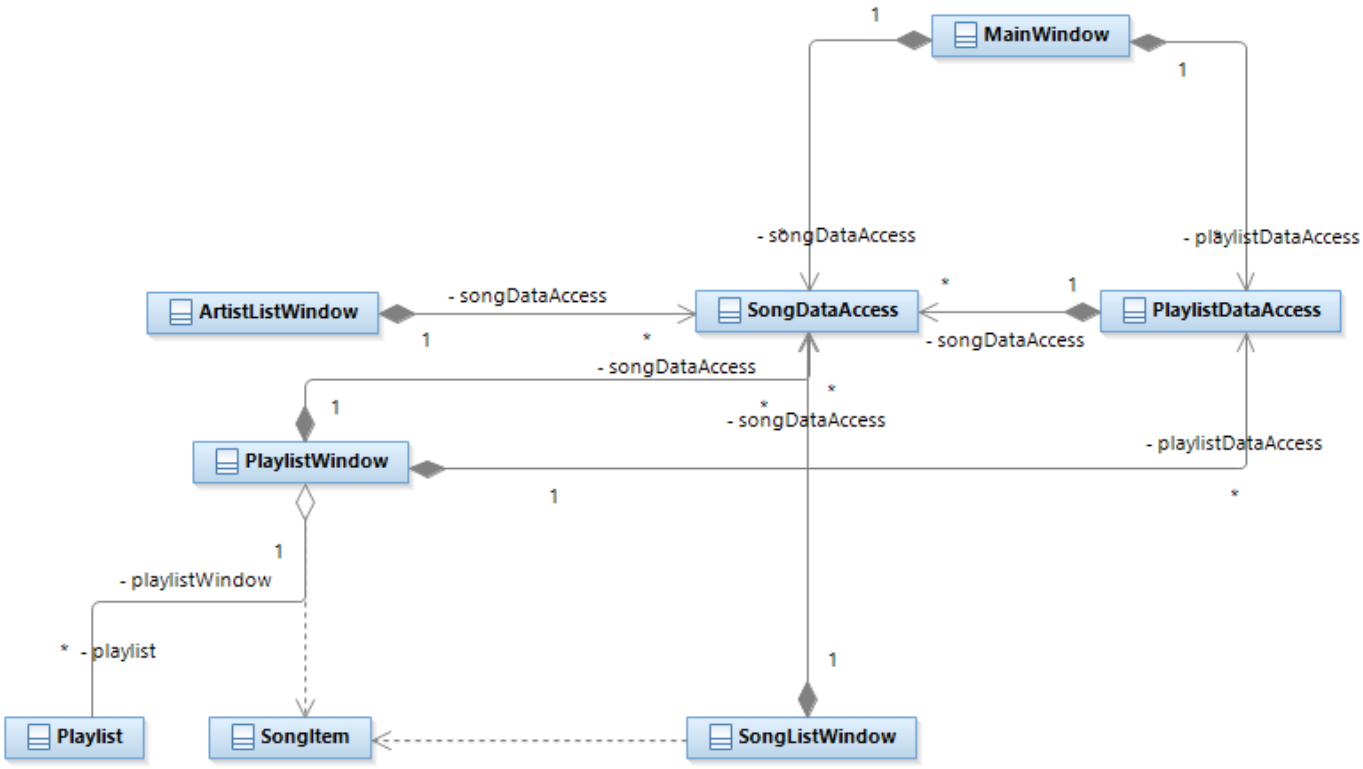

Рисунок 2.3 – Діаграма класів системи

Дана діаграма класів зображує структуру та відношення між класами у програмній системі цифрового музичного сервісу.

Головними класами цієї системи є: MainWindow, SongDataAccess, PlaylistDataAccess, SongListWindow, PlaylistWindow, ArtistListWindow, SongItem, Playlist.

Клас MainWindow представляє головне вікно додатку цифрового музичного сервісу. Він має два відношення композиції з класами SongDataAccess та PlaylistDataAccess, адже, відповідно до подійно-орієнтованої архітектури, цей клас буде за допомогою обробників подій буде отримувати та обробляти вміст розроблених баз даних.

Клас SongDataAccess представляє логіку взаємодій з базою даних, що містить усю необхідну інформацію про пісні (таку як назву, автора, тривалість тощо).

Схожим чином, клас PlaylistDataAccess – реалізація методів взаємодії з базою даних, яка містить інформацію про існуючі у системі плейлисти. Він містить відношення композиції до класу SongDataAccess, адже плейлисти, доступні у системі містять у собі список пісень. Окрім цього, він містить відношення агрегації та залежності до класів Playlist та SongItem відповідно, для коректної роботи інтерфейсу користувача.

Клас SongListWindow представляє собою вікно, у котрому виводиться список усіх пісень, тому він також має відношення композиції до класу SongDataAccess.

З тієї ж причини що і клас PlaylistDataAccess, клас PlaylistWindow, котрий представляє собою вікно перегляду плейлистів, містить відношення композиції до класів SongDataAccess та PlaylistDataAccess, для того щоб коректно реалізовувати роботу плейлистів у системі цифрового музичного сервісу.

Клас ArtistListWindow має відношення композиції до класу SongDataAccess тому, що інформація про виконавців міститься у тій же ж базі даних, що і пісні.

Класи Playlist та SongItem - користувацькі елементи системи. Вони визначаються властивостями залежностей, які дозволяють прив’язувати дані та взаємодіяти з xaml кодом для створення користувацького інтерфейсу.

3 КОНСТРУЮВАННЯ ЦИФРОВОГО МУЗИЧНОГО СЕРВІСУ

3.1. Реалізація ключових класів

Реалізація ключових класів – це критично важливий аспект розробки системи музичного сервісу. Ці класи слугують будівельними блоками програми, інкапсулюючи основні функціональні можливості та структури даних, необхідні для відтворення музики, керування нею та взаємодії з користувачем. Ефективно реалізувавши ці класи, можна створити надійний і багатофункціональний музичний сервіс, який задовольняє потребам користувачів.

Одним з ключових класів у системі музичних плеєрів є клас "MainWindow" (рисунок 3.1). Він є ключовим класом, що відповідає за керування головним вікном програми музичного сервісу. Він успадковується від стандартного класу Window [9] і містить різні змінні-члени та методи для управління функціональністю музичного плеєра.

Нижче наведено опис ключових компонентів і функціональності класу MainWindow:

Змінні-члени:

- songDataAccess: екземпляр класу SongDataAccess для доступу до даних пісень з бази даних.
- playlistDataAccess: екземпляр класу PlaylistDataAccess для доступу до даних плейлиста з бази даних.
- mediaPlayer: екземпляр класу MediaPlayer для відтворення пісень.
- songs: список об'єктів SongItem, що представляють пісні у музичному плеєрі.
- songQueue: черга об'єктів SongItem, що представляють пісні у черзі на відтворення.
- songHistory: стек об'єктів SongItem, що представляють історію пісень для перемотування назад.

- isPlaying: булевий прапорець, що вказує на те, чи відтворюється пісня в даний момент.
- isDraggingSlider: булевий прапорець, що вказує на те, чи перетягується користувачем повзунок прогресу пісні.
- sliderTimer: об'єкт DispatcherTimer для оновлення повзунка прогресу пісні.

Конструктор: ініціалізує змінні-члени та встановлює з'єднання з базою даних.

Обробники подій:

- Border_MouseDown: обробляє подію відведення миші вниз для перетягування вікна.
- Window_Loaded: завантаження вікна та ініціалізація різних компонентів.
- PlayButton_Click: відтворення або призупинення поточної пісні.
- SkipNextButton_Click: відтворення наступної пісні.
- SkipPreviousButton_Click: відтворення попередньої пісні в історії.
- ShuffleButton_Click: перемішування черги пісень.
- SongButton_Click: відображення списку пісень.
- ArtistButton_Click: відображення списку виконавців.
- PlaylistButton_Click: відображення списку плейлистів.
- slider_DragStarted: встановлення прапорця перетягування.
- slider_DragCompleted: пошук медіаплеєра у новій позиції.
- volumeSlider_ValueChanged: регулювання гучності медіаплеєра.

Допоміжні методи:

- LoadSongs: завантажує пісні з бази даних і додає їх до черги пісень.
- LoadDailySongs: завантажує певний набір пісень (наприклад, певного виконавця) у контейнер інтерфейсу користувача.
- PlaySong: відтворює вибрану пісню у медіаплеєрі та оновлює інтерфейс програвача.

- UpdatePlayerWithActiveSong: оновлює інтерфейс програвача деталями активної пісні.
- SetPopularSongsIsActive: встановлює активний стан популярних пісень на основі активної пісні у базі даних.
- OutputPlaylists: отримує плейлисти з бази даних і додає їх до контейнера інтерфейсу.
- StartSliderTimer: запускає таймер для оновлення повзунка прогресу пісні.
- StopSliderTimer: зупиняє таймер оновлення повзунка прогресу композиції.
- UpdateSliderPosition: оновлює позицію повзунка прогресу композиції на основі поточного відтворення композиції.

Загалом, клас MainWindow відповідає за основну функціональність програми музичного плеєра, включно із завантаженням пісень, відтворенням і призупиненням пісень, керуванням чергою відтворення та історією, оновленням інтерфейсу плеєра, обробкою взаємодії з користувачем і взаємодією з базою даних пісень та списків відтворення.

```
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using MusicPlayer.UserControls;
using MusicPlayer.Data;
using MusicPlayer.Views;
using System;
using MahApps.Metro.IconPacks;
using System.Linq;
using System.Windows.Threading;

namespace MusicPlayer
{
    3 references
    public partial class MainWindow : Window
    {
        private SongDataAccess songDataAccess;
        private PlaylistDataAccess playlistDataAccess;
        private MediaPlayer mediaPlayer;
        private List<SongItem> songs;
        private Queue<SongItem> songQueue;
        private Stack<SongItem> songHistory;
        private bool isPlaying = false;
        private bool isDraggingSlider = false;
        private DispatcherTimer sliderTimer;

        0 references
        public MainWindow()
        {
            InitializeComponent();

            string connectionString = @"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\riser
            \Documents\MusicPlayer.mdf;Integrated Security=True;Connect Timeout=30";
            songDataAccess = new SongDataAccess(connectionString);
            playlistDataAccess = new PlaylistDataAccess(connectionString);
            mediaPlayer = new MediaPlayer();
            songs = new List<SongItem>();
            songQueue = new Queue<SongItem>();
            songHistory = new Stack<SongItem>();
        }
```

Рисунок 3.1 – Ключовий код класу «MainWindow»

Іншим важливим класом є клас "PlaylistWindow". Цей клас відповідає за керування списками відтворення та їхніми піснями. Він включає функціональність для створення, видалення та модифікації списків відтворення, а також додавання та вилучення пісень зі списку відтворення.

Змінні-члени:

– songDataAccess: екземпляр класу SongDataAccess для доступу до даних пісень з бази даних.

- playlistDataAccess: екземпляр класу PlaylistDataAccess для доступу до даних плейлиста з бази даних.
- playlists: список об'єктів Playlist, що представляють плейлисти.
- selectedPlaylistName: рядкова змінна для зберігання назви поточного вибраного списку відтворення.

Конструктори: клас має два конструктори, кожен з яких приймає різні параметри (рисунок 3.2).

