Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Мірошника Микити Вадимовича*
(ПІБ)

академічної групи *122-18-3*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка додатку «аудіо-плеєр» на мові програмування C#*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц.Приходченко С.Д.*			
розділів:				
спеціальний	*доц.Приходченко С.Д.*			
економічний	*доц.Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2022

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2022 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-18-3* (група) *Мірошника М.В.* (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка додатку «аудіо-плеєр» на мові програмування C#*

затверджена наказом ректора НТУ «ДП» від *18.05.2022 р.* № *268-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***15.06.2022 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.06.2022 р.***

Завдання видав _____ *доц.Приходченко С.Д.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Мірошник М.В.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2022 р.*

Термін подання кваліфікаційної роботи до ЕК: *16.06.2022 р.*

РЕФЕРАТ

Пояснювальна записка: 58 с., 28 рис., 3 дод., 25 джерела.

Об'єкт розробки: Аудіоплеєр на мові програмування C# з використанням фреймворку Bunifu та бібліотеки Tag Lib.

Мета кваліфікаційної роботи: написання додатку, який надає користувачу аудіо-плєер з можливістю програвання усіх найпопулярніших аудіоформатів, та комбінує в собі унікальний та практичний дизайн.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточняється постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформи для розробки, виконано проектування і розробка програми, описана робота програми, алгоритм і структура її функціювання, а також виклик та завантаження програми, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення полягає у створенні програмного додатка, що дозволяє зручно та комфортно слухати аудіофайли різних форматів у одному додатку.

Актуальність даного програмного додатку стоїть у тому, щоб створити мультиформатний аудіоплеєр з компактним дизайном, який спростить життя звичайному слухачу.

Список ключових слів: АУДІОПЛЕЄР, МУЛЬТИФОРМАТНИЙ, СПИСОК КОМП’ЮТЕР, АЛГОРИТМ, ІНТЕРФЕЙС, ДОДАТОК, VISUAL STUDIO.

ABSTRACT

Explanatory note: 58 p., 28 figs., 3 apps., 25 sources.

Object of development: Audio player coded in C # programming language using Bunifu framework and Tag Lib library.

The purpose of the qualification work: coding an application that provides the user with an audio player with the ability to play all the most popular audio formats and combines a unique and practical design.

The introduction examines the current problem statement, specifies the purpose of the qualification work and the area of its application, justifies the relevance of the topic and specifies the problem statement.

In the first section carries out the analysis of the subject area, determines the relevance of the task and the dedication of the development, creates task statement, the software and hardware requirements of the product, specifies technologies and tools for development.

In the second section analyzes the existing solutions, chose the platform for development, creates design and finish development of the product, describes the algorithm, structure and architecture solutions in the system, defines the input and output data, describes characteristics of the technical resources used, describes how to run a program, features of user interaction, differences between serve and client parts of product.

The economic section defines the complexity of the information system, calculates the cost of work on creating the application and defines the time for its creation.

The practical significance is creation a product which provides an opportunity to improve knowledge and increase knowledge.

The relevance of this software application is to create a multi-format audio player with a compact design that will simplify the life of the average listener.

Keywords: AUDIO PLAYER, MULTI-FORMAT, PLAYLIST, COMPUTER, INFORMATION SYSTEM, INTERFACE, APPLICATION, VISUAL STUDIO.

СПИСОК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення;

ПК – персоальний комп’ютер;

ОС – операційна система;

ІК – інтерфейс користувача;

ЦП – центральний просесор;

UI – User Interface;

UX – User Experience;

VS – Visual Studio;

WinForms – Windows Forms;

WPF – Windows Presentation Foundation;

API – Application Programming Interface;

GPU – Graphics processing unit;

GUI – Graphical user interface;

CLI – Command line interface;

ЗМІСТ

ВСТУП

Завдання даної кваліфікаційної роботи та об’єкт його діяльності безпосередньо пов’язані з напрямом підготовки та відповідає узагальненій тематиці кваліфікаційних робіт і переліку зазначених виробничих функцій, типових задач діяльності, умінню та компетенціям, якими повинні володіти бакалаври напряму 122 «Комп’ютерні науки».

Ще з початку часів музика, у тому чи іншому вигляді, акомпанувала кожному кроку розвитку людства. Від дня народження до дня смерті, від весілля до розлучення, в радості і й в тузі - була, і є музика.

З плином розвитку технологій, люди винаходили все більше способів записувати та зберігати музику. Спочатку це було на папері, потім на воску, ще пізніше - на платівках. Прихід цифрових технологій також приніс із собою новий та зручний спосіб зберігання музики - у файлі. Зберігання її такою, якою вона була зіграна. Більше не потрібно громіздких програвачів та довгих проводів, тим паче не потрібно збирати оркестр кожного разу, коли тобі хочеться щось послухати.

Сьогодні, комп’ютери та смартфони дозволяють бути нерозлучними з улюбленою музикою. Але, як і для усього, для програвання аудіофайлів потрібні спеціальні додатки - плеєри. Чи може засіб програвання музики вплинути на задоволення від неї? Може! Саме цьому дана кваліфікаційна робота націлена на проектування та реалізацію додатку-плеєра, який зможе поєднати в собі такі важливі аспекти як функціональність та зручність.

Метою кваліфікаційної роботи є вивчення засобів створення та роботи з декількома методами, а саме: підключення і робота з фреймворками та біблотеками у середовищі програмування C#; опрацювання механізму вводу/виводу файлів, що є основоположним принципом у багатьох сферах розробки; отримання та опрацювання метаданих з файлів формату “mp3”, “flac”, “wav”, “m4a” тощо.

Дана робота ознайомлює з принципами та засобами розробки додатків з застосуванням API Windows Forms.

Основними вимогами до додатку є: простий та інтуїтивний дизайн; стабільна, надійна робота; підтримка основних аудіоформатів; надання основної інформації про поточний файл.

РОЗДІЛ І АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

Медіа плеєр(медіапрогравач) – вид програми або додатка, який розроблений для роботи з відтворення файлів мультимедіа, а саме: аудіо- та відеофайли. Медіапрогравачі можуть бути як мультимедія системами, так і поділятись на більш спеціалізовані під один конкретний формат програми. Ті програми, які націлені на відтворення відеофайлів називаюся відеопрогравачі (відео плеєри), а програми, які відтворюють аудіофайли – аудіопрогравачі (аудіо плеєри). Дана кваліфікаційна робота зосереджена саме на останньому типі медіарогравачів.

Майже усі сучасні операційні системи, як десктопні, так і мобільні, мають вбудовані мультимедіа плеєри. Прикладами таких э Windows Media Player для ОС Windows та QuickTime, iTunes для ОС Mac. Також існує безліч сторонніх програм, які можуть включати в себе більш обширний набор функцій, або можуть бути спрямованими на якусь конкретну професіональну особливість. Прикладами найпопулярніших сторонніх додатків є: VLC, Media Player Classic, KMplayer, AIMP, SMPlayer тощо.