Обробники подій:

- PlaylistControl_MouseDown отримує натиснутий елемент управління Playlist, знаходить відповідний плейлист у списку і викликає метод OpenSongListWindow, передаючи йому назву вибраного плейлиста.
- AddButton_Click та RemoveButton_Click виконують відповідні операції доступу до даних і викликають метод OpenSongListWindow, щоб оновити відображення списку пісень.
- CreateButton_Click та DeleteButton_Click: виконують відповідні операції доступу до даних і викликають метод UpdatePlaylistDisplay для оновлення відображення плейлиста.

Допоміжні методи:

- LoadPlaylists: ітераційно створює список об'єктів і їх відображення у вікні.
- OpenSongListWindow: відкриває нове вікно для відображення списку пісень для вибраного плейлиста.
- GetPlaylistIdFromDatabase: отримує ідентифікатор плейлиста для заданої назви плейлиста з класу доступу до даних.
- ConvertToSongItems: перетворює список номерів пісень на список об'єктів SongItem.
- UpdatePlaylistDisplay: оновлює відображення списку відтворення, очищаючи наявний вміст та отримуючи оновлений список плейлистів з класу доступу до даних.

Загалом, клас PlaylistWindow надає функціональність для керування плейлистами та їхніми піснями у програмі музичного плеєра. Він взаємодіє з класами доступу до даних, створює та оновлює елементи керування, а також реагує на дії користувача для виконання різних операцій зі списками відтворення та піснями.

```
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using MusicPlayer.Data;
using MusicPlayer.UserControls;

namespace MusicPlayer.Views
{
    5 references
    public partial class PlaylistWindow : Window
    {
        private PlaylistDataAccess playlistDataAccess;
        private SongDataAccess songDataAccess;
        private List<Playlist> playlists;
        private string selectedPlaylistName;
        0 references
        public PlaylistWindow(List<int> playlistIds, PlaylistDataAccess dataAccess, SongDataAccess songDataAccess)
        {
            InitializeComponent();
            playlistDataAccess = dataAccess;
            this.songDataAccess = songDataAccess;
            // Create Playlist UserControls based on playlistIds and add them to the StackPanel
            foreach (int playlistId in playlistIds)
            {
                // Retrieve playlist details using playlistDataAccess and create Playlist UserControls
                List<int> songNumbers = playlistDataAccess.GetPlaylistSongs(playlistId);
                Playlist playlist = new Playlist
                {
                    Title = $"Playlist {playlistId}",
                    Desc = $"Description {playlistId}",
                    IsActive = false
                };
                playlistStackPanel.Children.Add(playlist);
            }
        }
        1 reference
        public PlaylistWindow(List<Playlist> playlists, PlaylistDataAccess dataAccess, SongDataAccess songDataAccess)
        {
            InitializeComponent();
            this.playlists = playlists;
            playlistDataAccess = dataAccess;
            this.songDataAccess = songDataAccess;
            // Call the LoadPlaylists method to populate the stack panel
            LoadPlaylists();
        }
```

Рисунок 3.2 – Ключовий код класу «PlaylistWindow»

Клас «SongListWindow» (рисунок 3.3) представляє вікно, яке відображає список пісень і дозволяє користувачам шукати та взаємодіяти з ним.

Змінні-члени:

- songs: Спостережувана колекція об'єктів SongItem, що представляє список пісень, які відображаються у вікні.

Конструктор: ініціалізує змінні-члени.

Обробники подій:

- SearchTextBox_KeyDown: отримує текст пошуку, виконує пошук за допомогою методу PerformSearch() і оновлює інтерфейс користувача результатами пошуку.
- SongItem_MouseLeftButtonUp: ідентифікує натиснуту пісню і перемикає її властивість IsActive. Якщо раніше була активна пісня, то вона стає неактивною, а її стан зберігається в базі даних за допомогою методу SaveSongState.
- SearchTextBox_GotFocus: очищає текст-заповнювач і встановлює колір тексту на білий.
- SearchTextBox_LostFocus: встановлює текст-заповнювач і колір тексту до значень за замовчуванням.

Допоміжні методи:

- GetSongs: отримує повний список пісень з бази даних за допомогою класу SongDataAccess.
- PerformSearch: виконує пошук пісень на основі наданого пошукового тексту.
- UpdateSongList: оновлює інтерфейс з наданим списком пісень.
- SaveSongState: зберігає стан пісні у базі даних. Використовує клас SongDataAccess для оновлення стану пісні.

Загалом, клас SongListWindow надає функціональність для відображення списку пісень, пошуку пісень за назвою або виконавцем, перемикання активного стану пісень та збереження стану пісень у базі даних.

```
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Input;
using MusicPlayer.UserControls;
using MusicPlayer.Data;
using System.Windows.Media;
using System.Collections.ObjectModel;
using System.Windows.Controls;
using System.Linq;

namespace MusicPlayer.Views
{
    4 references
    public partial class SongListWindow : Window
    {
        6 references
        public ObservableCollection<SongItem> Songs { get; set; }

        1 reference
        public SongListWindow(List<SongItem> songs)
        {
            InitializeComponent();
            Songs = new ObservableCollection<SongItem>();
            DataContext = this;

            // Retrieve the full songs list
            List<SongItem> fullSongsList = GetSongs();

            // Populate the Songs collection with the full songs list
            foreach (var song in fullSongsList)
            {
                Songs.Add(song);
            }
            // Perform an initial search for an empty string (show all songs)
            List<SongItem> searchResults = PerformSearch(string.Empty);

            // Update the UI with the initial search results
            UpdateSongList(searchResults);
        }
```

Рисунок 3.3 – Ключовий код класу «SongListWindow»

Клас ArtistListWindow (рисунок 3.4) у наведеному коді представляє вікно, яке відображає список виконавців і дозволяє користувачам шукати та фільтрувати виконавців. Нижче наведено опис його ключових компонентів та функціональності:

Змінні-члени:

- Artists: представляє список виконавців, які відображаються у вікні.
- artistCollectionView: використовується для фільтрування та відображення виконавців у ListBox.
- songDataAccess: екземпляр класу SongDataAccess для доступу до даних пісні з бази даних.

Конструктор: ініціалізує змінні-члени.

Методи та обробники подій:

- SearchTextBox_TextChanged: спрацьовує, коли змінюється текст у пошуковому полі. Якщо текст пошуку порожній, він показує всіх виконавців, очищаючи фільтр. В іншому випадку він фільтрує виконавців на основі тексту пошуку.
- SearchTextBox_GotFocus: очищає текст заповнювача і встановлює колір тексту на білий.
- SearchTextBox_LostFocus: якщо текстове поле порожнє, встановлює текст-заповнювач і колір тексту до значень за замовчуванням.
- ArtistSearchTextBox_KeyDown: отримує текст пошуку, виконує пошук виконавця за допомогою методу PerformArtistSearch і оновлює інтерфейс з результатами пошуку.
- PerformArtistSearch: виконує пошук виконавців на основі наданого пошукового тексту.
- UpdateArtistList: оновлює інтерфейс з наданим списком виконавців.

Загалом, клас ArtistListWindow забезпечує функціональність для відображення списку виконавців, пошуку та фільтрації виконавців на основі даних, введених користувачем, та оновлення інтерфейсу користувача за результатами пошуку. Він взаємодіє з класом SongDataAccess, щоб отримати список усіх виконавців і виконати пошук виконавця.

```
using System;
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Media;
using System.Linq;
using MusicPlayer.Data;
using System.Windows.Input;
using System.ComponentModel;

namespace MusicPlayer.Views
{
    4 references
    public partial class ArtistListWindow : Window
    {
        4 references
        public List<string> Artists { get; set; }
        private ICollectionView artistCollectionView;
        private SongDataAccess songDataAccess; // Add a field for accessing song data

        1 reference
        public ArtistListWindow(List<string> artists)
        {
            InitializeComponent();
            Artists = artists;
            DataContext = this;

            // Create a CollectionView for the Artists list
            artistCollectionView = CollectionViewSource.GetDefaultView(Artists);

            // Set the ListBox ItemsSource to the CollectionView
            artistListBox.ItemsSource = artistCollectionView;

            // Initialize the SongDataAccess instance
            songDataAccess = new SongDataAccess(@"Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\riser
            \Documents\MusicPlayer.mdf;Integrated Security=True;Connect Timeout=30");
        }
```

Рисунок 3.4 – Ключовий код класу «ArtistListWindow»

Таким чином, реалізація ключових класів демонструє успішне створення додатку музичного сервісу. Загалом, ці ключові класи забезпечують необхідну функціональність для керування та відтворення пісень, відображення плейлистів, пошуку пісень та виконавців, а також взаємодії з базою даних для отримання та збереження даних про пісні та плейлисти. Разом вони формують основні компоненти програми музичного плеєра.

3.2 Розробка GUI

Розробка графічного інтерфейсу користувача (GUI) відіграє життєво важливу роль у сучасному ринку програмного забезпечення. Графічні інтерфейси надають користувачам візуально привабливі та інтуїтивно зрозумілі засоби взаємодії зі складними програмними системами, роблячи їх доступними для широкої аудиторії.

Розробка ефективного графічного інтерфейсу передбачає врахування різних факторів. Перш за все, інтерфейс повинен бути візуально привабливим, використовувати добре продуману графіку, та іконки для створення естетично приємного зовнішнього вигляду. Крім того, дуже важливою є швидкість програмної системи, оскільки користувачі очікують плавної взаємодії без будь-яких помітних затримок.

Простота і орієнтований на користувача дизайн мають першорядне значення при розробці графічного інтерфейсу. Інтерфейс повинен бути інтуїтивно зрозумілим, дозволяючи користувачам легко орієнтуватися в різних функціях і користуватися функціями додатку без особливих зусиль. Це передбачає продумане розміщення елементів керування, логічну організацію інформації та дотримання встановлених шаблонів і конвенцій дизайну.

Таким чином, розробка графічного інтерфейсу є критично важливим аспектом розробки програмного забезпечення, що формує сприйняття та враження користувача програми. Приділяючи достатньо уваги візуальній привабливості, швидкості реагування, зручності використання і використовуючи правильні інструменти, будуть створені графічні інтерфейси, які не лише відповідають очікуванням користувачів, але й підвищують загальну цінність програмного забезпечення.

З метою створення сучасного та привабливого інтерфейсу було використано технологію Windows Presentation Foundation – це потужна платформа від Microsoft для створення сучасних та інтерактивних графічних інтерфейсів користувача для Windows-додатків. Використовуючи можливості XAML (eXtensible Application

Markup Language) та фреймворку .NET, WPF дозволяє створювати користувацькі інтерфейси з широким вибором мультимедійного контенту, анімацією та гнучкими макетами.

Однією з ключових переваг використання WPF для розробки графічних інтерфейсів є розділення інтерфейсу та бізнес-логіки. За допомогою архітектурного шаблону Model-View-ViewModel (MVVM) створено чіткий поділ між візуальним представленням (View), базовими даними та поведінкою (Model) і проміжним шаром, який з'єднує View і Model (ViewModel).

Окрім того, WPF надає широкий спектр вбудованих елементів керування та панелей компонування, які можна легко налаштовувати та стилізувати відповідно до бажаного вигляду та стилю програми. WPF пропонує широкий набір елементів управління, таких як кнопки, текстові поля, списки або сітки даних, які можна легко розташувати і стилізувати для створення візуально привабливих інтерфейсів.

Ще однією сильною стороною WPF є його потужні можливості зв'язування даних. За допомогою прив'язки даних можна встановити зв'язок між елементами інтерфейсу та основним джерелом даних, гарантуючи, що будь-які зміни в даних автоматично відображатимуться в інтерфейсі, і навпаки. Це спрощує синхронізацію даних між інтерфейсом і внутрішньою логікою, забезпечуючи безперебійну роботу користувача.

Таким чином, за допомогою технології WPF було розроблено візуальну складову десктоп-додатку цифрового музичного сервісу. На рисунку 3.5 зображено загальний вигляд вікна розробки WPF-додатку та головну сторінку цифрового музичного сервісу.

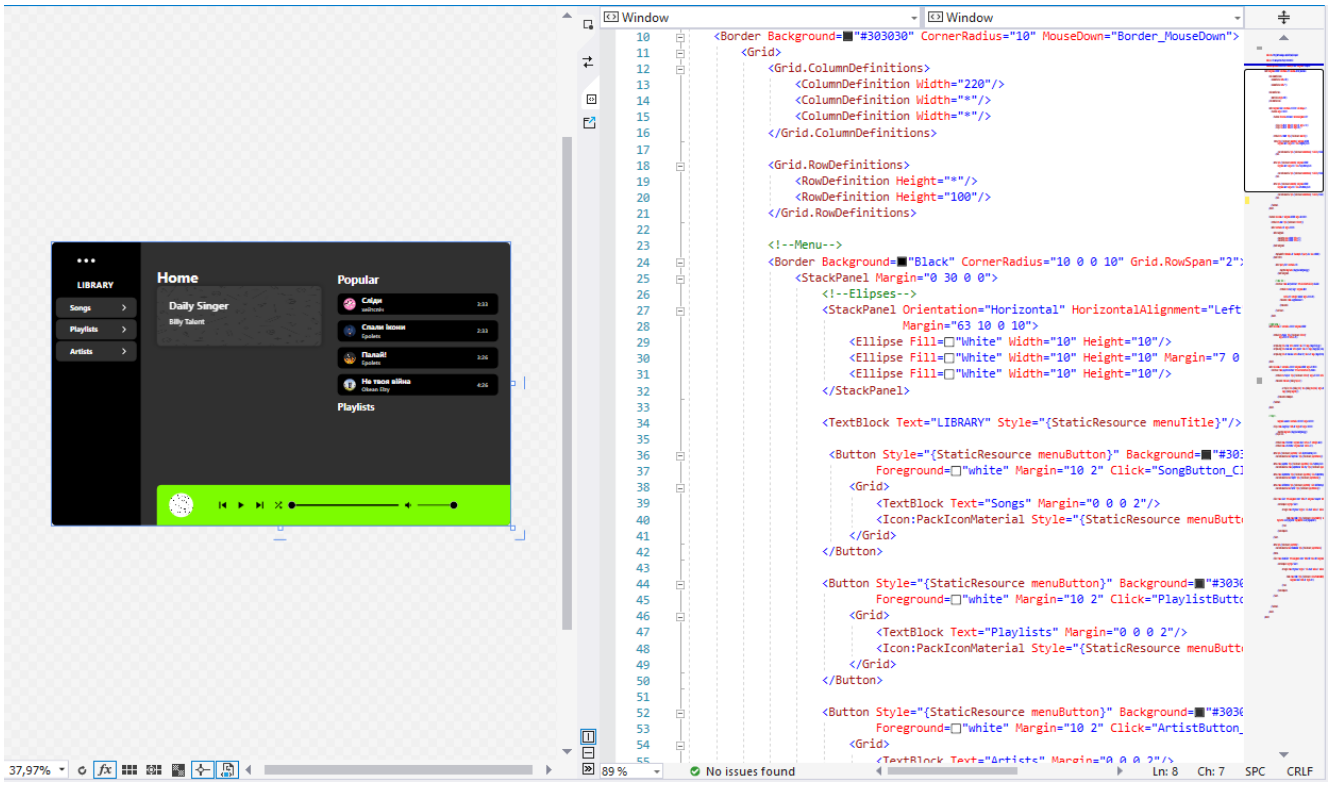

Рисунок 3.5 – Загальний вигляд вікна розробки

Таким же ж чином було створено додаткові, з універсальним мінімалістичним дизайном, сторінки перегляду списків пісень (рисунок 3.6), виконавців та плейлистів, котрі, за допомогою функціоналу зв’язування даних WPF динамічно заповнюються вмістом баз даних при користуванні додатком.

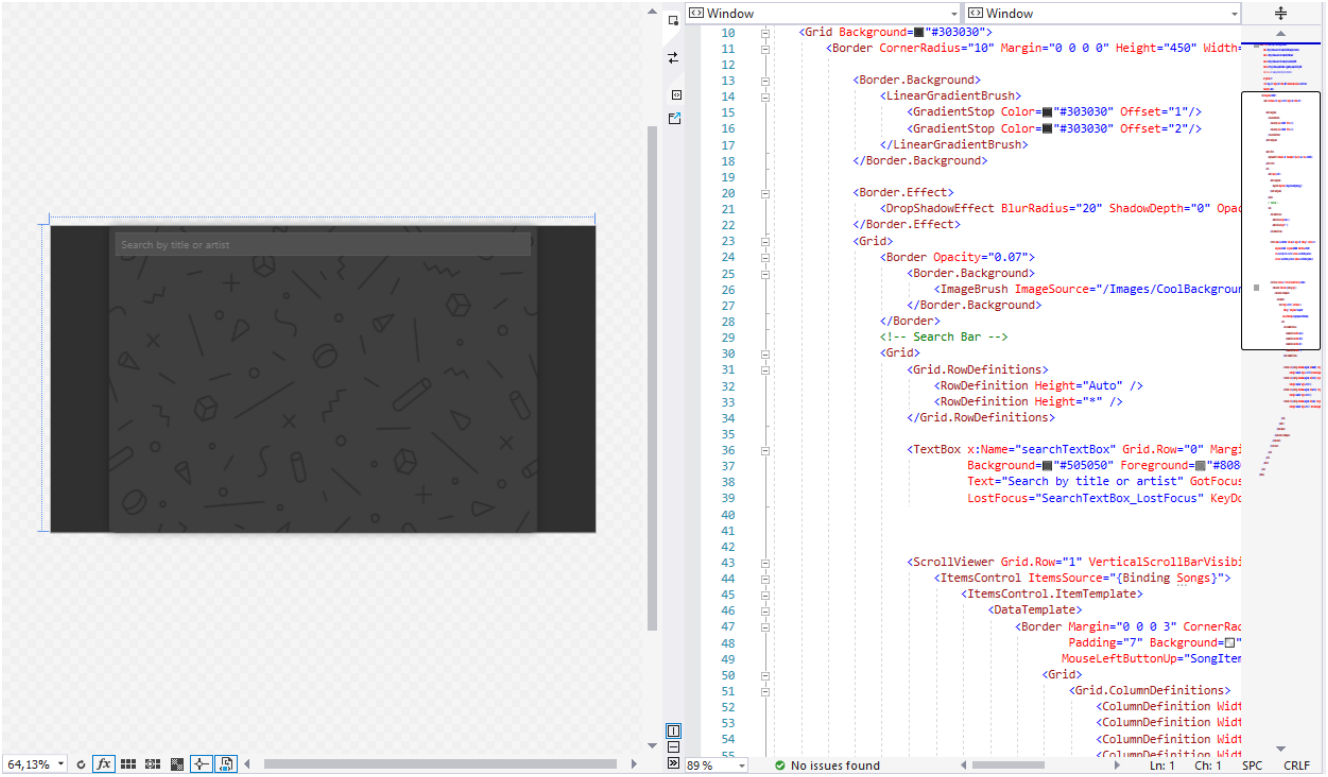

Рисунок 3.6 – Вікно перегляду списку пісень

Вікно перегляду списку пісень використовує можливість зв’язування даних, для того щоб використати окремий користувацький елемент SongItem, який визначає зовнішній вигляд і поведінку окремого елемента. Він містить елемент Border зі стилем, який змінює колір фону при наведенні на нього миші або коли властивість IsActive елемента SongItem має значення true. Всередині Border розміщено макет Grid з чотирма колонками, кожна з яких містить елемент TextBlock. Ці текстові блоки мають прив'язки, подібні до прив'язок у вікні SongListWindow XAML, для відображення властивостей елемента SongItem "Номер", "Назва", "Виконавець" і "Час". Розмір шрифту, вага і поля налаштовуються для досягнення бажаного вигляду.

Окрім цього, було створено декілька спеціальних стилів (рисунок 3.7), котрі використовуються у створенні якісного графічного дизайну музичного сервісу.

```
<Style x:Key="menuTitle" TargetType="TextBlock">
    <Setter Property="Foreground" Value="White"/>
    <Setter Property="FontSize" Value="22"/>
    <Setter Property="FontWeight" Value="Bold"/>
    <Setter Property="Margin" Value="63 30 0 15"/>
</Style>

<Style x:Key="menuButton" TargetType="Button">
    <Setter Property="Foreground" Value="#cfcfcf"/>
    <Setter Property="Height" Value="50"/>
    <Setter Property="Margin" Value="30 3 0 3"/>
    <Setter Property="FontSize" Value="18"/>
    <Setter Property="FontWeight" Value="Bold"/>
    <Setter Property="FocusVisualStyle" Value="{x:Null}"/>
    <Setter Property="Tag" Value="Collapsed"/>

    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                <Border x:Name="border" Background="#303030" CornerRadius="10">
                    <Grid>
                        <ContentPresenter HorizontalAlignment="Stretch" VerticalAlignment="Center" Margin="35 0 0 0"/>
                    </Grid>
                </Border>
                <ControlTemplate.Triggers>
                    <Trigger Property="IsMouseOver" Value="True">
                        <Setter TargetName="border" Property="Background" Value="LawnGreen"/>
                        <Setter Property="Foreground" Value="White"/>
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

<Style x:Key="menuButtonIcon" TargetType="Icon:PackIconMaterial">
    <Setter Property="VerticalAlignment" Value="Center"/>
    <Setter Property="Margin" Value="0 0 23 0"/>
    <Setter Property="Kind" Value="ChevronRight"/>
    <Setter Property="HorizontalAlignment" Value="Right"/>
    <Setter Property="Visibility" Value="{Binding Path=Tag, RelativeSource={RelativeSource FindAncestor, AncestorType={x:Type Button}}}"/>
</Style>
```

Рисунок 3.7 – Фрагмент коду розроблених спеціальних стилів

Розроблені спеціальні стилі розроблено для зручного впровадження (рисунок 3.8.) стандартизованих стилів у різноманітних файлах програми.

```
<TextBlock Text="LIBRARY" Style="{StaticResource menuTitle}"/>

 <Button Style="{StaticResource menuButton}" Background="#303030"
       Foreground="white" Margin="10 2" Click="SongButton_Click">
    <Grid>
        <TextBlock Text="Songs" Margin="0 0 0 2"/>
        <Icon:PackIconMaterial Style="{StaticResource menuButtonIcon}" Visibility="Visible"/>
    </Grid>
</Button>
```

Рисунок 3.8 – Приклад застосування розроблених спеціальних стилів

Таким чином, за допомогою технології Windows Presentation Foundation було створено сучасний та зручний графічний інтерфейс користувача, котрий привабить більшу кількість користувачів.