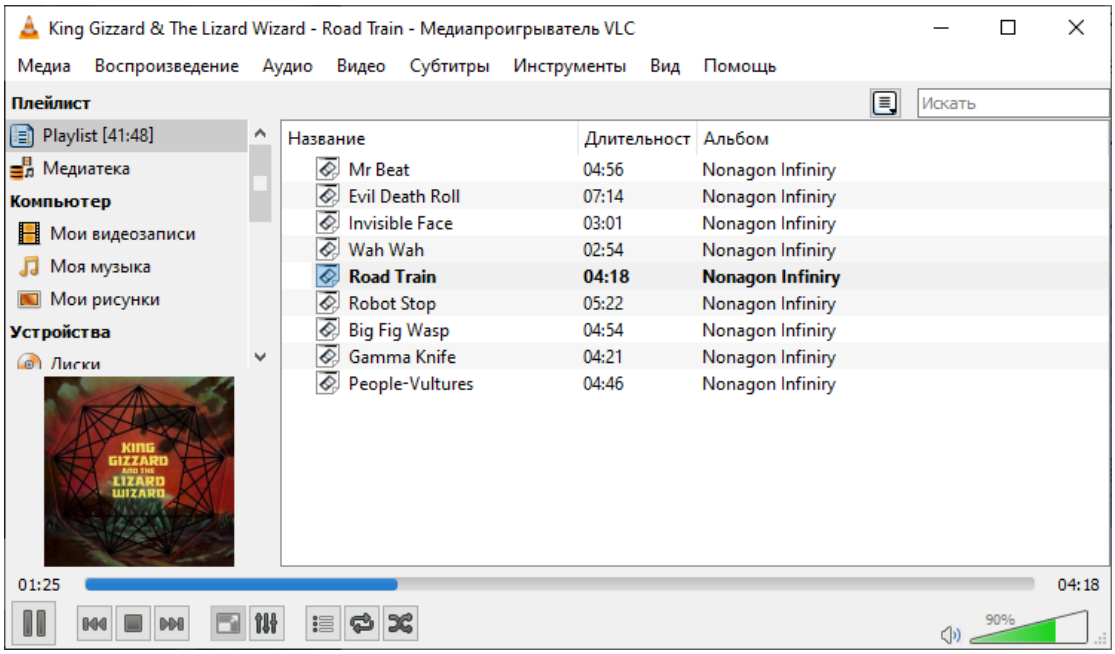

Рис. 1.1 Інтерфейс програми VLC Media Player для аудіоформату

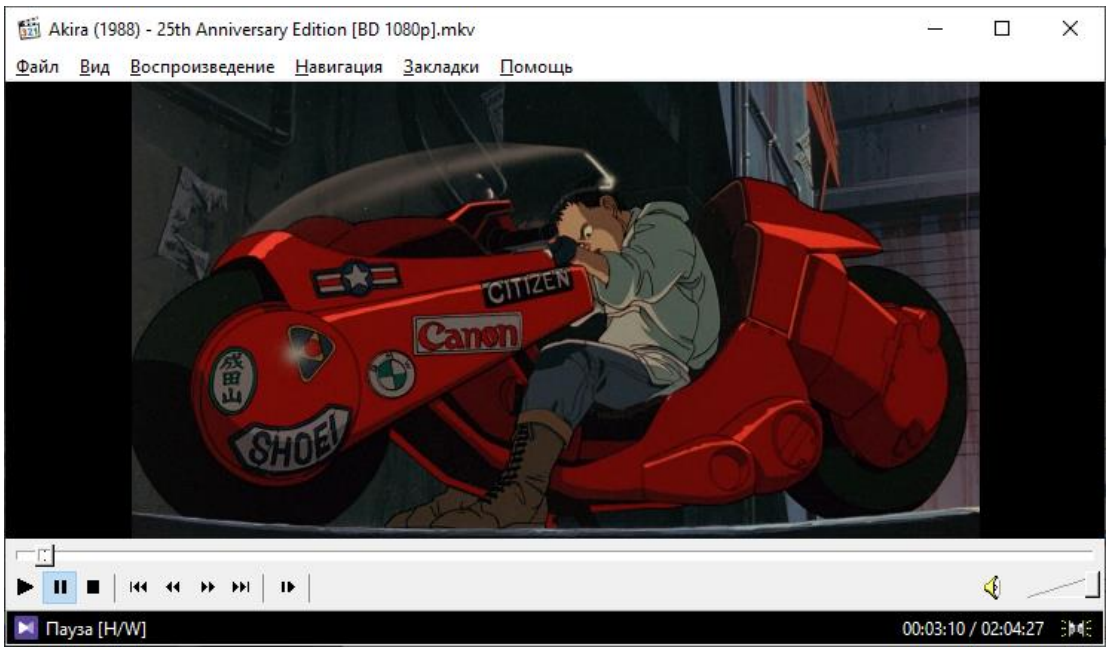

Рис. 1.2 Інтерфейс Classic Media Player для відеоформату

Якщо зосередити увагу на програмах, які спеціалізуються на програванні тільки аудіофайлів, то гарними прикладами будуть такі застосунки як WinAmp, Rhythmbox, XMPlay.

Рис. 1.3 Інтерфейс WinAmp

У час розвитку інтернет технологій, для взаємодії з медіа контентом все більшої популярності набирають так звані «стрімінгові сервіси» (Spotify, Apple Music, YouTube), які пропонують весь функціонал стандартних додатків-плеєрів,

але не потребують наявності файлів для програвання на фізичних носіях користувача. Вони також одразу пропонують доступ до великої бібліотеки медіаконтенту і списку інших можливостей. Чому тоді сегмент оффлайн плеєрів досі має деяку популярність? Звичайно, як і все, стрімінгові сервіси мають і низку значних недоліків, які можуть стати переломними для багатьох користувачів. По-перше, більшість сервісів потребують придбання платної щомісячної підписки для отримання доступу до їх контенту, або доступу до певного набору функцій. По-друге, для своєї роботи вони потребують постійного доступу до мережі інтернет, роблячи себе недоступними для певної категорії користувачів. По-третє, безпосередня передача медіаконтенту у режимі реального часу потребує значного стиску файлу, що іноді серйозно впливає на якість звуку. Через ці, та низку інших нюансів, багато людей досі віддають перевагу зберіганню файлів на своєму девайсі, через що доводиться звертатися до додатків-плеєрів.

Також до таких додатків звертаються люди, які працюють у сферах звукозапису або аудіомонтажу. У цих професіях потрібно дуже часто переслухувати велику кількість аудіофайлів у їх максимальній якості, тому вивантажувати їх у онлайн сервіси доволі нерентабельно і не має великого сенсу окрім як для зберігання.

Провівши цей аналіз, ми можемо зробити висновок, що більшість пропозицій ринку додатків аудіо плеєрів є дуже застарілими на сьогоднішній день, але додатки не втратили своєї долі ринку. Інтерфейси багатьох додатків перевантажені не потрібними елементами, що викликає стан замішання та спричиняє незручності під час використовування даних додатків, що значно впливає на загальний UX. Не дивлячись на «класичний» статус багатьох з цих програм, їм всім не вистачає оновлення інтерфейсові складової під сучасні вимоги. Саме ці фактори, та ті, що були описані вищче, спонукали на розробку додатка даної кваліфікаційної роботи.

Для реалізації програмного застосунку був обраний інтерфейс програмування додатків Windows Forms. У своїй основі ця технологія

представляє open-source бібліотеку, яка є частиною сім'ї .NET Framework. Windows Froms є мультиплатформеною API, з її застосуванням можна реалізувати додатки як для настільних комп'ютерів та ноутбуків, так і для планшетних ПК. Здебільшого WinForms дозволяє розробнику розташовувати елементи керування на середовище Windows Form за допомогою технології «drag & drop» і дає змогу модифікувати ці елементи за допомогою файлу коду, який може мати мову C#, VB.NET або будь-яку іншу мову з сімейства .NET. Кожен елемент керування WinForms є екземпляром класу, оскільки WinForms існує як «обгортка», яка має набір класів C++.

Visual Studio від Microsoft дозволяє легше застосовувати технологію WinForms, оскільки розробники можуть швидко перетягувати заготовлені елементи керування з панелі інструментів. У десктопній програмі, WinForms розробник може отримати доступ лише до файлу коду, де він може маніпулювати подіями керування.