3.3 Тестування програмного забезпечення та перевірка якості

Тестування програмного забезпечення та контроль його якості є критично важливими компонентами процесу розробки програмного забезпечення. Оскільки сучасні програмні системи стають дедалі складнішими та інтегрованими в різні галузі, забезпечення їхньої надійності, функціональності та продуктивності є надзвичайно важливим. Тестування програмного забезпечення передбачає систематичне оцінювання програмної системи або програми для виявлення дефектів, помилок і вразливостей, які потенційно можуть вплинути на її зручність, безпеку або ефективність. З іншого боку, контроль якості зосереджується на загальній якості програмного продукту, що охоплює не лише відсутність дефектів, але й відповідність галузевим стандартам, вимогам користувачів та найкращим практикам. Впроваджуючи надійні методології тестування та заходи контролю якості, організації можуть підвищити надійність, стабільність і задоволеність користувачів програмним продуктом, що в кінцевому підсумку призведе до покращення взаємодії з клієнтами та більшого успіху в бізнесі.

Тому, з метою забезпечення контролю якості розробленого десктоп-додатку було проведено його тестування. Спершу, запустивши додаток, на головній сторінці коректно виводяться усі необхідні елементи (рисунок 3.9). Активна пісня, що програється на даний момент також коректно виявлена, та виділяється яскраво-зеленим кольором.

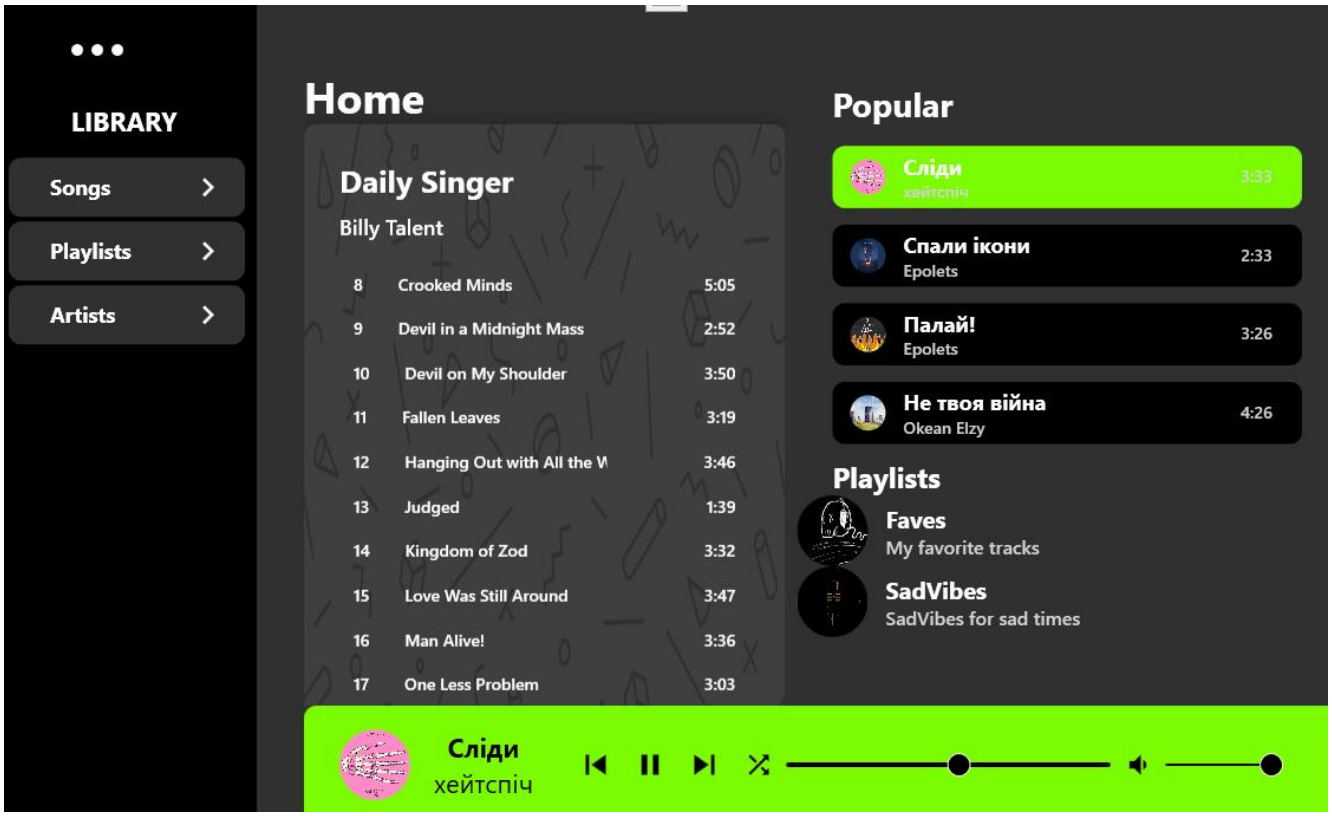

Рисунок 3.9 – Головна сторінка музичного сервісу

Наступним кроком було протестовано вікно перегляду списку пісень. Воно також працює коректно, дозволяючи прокручувати його для повного виводу вмісту бази даних пісень (рисунок 3.10).

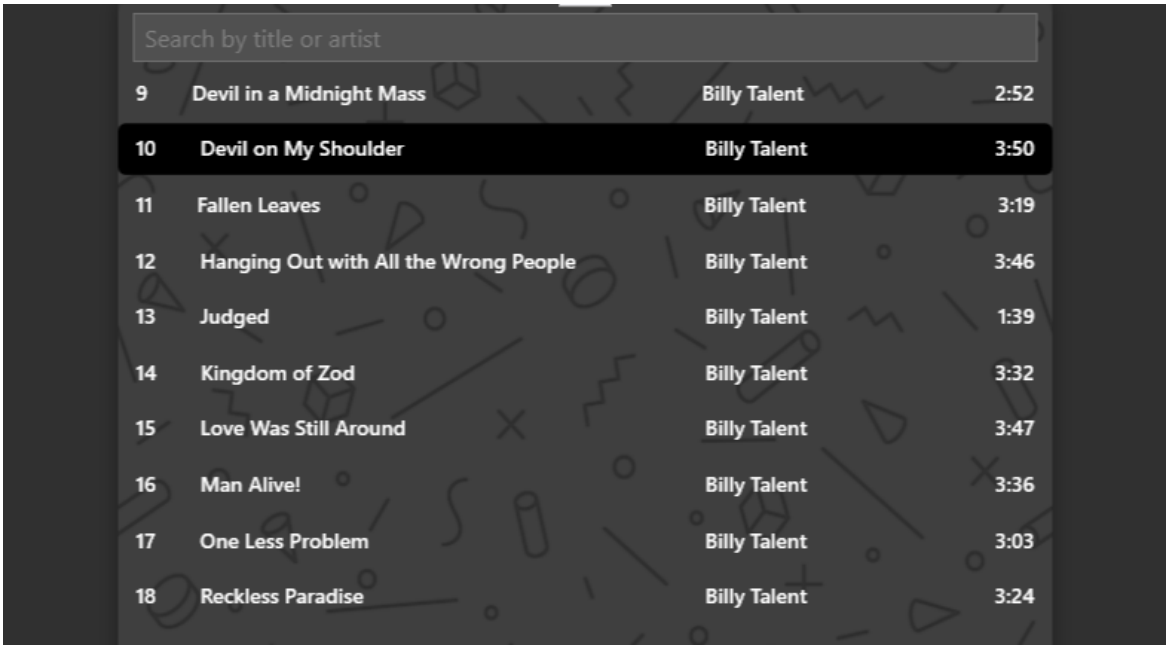

Рисунок 3.10 – Сторінка перегляду списку пісень

Наступним було перевірено на коректність роботу пошуку пісень за назвою або виконавцем (рисунок 3.11). Ця функція також працює відповідно до вимог.

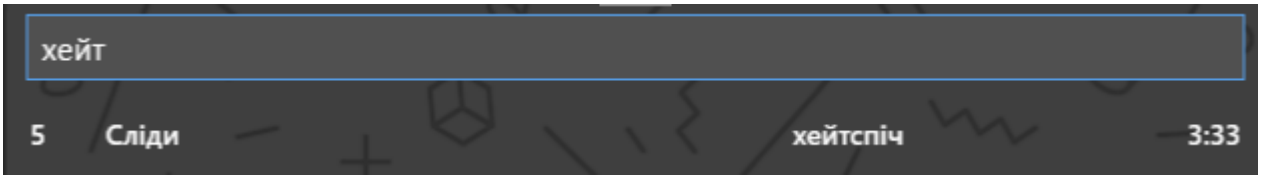

Рисунок 3.11 – Виконання пошуку

Після цього було проведено тестування вікна плейлистів для того, щоб оцінити правильність роботи функцій редагування самих плейлистів, та їх вмісту. Для цього спершу було створено новий плейлист, зображений на рисунку 3.12.

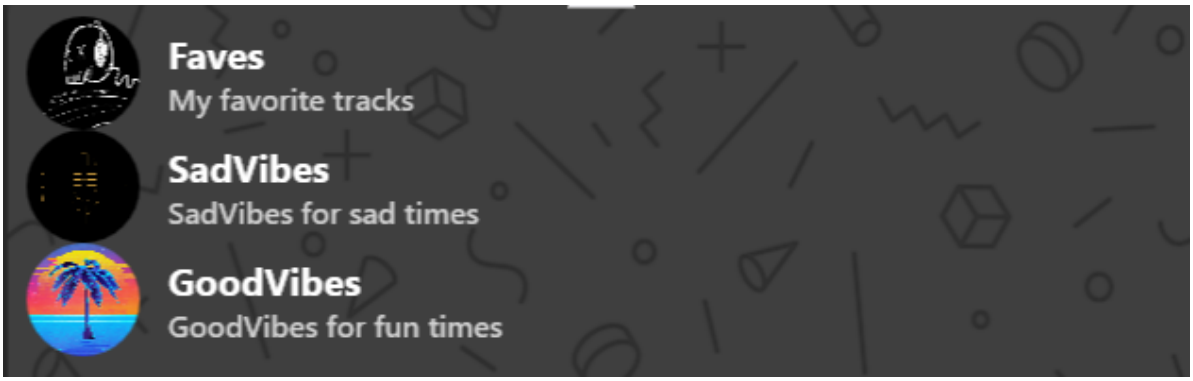

Рисунок 3.12 – Вивід оновленого списку плейлистів

Наступним кроком було перевірено вміст цього плейлиста, який наведений на рисунку 3.13.

Рисунок 3.13 – Вміст плейлиста

Після цього було перевірено редагування вмісту цього плейлиста, а саме видалення та додавання до нього пісень (рисунок 3.14, рисунок 3.15).

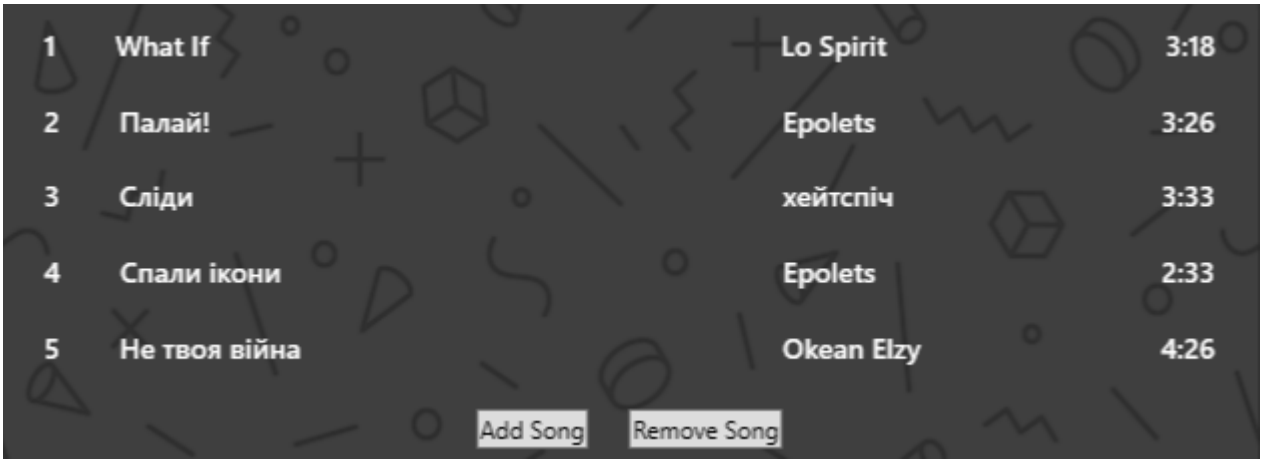

Рисунок 3.14 – Результат успішного додавання пісні до плейлиста

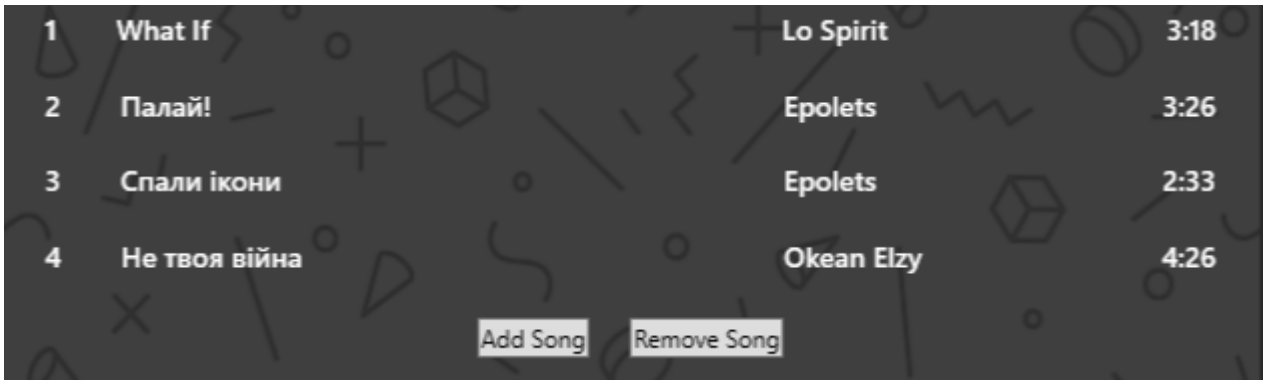

Рисунок 3.15 – Результат успішного видалення пісні з плейлиста

Останньою перевіркою коректності роботи плейлистів була перевірка видалення існуючого плейлиста. Результат виконання видалення наведено на рисунку 3.16.

Рисунок 3.16 – Результат видалення існуючого плейлиста

Окрім коректності роботи вікна перегляду плейлистів, також було перевірено роботу вікна перегляду виконавців та їх пошуку (рисунок 3.17).

Рисунок 3.17 – Результат пошуку виконавців

Останньою була проведена перевірка коректності роботи музичного плеєру, кнопок управління (пропустити вперед, назад, пауза/початок та перемішування). Результат наведено на рисунку 3.18.

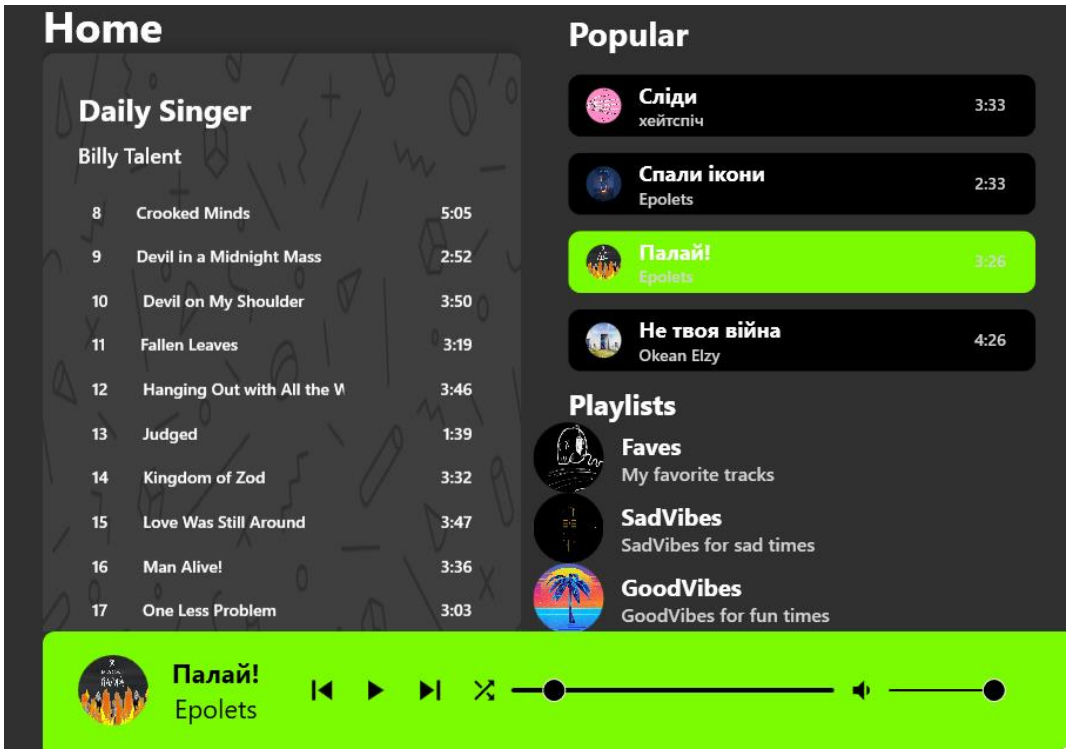

Рисунок 3.18 – Результат користування кнопками управління плеєра

Після завершення розробки та тестування додатку, його роботу можна оцінити як стабільну, при використанні не виникає помилок. Він також є продуктивним та надійним, справно працює при великій кількості пісень та іншої інформації у базах даних.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1. Вимоги безпеки до робочих місць для користувачів ПК

Десктоп додаток цифрового музичного сервісу було розроблено з використанням персонального комп'ютера з урахуванням вимог щодо безпеки та захисту здоров'я під час роботи з екранними пристроями, згідно зі затвердженим наказом Міністерства соціальної політики України від 14.02.2018 №207 (НПАОП 0.00-7.15-18) [10]. Це виконано відповідно до Директиви 90/270/ЄЕС від 29 травня 1990 року про мінімальні вимоги безпеки та здоров'я при роботі з екранними пристроями [10], а також Державними санітарними правилами і нормами роботи з візуальними дисплейними терміналами електронно-обчислювальних машин (ДСанПіН 3.3.2.007-98), затвердженими постановою Головного державного санітарного лікаря України від 01.12.1998 №7 [11].

Для забезпечення безпеки та захисту від випромінювання екранних пристроїв під час роботи з персональним комп'ютером, досягнуто гранично-допустимого рівня випромінювання, шуму, вібрації та температури. Це допоможе запобігти виникненню соматичних розладів та інших патологічних змін у стані здоров'я і працездатності, відповідно до вимог безпеки та охорони здоров'я[12].

У приміщенні робочого місця, відповідно до вимог Санітарних норм мікроклімату виробничих приміщень ДСанПіН 3.3.6.042-99 затверджених постановою Головного державного санітарного лікаря України від 01 грудня 1999 року №42 [13], повинні бути забезпечені оптимальні значення параметрів мікроклімату, таких як температура, відносна вологість і рухливість повітря.

При організації робочого місця для роботи з персональним комп'ютером необхідно забезпечити відповідність всіх елементів обладнання, а також їх ергономічного розташування та характеру роботи при розробці програмного забезпечення. Освітлення повинно відповідати вимогам ДСанПіН 3.3.2.007-98 [11] та створювати контраст між навколишнім середовищем та екраном.

При облаштуванні робочого місця необхідно використовувати таке обладнання, яке не випромінюватиме понаднормове тепло та не створюватиме додатковий шум. Гранично допустимий рівень шуму повинен відповідати вимогам Санітарних норм виробничого шуму, ультразвуку та інфразвуку ДСН 3.3.6.037-99 (01.12.1999), затверджених постановою Головного державного санітарного лікаря України від 01 грудня 1999 року №37 [14].

Таким чином, персональний комп’ютер та робоче місце, при проєктуванні та розробці десктоп додатку цифрового музичного сервісу є безпечним з точки зору охорони праці.

4.2. Надзвичайні ситуації, викликані пожежами, вибухами, техногенними та природніми причинами

За характером походження надзвичайні ситуації бувають [15]:

- природного характеру (стихійні лиха, захворюваність людей, заразні хвороби тварин та рослин тощо);
- техногенного характеру (транспортні аварії, пожежі, неспровоковані вибухи, аварії з викидом небезпечних хімічних і радіоактивних речовин тощо);
- соціальні небезпеки (злочинність, бродяжництво, алкоголізм, тютюнопаління та інші);
- воєнні небезпеки (наслідки застосування зброї масового ураження або звичайних засобів ураження).

Оскільки все більше використовуються великі енергетичні потужності, енергія яких сконцентрована на невеликих територіях, зокрема в містах та інших населених пунктах. Це спричиняє виникнення небезпек техногенного характеру. Цей процес супроводжується значним забрудненням навколишнього середовища,

знищенням лісів, опустелюванням, а також збільшенням кількості жертв серед людей через аварії на виробництві та транспорті.

Внаслідок порушень у експлуатації технічних об'єктів, аварії почали набувати катастрофічного масштабу ще у 20-30-х роках XX століття. Вплив таких аварій іноді виходить за межі держави й охоплює великі регіони. Негативний вплив на екологію, спричинений цими аваріями, може тривати від декількох днів до багатьох років. Ліквідація наслідків таких аварій потребує значних фінансових затрат та залучення багатьох спеціалістів. Види аварій, які зустрічаються найчастіше:

- аварії з витоком сильнодіючих отруйних речовин (аміаку, хлору, сірчаної та азотної кислот, чадного газу, сірчаного газу та інших речовин);
- аварії з викидом радіоактивних речовин у навколишнє середовище;
- пожежі та вибухи;
- аварії на транспорті та інші.

Найчастіше вибухи та пожежі відбуваються на об'єктах, де виробляються вибухонебезпечні та хімічні речовини [15]. Під час горіння багатьох матеріалів утворюються високотоксичні речовини, які часто становлять більшу небезпеку для життя людей, ніж саме полум'я. Останнім часом вогонь поширюється на багато штучних матеріалів, таких як полістирол, поліуретан, вініл, нейлон, поролон. Це призводить до викидів в повітря отруйних речовин, таких як синильна, соляна та мурашкова кислоти, метанол, формальдегід тощо.