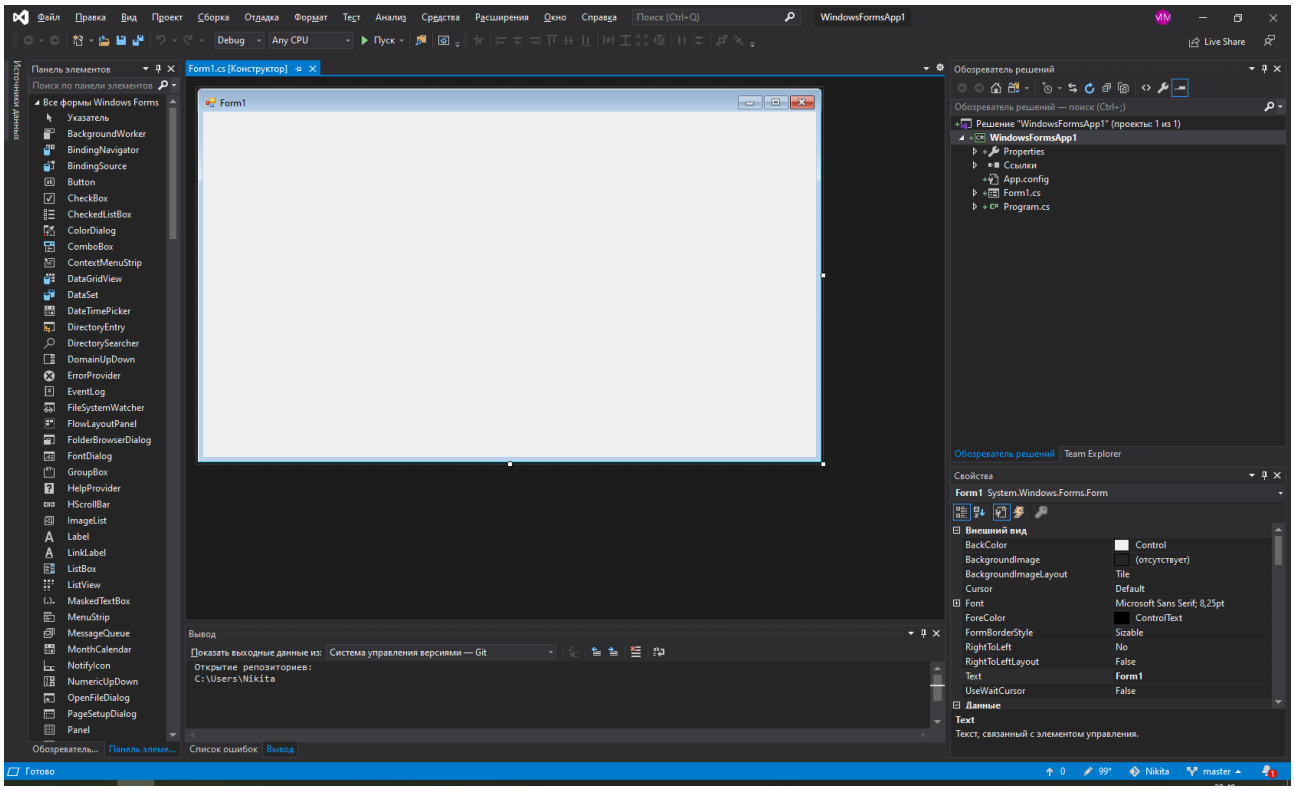

Рис. 1.4 Інтерфейс програми Visual Studio для проекту WinForms

Переваги застосування Windows Forms:

- Велика база документації з розробки додатків;

- Велика кількість прикладів;
- Зручний конструктор на основі Visual Studio;
- Можливість додавання сторонніх елементів з магазину додатків.

Недоліки застосування Windows Forms:

- Деякі сторонні елементи можуть бути розповсюджені на платній основі;
- Без додаткових елементів доволі тяжко налаштовувати дизайн інтерфейсу додатку.

У рамках дослідження технологій також було розглянуто технологію Windows Presentation Foundation або WPF, яка є аналогом WinForms. WPF застосовує мову XAML, яка є різновидом XML, для визначення та підключення окремих частин інтерфейсу. Програми основані на WPF можуть бути розгорнуті як самостійні десктопні застосунки, так і можуть бути вбудованим елементом веб-сторінки.

Основним концептом WPF э об'єднання ряду популярних елементів інтерфейсу користувача у єдиному середовищі. Цими елементами є: 2D/3D-рендерінг, типографіка, векторна графіка, елементи живої анімації, фіксовані та адаптивні документи та готові медіафайли. Після створення, дані елементи можна редагувати, підключати та змінювати відповідно до особливих подій або взаємодії з користувачем.

Переваги застосування WPF:

- Сумісний з Windows Forms;
- Можна використовувати для розробки та проектування як додатків Windows, так і веб-додатків;
- Детальне налаштування стилю інтерфейсів користувача;
- Підтримка векторної та тривимірної графіки;

Недоліки застосування WPF:

- Потребує детальних знань мови розмітки XAML для ство створення реня користувацького інтерфейсу;
- Потребує більшого об’єму оперативної пам’яті, порівняно з Windows Forms.