Найбільш вибухонебезпечні суміші з повітрям утворюються при витоку газоподібних та зріджених вуглеводнів, таких як метан, пропан, бутан, етилен, пропилен тощо.

За останні десять років від третини до половини всіх аварій на виробництві пов'язані з вибухами технологічних систем та обладнання, таких як реактори, ємності, трубопроводи тощо. Пожежі на підприємствах можуть виникати також через ушкодження електропроводки та машин, що перебувають під напругою, а також через системи опалення.

У зв’язку з повномасштабним вторгненням, розпочатим Росією 24.02.2022 в Україну, різко виріс ризик виникнення пожеж та вибухів через обстріли, а також рівень людських та матеріальних втрат. Для того щоб мінімізувати втрати та небезпеку, необхідно дотримуватись правил безпеки. У випадку коли сили протиповітряної оборони помічають рух ворожих літаків, ракет чи інших повітряних цілей у напрямку міста, оголошується повітряна тривога. Почувши сигнал тривоги, перебуваючи вдома необхідно [16]:

- за можливості попередити сусідів, самотніх та літніх людей які проживають неподалік;
- швидко одягнутися;
- закрити вікна, вимкнути усі електроприлади, перекрити газ, вимкнути світло;
- взяти «тривожну валізу» (запас води та їжі тривалого зберігання, особисті документи, аптечку, кишеньковий ліхтар, тощо);
- найкоротшим шляхом прямувати до найближчого укриття чи захисної споруди;

У разі відсутності укриття неподалік, слід використовувати підвальне, або будь-яке заглиблене чи напівзаглиблене приміщення, а якщо можливості швидко перейти до укриття немає – переміститися у більш безпечне місце у домі (подалі від вікон, у коридор – під несучі стіни, тощо), користуватися правилом «двох стін».

У випадку оголошення повітряної тривоги під час перебування на вулиці необхідно прямувати до найближчого укриття як-от підземного переходу чи метро, або, у разі їх відсутності, до найближчого будинку, де слід сховатися на першому або цокольному поверсі. Якщо будівель поблизу немає, але наявна лісосмуга, слід сховатися між дерев. Варто уникати будівель стратегічного значення на кшталт електростанцій, держустанов, заводів тощо, та не користуватися ліфтами під час повітряної тривоги [17].

Стихійні лиха – це природні явища, які призводять до порушення нормальної діяльності населення, загибелі людей, руйнування і знищення матеріальних цінностей [18].

За причиною їх виникнення стихійні лиха поділяють на:

- тектонічні (процеси, які відбуваються в надрах землі), – землетруси, виверження вулканів;
- топологічні (пов’язані з процесами, які відбуваються на поверхні землі), – повені, зсуви, селі;
- метеорологічні (процеси, які відбуваються в атмосфері), – спека, урагани, посуха та інші.

Під час великих вивержень вулканів лавовий потік може розповсюджуватися до 30 км, а іноді досягати навіть 100 км. Для запобігання негативним наслідкам лавового потоку використовується метод його відведення від населених пунктів шляхом створення штучного русла. Також може застосовуватись будівництво дамб або охолодження лавових потоків водою.

Землетруси є сильними коливаннями земної кори, спричиненими тектонічним рухом плит, які можуть призводити до руйнування споруд, пожеж та втрати людських життів. Щорічно фіксується приблизно один мільйон сейсмічних та мікросейсмічних коливань, приблизно 100 тисяч з них відчуваються людьми, а 1000 завдають значних збитків, як наприклад землетрус, що стався 06.02.2023 у Туреччині та Сирії. Ознаки наближення землетрусу можуть включати запах газу там, де раніше його не було помічено, а також тривогу серед птахів та домашніх тварин.

Повінь - це значне затоплення місцевості, спричинене підйомом рівня води в річці, озері, водосховищі або внаслідок зливів, весняного танення снігу, вітрового нагону води, руйнування дамб або гребель.

Зсуви - це зміщення мас гірських порід вниз по схилу, які виникають через порушення рівноваги та ослаблення міцності гірських порід внаслідок вивітрювання, вимивання опадами та підземними водами, систематичних поштовхів або нерозважливої господарської діяльності людини. Зсуви можуть відбуватися на схилах з нахилом понад 20 градусів в будь-яку пору року.

Селі – це природні катастрофи, які відбуваються в басейнах невеликих гірських річок та спричиняють затоплення з великою концентрацією ґрунту,

мінеральних часток, каміння та уламків гірських порід. Вони виникають через зливи, інтенсивне танення снігу, прориви завальних озер, обвали, зсуви та землетруси.

Шторми - це вітри, що супроводжуються значними поривами та атмосферними змінами. Шторми можуть бути різного типу, такі як піщані, сніжні або грозові бурі. Вони можуть мати широкий спектр наслідків, включаючи погіршення видимості, падіння дерев та пошкодження будівель.

Урагани - це сильні циклонічний вітер, які супроводжуються сильними дощами та грозами. Вони можуть мати значну шкідливу силу, здатну зруйнувати будинки, затопити райони та спричинити значні руйнування. Урагани класифікуються за шкалою Саффір-Сімпсона залежно від швидкості вітру та потенційного руйнування.

Лісові пожежі розрізняються за інтенсивністю горіння [19]. Низові пожежі характеризуються горінням трав'яного покриву, підстилки та підліску, але не захоплюють крони дерев. Швидкість руху фронту низової пожежі може бути від 0,3 до 16 м/хв, а висота полум'я зазвичай становить 1-2 м. Верхові пожежі характеризуються горінням крон дерев і можуть поширюватися зі швидкістю від 8 до 25 км/год. Підземні пожежі розповсюджуються під шаром торфу та мають повільне горіння зі швидкістю від 0,1 до 0,5 м/хв. Степові пожежі виникають на відкритій місцевості з пожухлою травою або хлібом, і їх швидкість поширення може досягати 20-30 км/год.

ВИСНОВКИ

У даній кваліфікаційній роботі було розроблено десктоп-додаток цифрового музичного сервісу, використовуючи технологію Windows Presentation Foundation та гнучкі архітектурні моделі.

У першому розділі було проведено аналіз предметної області. Виконано огляд конкурентів та проведено аналіз ринку стримінгових сервісів. Обґрунтовано вибраний напрям дослідження, для розробки програмного продукту вибрано середовище Visual Studio через його зручність та повну інтегрованість з об’єктно-орієнтованою мовою програмування C#. Окрім цього, для створення сучасного користувацького інтерфейсу використано технологію Windows Presentation Foundation.

У другому розділі проведено проєктування системи, розроблено модель предметної області цифрового музичного сервісу. Також, було створено бізнес модель додатку, і проведено проєктування архітектури системи.

У третьому розділі виконано конструювання, спроєктованої системи. З цією метою було реалізовано ключові класи розробленої архітектури, розроблено і впроваджено усі визначені вимогами функції. Окрім цього, було розроблено сучасний та привабливий графічний інтерфейс, котрий залучить більшу кількість користувачів. Після завершення розробки додатку, було успішно проведено його тестування та контроль якості.

У четвертому розділі описано вимоги безпеки до робочих місць для користувачів ПК та надзвичайні ситуації, викликані пожежами, вибухами, техногенними та природніми причинами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Фінансовий звіт Spotify. [Електронний ресурс] – Режим доступу до ресурсу: https://investors.spotify.com/financials/default.aspx#quarterly-results.

2. Фінансовий звіт Apple. [Електронний ресурс] – Режим доступу до ресурсу: https://investor.apple.com/investor-relations/default.aspx

3. Фінансовий звіт Alphabet. [Електронний ресурс] – Режим доступу до ресурсу: https://abc.xyz/investor/

4. Фінансовий звіт Deezer. [Електронний ресурс] – Режим доступу до ресурсу: https://www.deezer-investors.com/fin/

5. Документація C#. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/

6. Документація WPF. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-7.0

7. Event-Driven Architecture / R. Bruns, J. Dunkel. – Heidelberg: Springer Berlin, 2010. – 241

8. Документація IBM RSA. [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=designer-rational-software-architect-product-overview

9. Документація стандартного класу C# Window. [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/api/system.windows.window?view=windowsdesktop-7.0

10. Наказ «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями». [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508-18#Text

11. ДСанПіН 3.3.2.007-98 «Норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text

12. Грибан В. Г., Негодченко О. В. Охорона праці : навч. посіб. Київ : Центр учбової літератури, 2009, 280 с.

13. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень» [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99#Text

14. ДСН 3.3.6.037-99 «Санітарні норми виробничого шуму, ультразвуку та інфразвуку» [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va037282-99#Text

15. Безпека життєдіяльності : навч. посіб. / Скобло Ю. С. та ін. Київ : Кондор, 2003, 424 с.

16. У разі надзвичайної ситуації або війни. // Центр стратегічних комунікацій та інформаційної безпеки МКІП. – 2021. – С. 14.

17. Як вижити під час надзвичайної ситуації та під час війни: легко про серйозне / Н.Нечаєва-Юрійчук, О. Танасійчук, О. Ноздрачов, С. Дмитрієв. – Київ: Упор, 2021. – 31 с.

18. Кодекс цивільного захисту України [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/5403-17#Text

19. Наказ «Про затвердження Правил пожежної безпеки в лісах України» [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0328-05#Text

ДОДАТКИ

ДОДАТОК А

Лістинг коду розробленого додатку

Лістинг 1 – файл MainWindow.xaml.cs

```
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using MusicPlayer.UserControls;
using MusicPlayer.Data;
using MusicPlayer.Views;
using System;
using MahApps.Metro.IconPacks;
using System.Linq;
using System.Windows.Threading;

namespace MusicPlayer
{
    public partial class MainWindow : Window
    {
        private SongDataAccess songDataAccess;
        private PlaylistDataAccess playlistDataAccess;
        private MediaPlayer mediaPlayer;
        private List<SongItem> songs;
        private Queue<SongItem> songQueue;
        private Stack<SongItem> songHistory;
        private bool isPlaying = false;
        private bool isDraggingSlider = false;
        private DispatcherTimer sliderTimer;

        public MainWindow()
        {
            InitializeComponent();

            string connectionString = @"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\riser\Documents\Mus
icPlayer.mdf;Integrated Security=True;Connect Timeout=30";
            songDataAccess = new SongDataAccess(connectionString);
            playlistDataAccess = new PlaylistDataAccess(connectionString);
            mediaPlayer = new MediaPlayer();
            songs = new List<SongItem>();
            songQueue = new Queue<SongItem>();
            songHistory = new Stack<SongItem>();
        }
        private void Border_MouseDown(object sender, MouseButtonEventArgs
e)
        {
            if (e.ChangedButton == MouseButton.Left)
                this.DragMove();
        }
        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            LoadSongs();
            LoadDailySongs();
```

```
            UpdatePlayerWithActiveSong();
            SetPopularSongsIsActive();
            OutputPlaylists();
        }
        private void LoadSongs()
        {
            songs = songDataAccess.GetSongs();
            /*songItemsContainer.Items.Clear();
            songItemsContainer.ItemsSource = songs;*/

            foreach (SongItem song in songs)
            {
                if (song.IsActive)
                {
                    // Play the active song
                    PlaySong(song);
                }
                else
                {
                    // Add the non-active songs to the song queue
                    songQueue.Enqueue(song);
                }
            }
        }
        private void LoadDailySongs()
        {
            List<SongItem> dailySongs = songs.Where(song => song.Artist ==
"Billy Talent").ToList();

            // Clear the existing song items in the container
            songItemsContainer.Items.Clear();

            foreach (SongItem song in dailySongs)
            {
                // Create a new instance of the SongItem UserControl
                SongItem songItem = new SongItem
                {
                    Number = song.Number,
                    Title = song.Title,
                    Time = song.Time
                };

                // Add the songItem to the songItemsContainer
                songItemsContainer.Items.Add(songItem);
            }
        }

        private void PlaySong(SongItem song)
        {
            // Set the source of the media player to the selected song
            mediaPlayer.Open(new Uri(song.FilePath,
UriKind.RelativeOrAbsolute));

            // Update the player UI with the selected song details
            player.Children.Clear();
```

```
            player.Children.Add(new TextBlock { Text = song.Title,
Foreground = Brushes.White, FontSize = 22, FontWeight = FontWeights.Bold
});
            player.Children.Add(new TextBlock { Text = song.Artist,
Foreground = Brushes.White, FontSize = 22 });
        }
        private void PlayButton_Click(object sender, RoutedEventArgs e)
        {
            if (isPlaying)
            {
                // Pause the song
                mediaPlayer.Pause();
                isPlaying = false;
                StopSliderTimer();
                playButtonIcon.Kind = PackIconMaterialKind.Play;
            }
            else
            {
                // Start playing the song
                mediaPlayer.Play();
                isPlaying = true;
                StartSliderTimer();
                playButtonIcon.Kind = PackIconMaterialKind.Pause;
            }
        }
        private void SkipNextButton_Click(object sender, RoutedEventArgs e)
        {
            // Check if there are songs in the queue
            if (songQueue.Count > 0)
            {
                // Get the next song from the queue
                SongItem nextSong = songQueue.Dequeue();

                // Set the next song as active in the database
                nextSong.IsActive = true;
                songDataAccess.SetActiveSong(nextSong.Number);


                // Play the next song
                mediaPlayer.Open(new Uri(nextSong.FilePath));
                mediaPlayer.Play();
                StartSliderTimer();

                // Update the UI with the new song
                UpdatePlayerWithActiveSong();
                SetPopularSongsIsActive();

                // Add the skipped song back to the end of the queue for
infinite looping
                songQueue.Enqueue(nextSong);

                // Add the current song to the history
                songHistory.Push(nextSong);
            }
        }
        private void SkipPreviousButton_Click(object sender,
RoutedEventArgs e)
```

```
        {
            // Check if there are songs in the history
            if (songHistory.Count > 1)
            {
                // Get the previous song from the history
                songHistory.Pop(); // Pop the current song
                SongItem previousSong = songHistory.Peek(); // Get the
previous song

                // Set the previous song as active in the database
                previousSong.IsActive = true;
                songDataAccess.SetActiveSong(previousSong.Number);

                // Play the previous song
                mediaPlayer.Open(new Uri(previousSong.FilePath));
                mediaPlayer.Play();
                StartSliderTimer();

                // Update the UI with the new song
                UpdatePlayerWithActiveSong();
                SetPopularSongsIsActive();

                // Add the previous song back to the queue for infinite
looping
                songQueue.Enqueue(previousSong);
            }
        }
        private void ShuffleButton_Click(object sender, RoutedEventArgs e)
        {
            // Convert the song queue to a list
            List<SongItem> songList = songQueue.ToList();

            // Create a random number generator
            Random random = new Random();

            // Shuffle the song list using Fisher-Yates algorithm
            for (int i = songList.Count - 1; i > 0; i--)
            {
                int j = random.Next(i + 1);
                SongItem temp = songList[i];
                songList[i] = songList[j];
                songList[j] = temp;
            }

            // Clear the song queue
            songQueue.Clear();

            // Enqueue the shuffled songs back into the song queue
            foreach (SongItem song in songList)
            {
                songQueue.Enqueue(song);
            }
        }

        private void SongButton_Click(object sender, RoutedEventArgs e)
        {
```

```
            SongDataAccess songDataAccess = new SongDataAccess(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\riser\Documents\Mus
icPlayer.mdf;Integrated Security=True;Connect Timeout=30");

            List<SongItem> songs = songDataAccess.GetSongs();

            ListBox songListBox = new ListBox();
            songListBox.ItemsSource = songs;

            // Create an instance of the SongListWindow
            SongListWindow songListWindow = new SongListWindow(songs);

            // Show the SongListWindow
            songListWindow.Show();
        }
        private void ArtistButton_Click(object sender, RoutedEventArgs e)
        {
            SongDataAccess songDataAccess = new SongDataAccess(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\riser\Documents\Mus
icPlayer.mdf;Integrated Security=True;Connect Timeout=30");

            List<string> artists = songDataAccess.GetArtists();

            // Create an instance of the ArtistListWindow and pass the
artists
            ArtistListWindow artistListWindow = new
ArtistListWindow(artists);

            // Show the ArtistListWindow
            artistListWindow.Show();
        }
        private void UpdatePlayerWithActiveSong()
        {
            SongItem activeSong = songDataAccess.GetActiveSong();

            if (activeSong != null)
            {
                player.Children.Clear();

                // Create a Grid panel to hold the player UI elements
                Grid grid = new Grid();

                // Define fixed row and column definitions
                grid.RowDefinitions.Add(new RowDefinition { Height = new
GridLength(1, GridUnitType.Star) });
                grid.RowDefinitions.Add(new RowDefinition { Height = new
GridLength(1, GridUnitType.Star) });

                // Add the song title to the first row
                TextBlock titleTextBlock = new TextBlock
                {
                    Text = activeSong.Title,
                    Foreground = Brushes.Black,
                    FontSize = 22,
                    FontWeight = FontWeights.Bold,
                    HorizontalAlignment = HorizontalAlignment.Center
                };
```

```
                Grid.SetRow(titleTextBlock, 0);
                grid.Children.Add(titleTextBlock);

                // Add the artist name to the second row
                TextBlock artistTextBlock = new TextBlock
                {
                    Text = activeSong.Artist,
                    Foreground = Brushes.Black,
                    FontSize = 22,
                    HorizontalAlignment = HorizontalAlignment.Center
                };
                Grid.SetRow(artistTextBlock, 1);
                grid.Children.Add(artistTextBlock);

                // Add the Grid panel to the player's Children collection
                player.Children.Add(grid);

                // Update the image source of the ellipse
                ImageBrush imageBrush = new ImageBrush();
                imageBrush.ImageSource = new BitmapImage(new
Uri(activeSong.ImagePath, UriKind.RelativeOrAbsolute));
                imageEllipse.Fill = imageBrush;
            }
        }
        private void SetPopularSongsIsActive()
        {
            List<SongItem> songs = songDataAccess.GetSongs();
            SongItem activeSong = songDataAccess.GetActiveSong();

            foreach (UIElement child in popularSection.Children)
            {
                if (child is PopularSong popularSong)
                {
                    string popularSongTitle = popularSong.Title;

                    if (activeSong != null &&
popularSongTitle.Equals(activeSong.Title))
                    {
                        popularSong.IsActive = true;
                    }
                    else
                    {
                        popularSong.IsActive = false;
                    }
                }
            }
        }
        private void PlaylistButton_Click(object sender, RoutedEventArgs e)
        {
            PlaylistDataAccess playlistDataAccess = new
PlaylistDataAccess(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\riser\Documents\Mus
icPlayer.mdf;Integrated Security=True;Connect Timeout=30");
            SongDataAccess songDataAccess = new SongDataAccess(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\riser\Documents\Mus
icPlayer.mdf;Integrated Security=True;Connect Timeout=30");
```

```
            // Retrieve the playlists from the database
            List<Playlist> playlists = playlistDataAccess.GetPlaylists();

            // Create an instance of the PlaylistWindow and pass the
playlists, playlistDataAccess, and songDataAccess
            PlaylistWindow playlistWindow = new PlaylistWindow(playlists,
playlistDataAccess, songDataAccess);

            // Show the PlaylistWindow
            playlistWindow.Show();
        }
        private void OutputPlaylists()
        {
            List<Playlist> playlists = playlistDataAccess.GetPlaylists();

            foreach (Playlist playlist in playlists)
            {
                // Create a new instance of the Playlist UserControl
                Playlist playlistControl = new Playlist
                {
                    Id = playlist.Id,
                    Title = playlist.Title,
                    Desc = playlist.Desc,
                    ImagePath = playlist.ImagePath
                };

                // Add the playlistControl to the playlistItemsContainer
                playlistSection.Children.Add(playlistControl);
            }
        }
        private void UpdateSliderPosition()
        {
            if (mediaPlayer.NaturalDuration.HasTimeSpan)
            {
                TimeSpan duration = mediaPlayer.NaturalDuration.TimeSpan;
                TimeSpan currentPosition = mediaPlayer.Position;

                // Calculate the progress percentage
                double progress = currentPosition.TotalSeconds /
duration.TotalSeconds;

                // Set the slider value based on the progress
                slider.Value = progress * slider.Maximum;
            }
        }
        private void slider_DragStarted(object sender, RoutedEventArgs e)
        {
            // Set the dragging flag to true when the slider dragging
starts
            isDraggingSlider = true;
        }
        private void slider_DragCompleted(object sender, RoutedEventArgs e)
        {
            // Set the dragging flag to false when the slider dragging
completes
            isDraggingSlider = false;
```

```
            // Calculate the new position based on the slider value
            double progress = slider.Value / slider.Maximum;
            TimeSpan newPosition = TimeSpan.FromSeconds(progress *
mediaPlayer.NaturalDuration.TimeSpan.TotalSeconds);

            // Seek the media player to the new position
            mediaPlayer.Position = newPosition;
        }

        private void StartSliderTimer()
        {
            // Create a new DispatcherTimer
            sliderTimer = new DispatcherTimer();
            sliderTimer.Interval = TimeSpan.FromSeconds(1);

            // Set the event handler for the Tick event
            sliderTimer.Tick += SliderTimer_Tick;

            // Start the timer
            sliderTimer.Start();
        }
        private void StopSliderTimer()
        {
            // Stop the timer and clean up
            sliderTimer.Stop();
            sliderTimer.Tick -= SliderTimer_Tick;
            sliderTimer = null;
        }
        private void SliderTimer_Tick(object sender, EventArgs e)
        {
            // Update the slider position
            UpdateSliderPosition();
        }
        private void volumeSlider_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            double volume = volumeSlider.Value;

            // Change the volume based on slider value
            mediaPlayer.Volume = volume / 100;
        }
    }
}
```