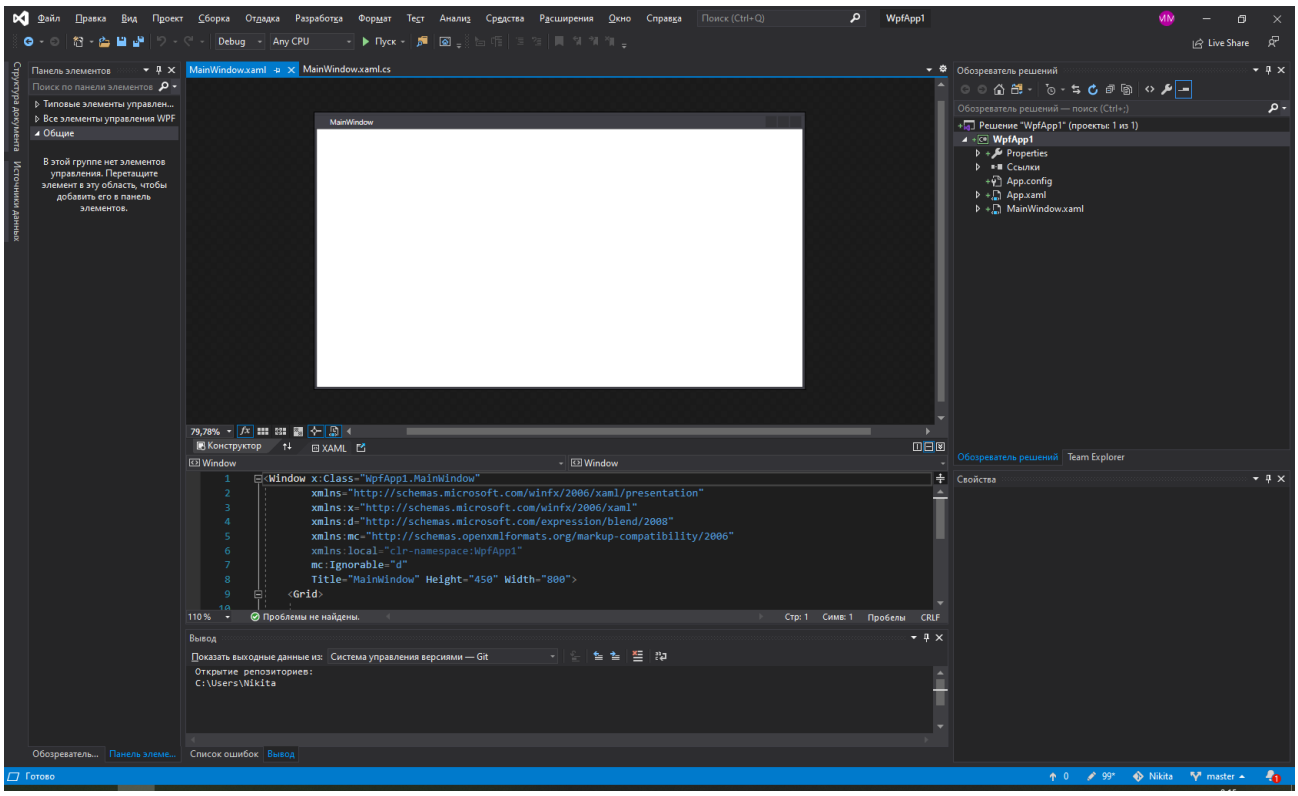

Рис. 1.5 Інтерфейс програми Visual Studio для проекту WPF

Проаналізувавши ці дві технології, можна зробити висновок, що для реалізації додатку цієї кваліфікаційної роботи більш усього підходить перший варіант, а саме – технологія Windows Forms. Вона є перевіреною часом, та показала себе як дуже надійне рішення. Також вона потребує менших обчислювальних потужностей, що додає до компактності, та сумісності додатку. Хоча WPF пропонує більшу гнучкість в налаштуванні елементів користувацького інтерфесу, з застосуванням додаткових фреймворків та бібліотек, можливості Windows Forms також можуть бути розширені і отримати низку своїх переваг.

Бібліотека — це набір частин програми, які виконують звичайні та/або спеціалізовані речі, звільняючи програміста від необхідності «винаходити колесо» під час написання програмного забезпечення. Зазвичай цей набір складається з функцій для виклику та класів об’єктів, які ви можете створити у своєму проекті. Найпоширенішим прикладом можуть бути функції, які оброблюють дати та час, а прикладом спеціалізованих може бути програмне

забезпечення для керування розумним термостатом. Такі спеціалізовані бібліотеки, зазвичай, надаються при покупці самого пристрою.

Використовуючи бібліотеки, програміст може зосередитися на унікальних аспектах програми що розробляється, і не відволікатися на кодування найпростіших речей.

У проекті кваліфікаційної роботи застосовано сторонню бібліотеку TagLib.

TagLib — це бібліотека для читання та редагування метаданих усіх найпопулярніших аудіоформатів, таких як: .mp3, .ogg, .spx, .mpc, .ape, .flac, .wv, .tta, .wma, .m4a, .wav, .aif, .opus. Наразі вона підтримує як ID3v1 стандарт, так і ID3v2 для файлів MP3, коментарі Ogg Vorbis і теги ID3, а також коментарі Vorbis у файлах FLAC, MPC, Speex, WavPack, TrueAudio, WAV, AIFF і ASF.

Переваги бібліотеки TagLib:

- TagLib простий - TagLib пропонує рівень абстракції, який дозволяє легко ігнорувати відмінності між різними форматами файлів та їх реалізацією;
- TagLib є потужним - TagLib надає доступ до реалізацій обробки окремих форматів файлів і надає інструментарій для розширених маніпуляцій з аудіо метаданими;
- TagLib добре задокументований - кожен клас, простір імен та функція TagLib є задокументованими, що значно спрощує розробку;
- TagLib швидкий - тести показали, що він приблизно в 6 разів швидше, ніж id3lib, і в 3 рази швидше, ніж libvorbisfile при читанні тегів (час ЦП);
- TagLib є розширюваним - TagLib не реалізує всі функції ID3v2, але натомість дає можливість авторам додатків розширити бібліотеку TagLib для підтримки конкретних функцій, які їм потрібні в своїх програмах.

Фреймворк (Framework) — це платформа, яка забезпечує основу для розробки програмних додатків. Вона представляє собою «шаблон» робочої програми, який можна вибірково змінювати, додаючи власний або сторонній код. Фреймворк використовує спільні ресурси – такі як бібліотеки, файли

зображень та довідкові документи – і об’єднує їх в один пакет. Цей пакет можна змінити відповідно до конкретних потреб проекту. За допомогою фреймворка розробник може додавати або замінювати функції, щоб надати програмі нову функціональність.

Мета фреймворків і бібліотек одна: розширити діапазон готових функцій, доступних для розробників, оптимізуючи їхнє навантаження та зменшуючи простір для помилок і неефективного коду.

Однак між ними є технічні відмінності. Однією з основних відмінностей між фреймворком та бібліотекою є використання запитів. При використанні бібліотеки програміст сам вирішує, коли і куди її викликати. При використанні фреймворка це диктує фреймворк. Він забезпечує базову структуру та повідомляє програмісту, що для неї потрібно. Тому, необхідний код вставляється розробником і доповнює фреймворк відповідно до потрібної функції. Зрештою, саме фреймворк викликає код, коли йому це потрібно, а також відповідає за виконання програми.

Для розробки елементів інтерфейсу кваліфікаційної роботи було застосовано фреймворк Bunifu UI. Даний фреймворк від компанії Bunifu пропонує набір заготовлених елементів користувацького інтерфейсу, підтримуючих можливість просторої модифікації та зміни під потреби розробника. Фреймворк підтримує Windows Forms, C# та VB.NET.

Переваги:

- Гнучкість – кожний елемент Bunifu UI має обширний потенціал для модифікації та налаштування, доступний в панелі налаштувань елементів у Visual Studio;
- Продуктивність – Bunifu UI майже не навантажує систему, що позитивно впливає на загальну продуктивність програми;
- Відповідність часу – елементи, включені до фреймворку, мають сучасний та стильний дизайн.

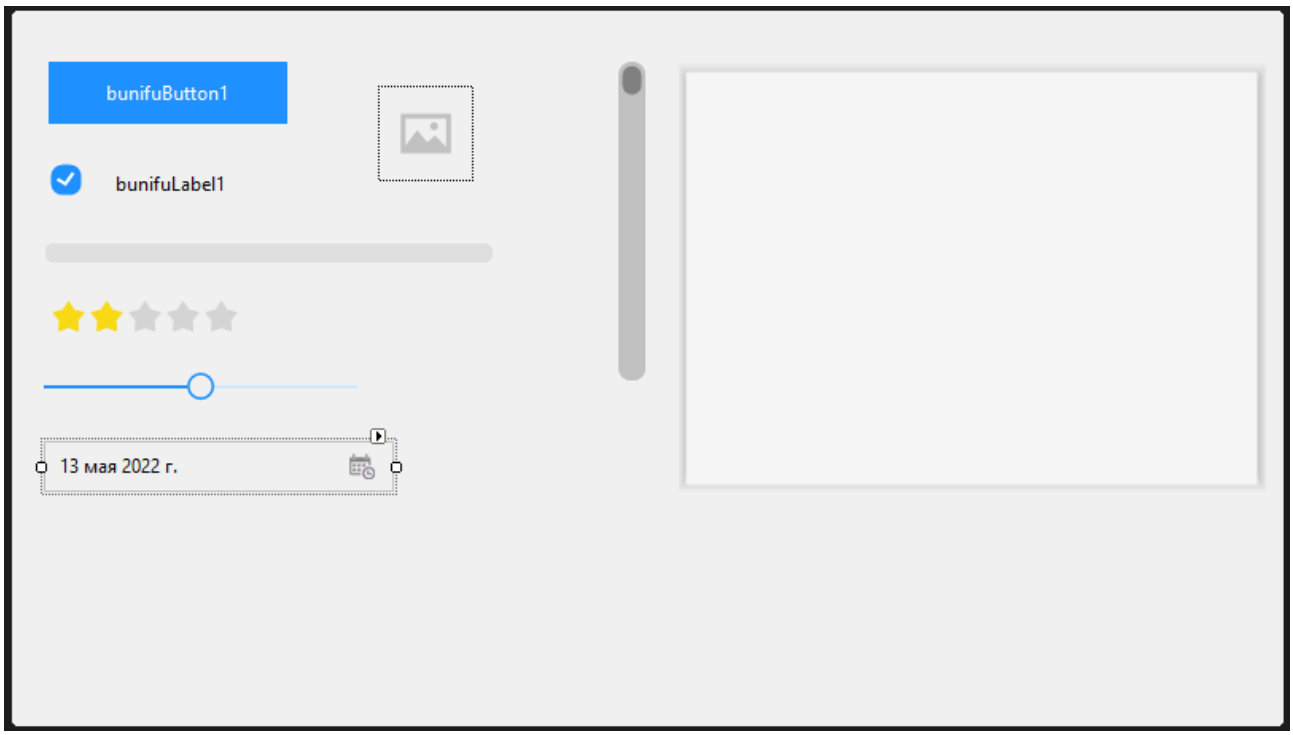

Рис. 1.6 Приклад деяких елементів які входять до складу Bunifu UI

1.2. Призначення розробки та галузь застосування

Виконанний в рамках кваліфікаційної роботи додаткок має назву «Розробка додатка "аудіо-плеєр" на мові програмування C#».

Призначення розробки - надати зацікавленим у темі користувачам зручний та компактний додаток-аудіо плеєр, який об’єднає у собі свіжість сучасних дизайн трендів та основну функціональність усіх представлених на ринку додатків конкурентів. Програма може бути застосована як для прослуховування музикальних композицій, так і для звичайних звукозаписів завдяки підтримки широкого списку різних аудіоформатів.

1.3. Підстави для розробки

Відповідно до освітньої програми, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу.

Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується з наказом ректора.

Таким чином підставами для розробки (виконанням кваліфікаційної роботи) є:

- освітня програма спеціальності 122 “Комп’ютерні науки”;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету “Дніпровська політехніка” № 268-с від 18.05.2022р;
- завдання на кваліфікаційну роботу на тему “ Розробка додатка "аудіо-плеєр" на мові програмування C#”.

1.4. Постановка завдання

Метою кваліфікаційної роботи є написання додатку-аудіо плеєру для програвання файлів аудіоформату.

Додаток повинен втілювати низку пунктів, а саме:

- Мати практичний та зручний дизайн;
- Відповідати сучасним стандартам UI;
- Надавати надійність та стабільність при використанні;
- Підтримувати та програвати усі заявлені аудіоформати;

Задля досягнення вищеперелічених вимог необхідно:

- Провести аналіз пропозицій на ринку та установити їх недоліки;
- Зробити сортування потрібного і застарілого функціоналу даних додатків;
- Установити найбільш підходящу технологію для розробки додатку такого типу;
- Створити алгоритм реалізації додатку;
- Безпосередньо кодування додатку;
- Написання користувацьких рекомендацій з використання та застосування додатку.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт повинен задовільняти наступні вимоги з функціональності:

- Інтуїтивно зрозумілий дизайн програми;
- Наявність основних навігаційних кнопок додатку-плеєра («Програвання», «Пауза», «Стоп», «Наступний трек» тощо);
- Кнопки додаткової функціональності («Перемішування плейлиста», «Зациклювання треку/плейлиста» тощо)
- Наявніть слайдеру гучності;
- Наявність слайдеру позиції для навігації по поточному треку;
- Вивід інформації з метаданих треку;
- Кнопка обирання файлів для програвання, як і функція «drag & drop»;
- Кнопка трансформації додатку у більш компактний вигляд;
- Графічний вимірювач рівня звуку типу «хвиля».

1.5.2. Вимоги до інформаційної безпеки

Додаток не вимагає додаткових вимог до інформаційної безпеки так як є незалежною скомпільованою програмою.

1.5.3. Вимоги до складу та параметрів технічних засобів

Додаток потребує більшість вбудованих можливостей .NET Framework 4.0 і тому вимагає установки його runtime версії для свого запуску. Також необхідно

мати наступні мінімальні характеристики системи, на якій буде відбуватися запуск:

- Наявність ОС Windows версій 7, 8 , 8.1 або 10;
- ЦП розрядністю x86 або x64;
- Частоту ЦП мінімум у 2 GHz;
- Найпростіший відеоадаптер (GPU);
- Оперативна пам’ять мінімум 512 мегабайт;
- Аудіокарта з підтримкою DX9;

1.5.4. Вимоги до інформаційної та програмної сумісності

Додаток був розроблений за допомогою мови програмування C#. Середовищем розробки є інтегроване середовище розробки Microsoft Visual Studio. Додаток потребує наявності .NET Framework не молодше версії 4.0 та DirectX API не молодше версії 9.0.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Результатом даної кваліфікаційної роботи має стати додаток-аудіоплеєр, написаний на мові програмування C# з застосуванням додаткового фреймворку та бібліотек.

Кінцевий додаток націлений на користувачів, які бажають прослухати аудіофайли. Файли можуть містити як музикальні композиції, так і голосові записи або звукові записи.

Основне призначення додатку:

- програвання файлів формату аудіо;
- зберігання списку програвання файлів, якщо ці файли досі присутні на дискових носіях користувача;
- надання функціонального блоку навігації;
- сумісність з ОС Windows версій 7, 8, 8.1, 10 та 11;
- підтримка наступних форматів: .mp3, .wav, .flac, .m4a, .aac, .ogg, .aif, .aiff, .midi;

Задля досягнення можливості задовольняти ці призначення, додаток повинен мати наявність технології Open file dialog, яка надає можливість обирати та відкривати файли.

2.2. Опис застосованих математичних методів

Додаток даної кваліфікаційної роботи не використовує спеціальних математичних функцій. В ході розробки додатка було використано вбудовані базові математичні методи мови C#.

Список методів: оператор додавання (+), оператор віднімання (-), оператор ділення (/), оператор множення (*), оператор інкременту (++), оператор дорівнює (==), оператор не дорівнює (!=), оператор більше (>), оператор менше (<).

Оператор додавання (+) призначений для обчислення суми призначених до нього операндів.

Оператор віднімання (-) призначений для обчислення різності призначених до нього операнд, віднімаючи правий операнд від лівого.

Оператор ділення (/) призначений для ділення призначених до нього операнд, ділючи лівий операнд на правий.

Оператор множення (*) застосовується для обчислення математичного добутку операнд, які до нього призначені.

Оператор інкременту (++) проводить операцію збільшення операнда на 1. Може мати вигляд як префіксу (++a), так і суфіксу (a++). Префіксальний варіант запису спочатку збільшує операнд, а потім присвоює його до змінної. Суфіксальний варіант запису спочатку присвоює значення операнди до змінної, а потім збільшує значення операнди.

Оператор дорівнює (==) перевіряє, чи дорівнює один оеранд іншому, або якомусь значенню. Аналогічно, оператор не дорівнює (!=) перевіряє, чи не дорівнює один операнд іншому, або заданому значенню.

Оператори більше (>) та менше (<), відповідно, перевіряють операнд на дорівнюваня чи не дорівнюваня іншому операнду або заданому значенню.

2.3. Опис використаних технологій та мов програмування

Додаток кваліфікаційної роботи був розроблений за допомогою наступних технологій та засобів:

- інтегроване середовище розробки MS Visual Studio;
- технологія Windows Forms;
- шаблон MVP;
- мова програмування C#.