Лістинг 2 – файл PlaylistWindow.xaml.cs

```
using System.Collections.Generic;
using System.Windows;
using System.Windows.Controls;
using MusicPlayer.Data;
using MusicPlayer.UserControls;

namespace MusicPlayer.Views
{
    public partial class PlaylistWindow : Window
    {
        private PlaylistDataAccess playlistDataAccess;
```

```
        private SongDataAccess songDataAccess;
        private List<Playlist> playlists;
        private string selectedPlaylistName;
        public PlaylistWindow(List<int> playlistIds, PlaylistDataAccess
dataAccess, SongDataAccess songDataAccess)
        {
            InitializeComponent();
            playlistDataAccess = dataAccess;
            this.songDataAccess = songDataAccess;
            // Create Playlist UserControls based on playlistIds and add
them to the StackPanel
            foreach (int playlistId in playlistIds)
            {
                // Retrieve playlist details using playlistDataAccess and
create Playlist UserControls
                List<int> songNumbers =
playlistDataAccess.GetPlaylistSongs(playlistId);
                Playlist playlist = new Playlist
                {
                    Title = $"Playlist {playlistId}",
                    Desc = $"Description {playlistId}",
                    IsActive = false
                };
                playlistStackPanel.Children.Add(playlist);
            }
        }
        public PlaylistWindow(List<Playlist> playlists, PlaylistDataAccess
dataAccess, SongDataAccess songDataAccess)
        {
            InitializeComponent();
            this.playlists = playlists;
            playlistDataAccess = dataAccess;
            this.songDataAccess = songDataAccess;
            // Call the LoadPlaylists method to populate the stack panel
            LoadPlaylists();
        }

        private void LoadPlaylists()
        {
            foreach (Playlist playlist in playlists)
            {
                // Create a Playlist UserControl for each playlist
                Playlist playlistControl = new Playlist
                {
                    Title = playlist.Title,
                    Desc = playlist.Desc,
                    ImagePath = playlist.ImagePath
                };

                // Add a click event handler for the playlist UserControl
                playlistControl.MouseDown += PlaylistControl_MouseDown;

                // Add the playlist UserControl to the stack panel
                playlistStackPanel.Children.Add(playlistControl);
            }
        }
```

```
        private void PlaylistControl_MouseDown(object sender,
System.Windows.Input.MouseButtonEventArgs e)
        {
            // Get the clicked playlist UserControl
            Playlist clickedPlaylist = (Playlist)sender;

            // Get the corresponding playlist from the list
            Playlist selectedPlaylist = playlists.Find(p => p.Title ==
clickedPlaylist.Title);

            if (selectedPlaylist != null)
            {
                // Store the selected playlist name
                selectedPlaylistName = selectedPlaylist.Title;
                // Open the song list window for the selected playlist
                OpenSongListWindow(selectedPlaylist.Title);
            }
        }
        private void OpenSongListWindow(string playlistName)
        {
            List<int> songNumbers =
playlistDataAccess.GetSongNumbersForPlaylist(playlistName);

            // Convert song numbers to SongItem objects
            List<SongItem> songs = ConvertToSongItems(songNumbers);

            // Clear the current stack panel contents
            playlistStackPanel.Children.Clear();
            playlistButtons.Children.Clear();

            // Create a grid to hold the song details and buttons
            Grid grid = new Grid();

            // Add the song details to the stack panel
            for (int i = 0; i < songs.Count; i++)
            {
                SongItem song = songs[i];

                // Create a SongUserControl to display the song details
                SongItem songItem = new SongItem();
                songItem.Number = i + 1;  // Set the song number starting
from 1
                songItem.Title = song.Title;
                songItem.Artist = song.Artist;
                songItem.Time = song.Time;

                // Add the SongUserControl to the stack panel
                playlistStackPanel.Children.Add(songItem);
            }
            // Add the buttons for editing the playlist
            Button addButton = new Button
            {
                Content = "Add Song",
                Margin = new Thickness(10),
                HorizontalAlignment = HorizontalAlignment.Left,
                VerticalAlignment = VerticalAlignment.Bottom
            };
```

```
            addButton.Click += AddButton_Click;

            Button removeButton = new Button
            {
                Content = "Remove Song",
                Margin = new Thickness(10),
                HorizontalAlignment = HorizontalAlignment.Right,
                VerticalAlignment = VerticalAlignment.Bottom
            };
            removeButton.Click += RemoveButton_Click;

            // Create a stack panel to hold the buttons
            StackPanel buttonPanel = new StackPanel
            {
                Orientation = Orientation.Horizontal,
                HorizontalAlignment = HorizontalAlignment.Center,
                VerticalAlignment = VerticalAlignment.Bottom
            };

            // Add the buttons to the stack panel
            buttonPanel.Children.Add(addButton);
            buttonPanel.Children.Add(removeButton);

            // Set the grid row for the button panel
            Grid.SetRow(buttonPanel, songs.Count);

            // Add the grid with song details and buttons to the main stack
panel
            playlistStackPanel.Children.Add(grid);
            playlistStackPanel.Children.Add(buttonPanel);
        }
        private void AddButton_Click(object sender, RoutedEventArgs e)
        {
            // Prompt the user for the song title
            string songTitle =
Microsoft.VisualBasic.Interaction.InputBox("Enter the song title:", "Add
Song");

            if (!string.IsNullOrEmpty(songTitle))
            {
                int playlistId =
GetPlaylistIdFromDatabase(selectedPlaylistName);

                // Call the method to add the song to the playlist
                playlistDataAccess.AddSongToPlaylist(playlistId,
songTitle);

                // Display a message after adding the song
                MessageBox.Show($"Song '{songTitle}' added to the playlist
successfully.", "Add Song", MessageBoxButton.OK,
MessageBoxImage.Information);
            }
            else
            {
                // Handle the case when the user cancels or provides an
empty song title
```

```
                MessageBox.Show("No song title entered. Song was not added
to the playlist.", "Add Song", MessageBoxButton.OK,
MessageBoxImage.Warning);
            }
            OpenSongListWindow(selectedPlaylistName);
        }
        private void RemoveButton_Click(object sender, RoutedEventArgs e)
        {
            // Prompt the user for the song title
            string songTitle =
Microsoft.VisualBasic.Interaction.InputBox("Enter the song title:", "Remove
Song");

            if (!string.IsNullOrEmpty(songTitle))
            {
                int playlistId =
GetPlaylistIdFromDatabase(selectedPlaylistName);

                // Call the method to remove the song from the playlist
                playlistDataAccess.RemoveSongFromPlaylist(playlistId,
songTitle);

                // Display a message after removing the song
                MessageBox.Show($"Song '{songTitle}' removed from the
playlist successfully.", "Remove Song", MessageBoxButton.OK,
MessageBoxImage.Information);
            }
            else
            {
                // Handle the case when the user cancels or provides an
empty song title
                MessageBox.Show("No song title entered. Song was not
removed from the playlist.", "Remove Song", MessageBoxButton.OK,
MessageBoxImage.Warning);
            }
            OpenSongListWindow(selectedPlaylistName);
        }
        private int GetPlaylistIdFromDatabase(string playlistName)
        {
            int playlistId =
playlistDataAccess.GetPlaylistIdByName(playlistName);

            return playlistId;
        }

        private List<SongItem> ConvertToSongItems(List<int> songNumbers)
        {
            List<SongItem> songs = new List<SongItem>();
            // Retrieve the songs that belong to the given song numbers
            List<SongItem> playlistSongs =
songDataAccess.GetSongsByNumbers(songNumbers);

            // Iterate over the song numbers and find the corresponding
song in the playlistSongs list
            foreach (int songNumber in songNumbers)
            {
```

```
                SongItem song = playlistSongs.Find(s => s.Number ==
songNumber);
                if (song != null)
                {
                    songs.Add(song);
                }
            }

            return songs;
        }

        private void CreateButton_Click(object sender, RoutedEventArgs e)
        {
            // Prompt the user for the playlist name
            string playlistName =
Microsoft.VisualBasic.Interaction.InputBox("Enter the playlist name:",
"Create Playlist");

            if (!string.IsNullOrEmpty(playlistName))
            {
                // Prompt the user for the playlist description and
imagePath
                string playlistDescription =
Microsoft.VisualBasic.Interaction.InputBox("Enter the playlist
description:", "Create Playlist");
                string playlistImage =
Microsoft.VisualBasic.Interaction.InputBox("Enter the playlist image
path:", "Create Playlist");

                // Get the last PlaylistId from the database
                int lastPlaylistId =
playlistDataAccess.GetLastPlaylistId(@"Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=C:\Users\riser\Documents\Mus
icPlayer.mdf;Integrated Security=True;Connect Timeout=30");

                // Calculate the new PlaylistId as the last PlaylistId + 1
                int newPlaylistId = lastPlaylistId + 1;

                // Call the method to create a new playlist with the new
PlaylistId
                playlistDataAccess.CreatePlaylist(newPlaylistId,
playlistName, playlistDescription, playlistImage);

                // Display a message after creating the playlist
                MessageBox.Show($"Playlist '{playlistName}' created
successfully.", "Create Playlist", MessageBoxButton.OK,
MessageBoxImage.Information);
            }
            else
            {
                // Handle the case when the user cancels or provides an
empty playlist name
                MessageBox.Show("No playlist name entered. Playlist was not
created.", "Create Playlist", MessageBoxButton.OK,
MessageBoxImage.Warning);
            }
            UpdatePlaylistDisplay();
```

```
        }
        private void DeleteButton_Click(object sender, RoutedEventArgs e)
        {
            // Prompt the user for the playlist name to delete
            string playlistName =
Microsoft.VisualBasic.Interaction.InputBox("Enter the playlist name to
delete:", "Delete Playlist");

            if (!string.IsNullOrEmpty(playlistName))
            {
                // Check if the playlist exists
                int playlistId =
playlistDataAccess.GetPlaylistIdByName(playlistName);
                if (playlistId != 0)
                {
                    // Confirm the deletion with the user
                    MessageBoxResult result = MessageBox.Show($"Are you
sure you want to delete the playlist '{playlistName}'?", "Confirm
Deletion", MessageBoxButton.YesNo, MessageBoxImage.Warning);
                    if (result == MessageBoxResult.Yes)
                    {
                        // Call the method to delete the playlist
                        playlistDataAccess.DeletePlaylist(playlistId);

                        // Display a message after deleting the playlist
                        MessageBox.Show($"Playlist '{playlistName}' deleted
successfully.", "Delete Playlist", MessageBoxButton.OK,
MessageBoxImage.Information);
                    }
                }
                else
                {
                    // Handle the case when the playlist does not exist
                    MessageBox.Show($"Playlist '{playlistName}' does not
exist.", "Delete Playlist", MessageBoxButton.OK, MessageBoxImage.Warning);
                }
            }
            else
            {
                // Handle the case when the user cancels or provides an
empty playlist name
                MessageBox.Show("No playlist name entered. Playlist was not
deleted.", "Delete Playlist", MessageBoxButton.OK,
MessageBoxImage.Warning);
            }
            UpdatePlaylistDisplay();
        }
        private void UpdatePlaylistDisplay()
        {
            // Clear the existing playlist display
            playlistStackPanel.Children.Clear();

            // Retrieve the updated list of playlists
            playlists = playlistDataAccess.GetPlaylists();

            // Populate the stack panel with the updated playlists
            foreach (Playlist playlist in playlists)
```

```
            {
                // Create a Playlist UserControl for each playlist
                Playlist playlistControl = new Playlist
                {
                    Title = playlist.Title,
                    Desc = playlist.Desc,
                    ImagePath = playlist.ImagePath
                };

                // Add a click event handler for the playlist UserControl
                playlistControl.MouseDown += PlaylistControl_MouseDown;

                // Add the playlist UserControl to the stack panel
                playlistStackPanel.Children.Add(playlistControl);
            }
        }
    }
}
```

ДОДАТОК Б

Диск з розробленою програмою