Інтегроване середовище розробки (IDE) — це програмне забезпечення для створення програм, яке поєднує звичайні інструменти розробника в єдиний графічний інтерфейс користувача (GUI). IDE зазвичай складається з:

- Редактор вихідного коду: текстовий редактор, який може допомогти в написанні програмного коду з такими функціями, як підсвічування синтаксису за допомогою візуальних підказок, забезпечення автоматичного заповнення для певної мови та перевірка помилок під час написання коду.
- Автоматизація локальної збірки: утиліти, які автоматизують прості, повторювані завдання в рамках створення локальної збірки програмного забезпечення для використання розробником, як-от компіляція вихідного коду комп'ютера в двійковий код, упаковка двійкового коду та запуск автоматизованих тестів.
- Налагоджувач: програма для тестування інших програм, яка може графічно відображати розташування помилки в оригінальному коді.

Застосування IDE дозволяє розробникам швидко починати програмувати нові додатки, оскільки немає потреби вручну конфігурувати та інтегрувати декілька утиліт як частину процесу налаштування. Розробникам також не потрібно витрачати години вивчаючи, як використовувати різні інструменти, коли кожна утиліта представлена в єдиному робочому місці. Це може бути особливо корисно для залучення нових розробників, які можуть покладатися на IDE, щоб освоювати стандартні інструменти та робочі процеси. Здебільшого, основна частина функцій IDE призначена для економії часу, як-от інтелектуальне завершення коду та автоматичне генерування коду, що усуває необхідність вводити повні послідовності символів.

Інші функції IDE покликані допомогти розробникам краще організувати свій робочий процес і вирішувати проблеми. IDE аналізують код таким, яким він був написаний, тому помилки, спричинені людським фактором, відмічаються в режимі реального часу. Оскільки усі утиліти представлені в одному графічному інтерфейсі, розробники можуть виконувати потрібні дії без перемикання між додатками. Виділення синтаксису також є поширеним у більшості IDE, які використовують візуальні підказки. Деякі IDE додатково включають браузери класів та об'єктів, а також діаграми ієрархії класів для певних мов.

Сьогодні більшість команд розробників вибирають попередньо налаштовану IDE, яка найкраще підходить для їх конкретного випадку.

Для розробки додатку кваліфікаційної роботи був обраний IDE від компанії Microsoft - Visual Studio. Його використовують як для розробки десктопних програм, так і для розробки веб-сайтів та додатків. Також Visual Studio підходить для розробки мобільних додатків. Visual Studio базується на платформах розробки ПЗ Microsoft, таких як Windows API, Windows Forms, Windows Presentation Foundation і Microsoft Silverlight.

Visual Studio містить у собі редактор коду, який застосовує технологію IntelliSense, що є компонентом автоматичного завершення коду, а також рефакторингу коду. Вбудований у середовище відлагоджувач (Debuger) працює у двох режимах: як відлагоджувач рівня джерела та як відлагоджувач на машинному рівні. Інші вбудовані інструменти містять: профайлер коду, конструктор для створення додатків із застосуванням GUI, інструмент для дизайну у веб середовищі, класовий конструктор і дизайнер схем баз даних.

Visual Studio підтримує плагіни, що розширюють функціональність майже на кожному рівні системи, включаючи також додавання підтримки для систем контролю джерел і додавання наборів інструментів, як-от редактори та візуальні дизайнери для мов.

Visual Studio нативно працює з 36 основними мовами програмування та дозволяє редактору коду і відлагоджувачу підтримувати майже будь-яку мову програмування, за умови, що існує потрібна служба. Вбудований набір мов включає в себе: C, C++, Visual Basic .NET, C#, JavaScript, TypeScript, XML, XSLT, HTML і CSS та інші. Підтримка додаткових мов, таких як Python, Ruby, Node.js та M, серед інших, доступна з застосуванням плагінів.

Стартова версія Visual Studio доступна безкоштовно. Microsoft має таку позицію щодо цієї версії — «Безкоштовна повнофункціональна IDE для студентів, відкритих і індивідуальних розробників».

Visual Studio нативно не підтримує будь яку мову кодування, рішення чи набір інструментів. Але, він дозволяє додавати розширені функціональні

можливості, які мають бути закодовані у форматі VSPackage. Після встановлення даного пакету, додана функція стає доступною у програмі як сервіс. Visual Studio надає три основних сервіси:

- SVsSolution, який дає можливість перераховувати проекти та рішення;
- SVsUIShell, який надає набір функцій вікон, панелі інструментів та інтерфейсу користувача;
- SVsShell, який займається реєстрацією VSPackages.

Крім того, середовище також відповідає за координацію та надання зв'язку між службами. Усі редактори, конструктори, типи проектів та інші інструменти реалізовані як набір VSPackages. Visual Studio застосовує технологію COM для доступу до пакетів VSPackages. SDK Visual Studio також містить фреймворк Managed Package Framework (MPF), який представляє з себе набор "обгорток" керування навколо COM-інтерфейсів, які дозволяють кодувати пакети будь-якою мовою, сумісною з CLI.

Підтримка сторонніх мов програмування додається з застосуванням спеціального пакету VSP, який називається Language service (Мовною службою). Language service визначає інтерфейси, застосування яких може реалізувати пакет VSPackage, для того щоб додати підтримку сторонніх функцій. Пакет функцій, котрий можна додати таким чином, може містити в собі:

- підсвітка синтаксису;
- автоматичне доповнення операторів;
- перевірка на відповідність дужок;
- підказки з інформацією про параметри;
- списки учасників та маркери помилок для фонової компіляції.

Якщо потрібний інтерфейс є реалізованим, пакет функцій для мови стане доступним. Реалізації також спроможні повторно застосовувати код синтаксичного аналізатора чи компілятора мови. Мовні служби можуть бути написанні або у нативному коді, або у керованому. Для нативного коду можна застосовувати або власні COM інтерфейси, або Babel Framework. Для керованого коду MPF включає "обгортки" для кодування мовних служб.

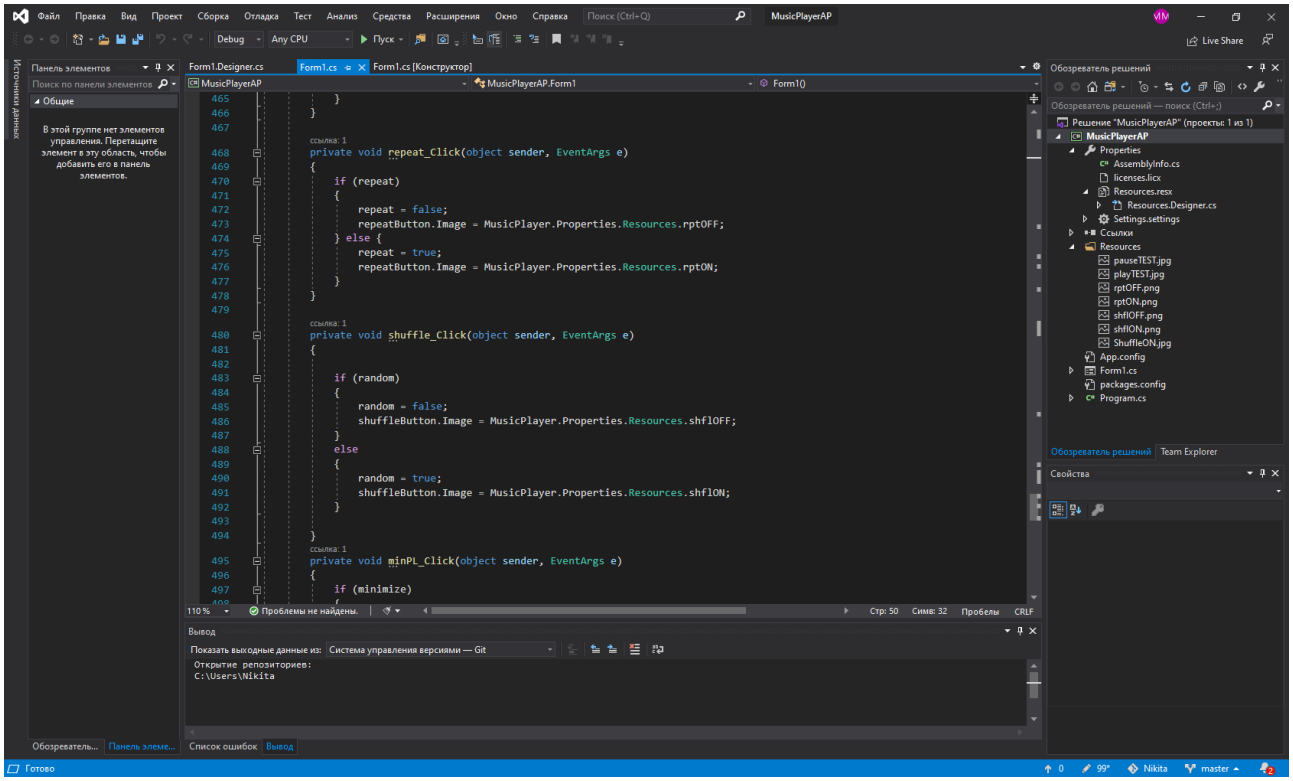

Рис. 2.1. Інтерфейс IDE Microsoft Visual Studio

Visual Studio підтримує одночасне використання кількох екземплярів середовища (кожне із власним набором VSPackages). Ці екземпляри застосовують різні, особисті "вулики реєстру" для зберігання свого стану конфігурації та мають відмінний від один одного AppId (ідентифікатор програми). Примірники запускаються в індивідуальному для кожнго AppId .exe файлі, який обирає потрібний AppId, встановлює кореневий вулик і запускає IDE. VSPackages, зареєстровані для одного AppId, інтегруються з іншими VSPackage. Різні версії продуктів Visual Studio створюються за допомогою різних AppIds.

Visual Studio не містить вбудовану підтримку функції керування джерелами, але надає два альтернативні способи інтеграції систем керування кодом із IDE. Source Control VSPackage може надати власний налаштований UI. На відміну від цього, плагін керування кодом із використанням MSSCCI (Інтерфейс керування вихідним кодом Microsoft) реалізує набір функцій, які

застосовуються для реалізації різноманітних функцій керування кодом зі стандартним інтерфейсом користувача у Visual Studio.

Під час розробки додатку було застосовано такий шаблон програмування як Model View Presenter (MPV). Model View Presenter — це парадигма проектування, яка у архітектурному плані є виведенням шаблону MVC, та зазвичай використовується для створення інтерфейсів користувача.

Шаблон програмного забезпечення «модель-представлення-пред'явник» виник на початку 1990-х років у Taligent, спільному підприємстві Apple, IBM і Hewlett-Packard. У 1998 році він був популяризований Dolphin Smalltalk, а потім у 2006 році Microsoft прийняла MVP для програмування інтерфейсу користувача в платформі .NET.

Типові взаємодії, що відбуваються в архітектурі MVP, можна зрозуміти з наступної діаграми:

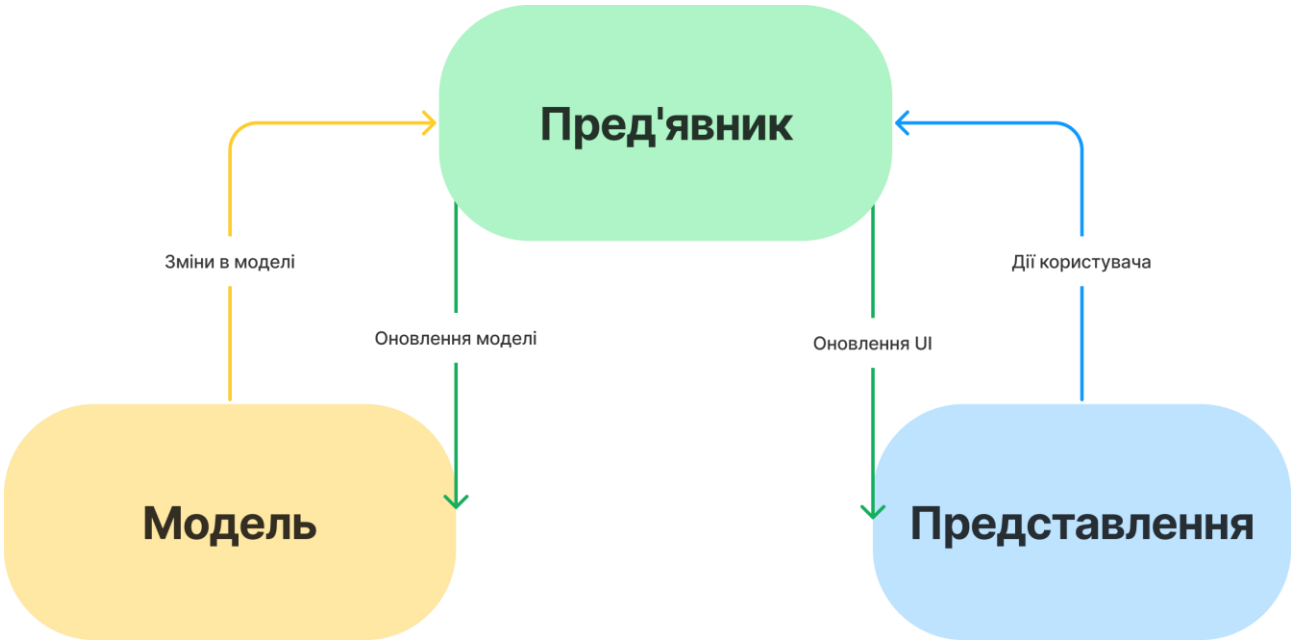

Рис. 2.2. Діаграма моделі Model View Presenter

MVP з'явився як архітектурний шаблон користувальницького інтерфейсу, розроблений для полегшення автоматизованого модульного тестування та покращення розділення проблем у логіці презентації.

Модель — це інтерфейс, що визначає дані, які відображатимуться або з якими треба провести дії в інтерфейсі користувача. Модель — це рівень даних, який відповідає за обробку бізнес-логіки та за зв'язок з мережею та рівнями бази даних. Обов’язки моделі включають використання API, кешування даних, керування базами даних тощо.

Перегляд — який зазвичай реалізується за допомогою дії, міститиме посилання на пред’явник. Єдине, що буде робити перегляд, це викликати метод із пред’явника щоразу, коли виконується дія інтерфейсу. Він несе відповідальність лише за подання даних у спосіб, визначений пред’явником. Функціональність представлення, як правило, зводиться до мінімуму, і воно є пасивним компонентом, який переміщує бізнес-логіку до пред’явника. Перегляд також захищений від моделі, делегуючи всю взаємодію через посередника.

Пред’явник — несе відповідальність за роль посередника між Переглядом та Моделлю. Він отримує дані з Моделі та повертає їх у форматі перегляду. Але на відміну від типового MVC, він також визначає, що станеться під час взаємодії з Переглядом. Завдяки цьому Перегляд і Пред’явник тісно співпрацюють, так як вони повинні мати посилання один на одного. Таким чином, зв’язок між Пред’явником і відповідним представленням визначається в інтерфейсі. Пред’явник також відокремлений безпосередньо від Перегляду і спілкується з ним через цей інтерфейс.

Для початку роботи з середовищем Microsoft Visual Studio, користувачу потрібно створити новий проект, або відкрити вже існуючий. Зробити це можна через Вікно запуску, яке з’явиться одразу після запуску програми. В ньому також присутня функція клонування коду, задля його отримання та внесення змін до нього. Вікно запуску Visual Studio Community 2019 має наступний вигляд:

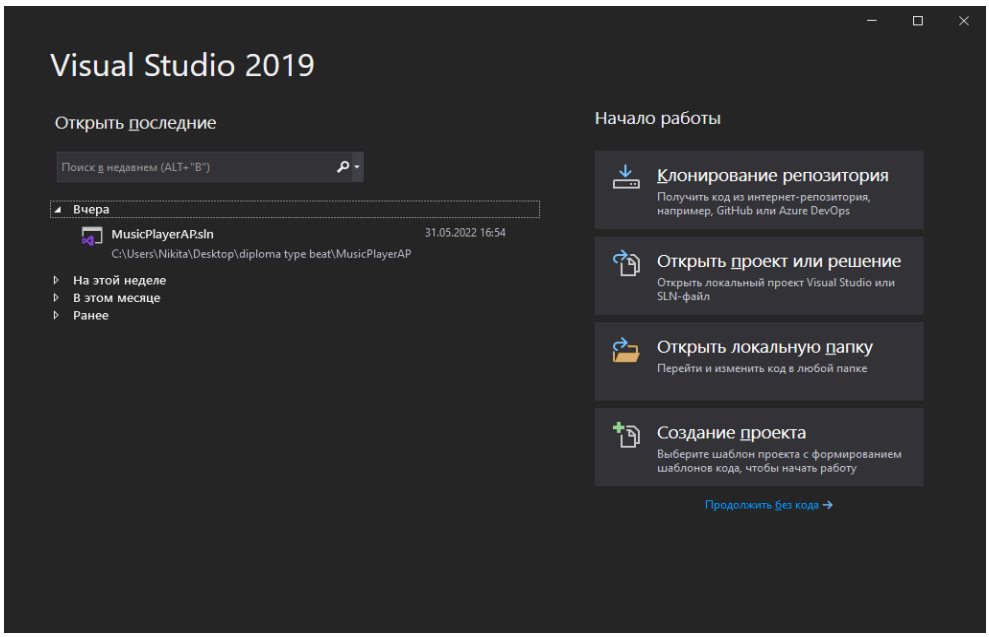

Рис. 2.3. Вікно запуску Microsoft Visual Studio.

Якщо обрати варіант «Продовжити без коду», то Visual Studio відкриє середовище розробки без конкретного файлу проекту чи завантаженого коду.

Для того щоб створити новий проект у Visual Studio, потрібно пройти наступні кроки:

- у Вікні запуску обрати пункт «Створення проекту»;
- обрати у списку потрібний тип проекту, або скористатись функцією пошуку у верхній частині вікна (Рис. 2.4.);
- задати ім’я проекту;
- вказати місце, де проект буде зберігатися на дисковому носії;
- задати ім’я вирішення.

Також у цьому віні можна обрати потрібну версію платформи .NET (Рис. 2.5.).

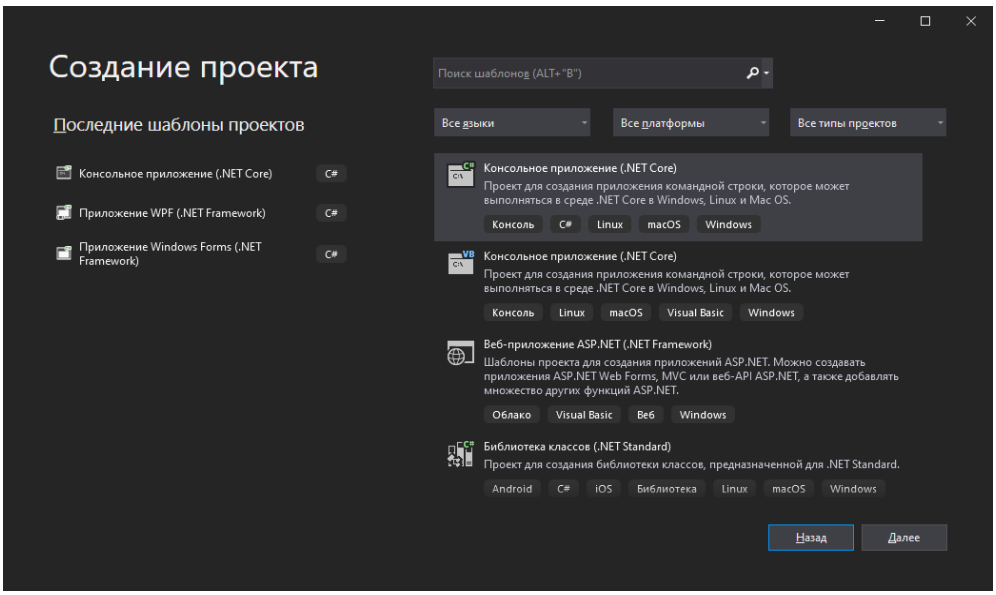

Рис. 2.4. Вікно створення проекту Microsoft Visual Studio

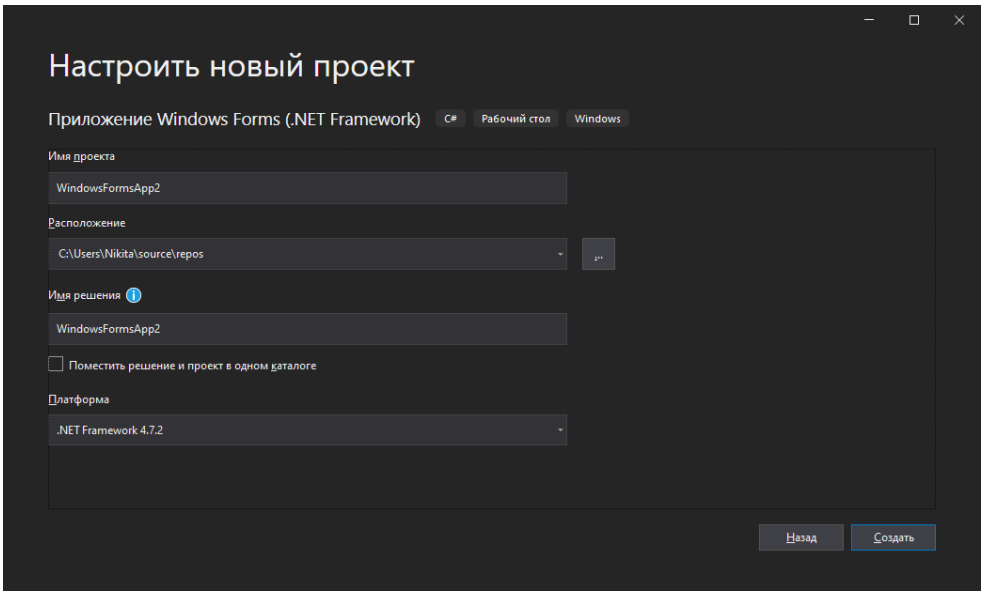

Рис. 2.5. Вікно налаштування проекту Microsoft Visual Studio

Після пророблених дій, проект буде створено та відкрито робоче середовище Mirosoft Visual Studio. При обраному типі проекту «Додаток Windows Forms» програма відкриє вікно конструктору Windows Forms.

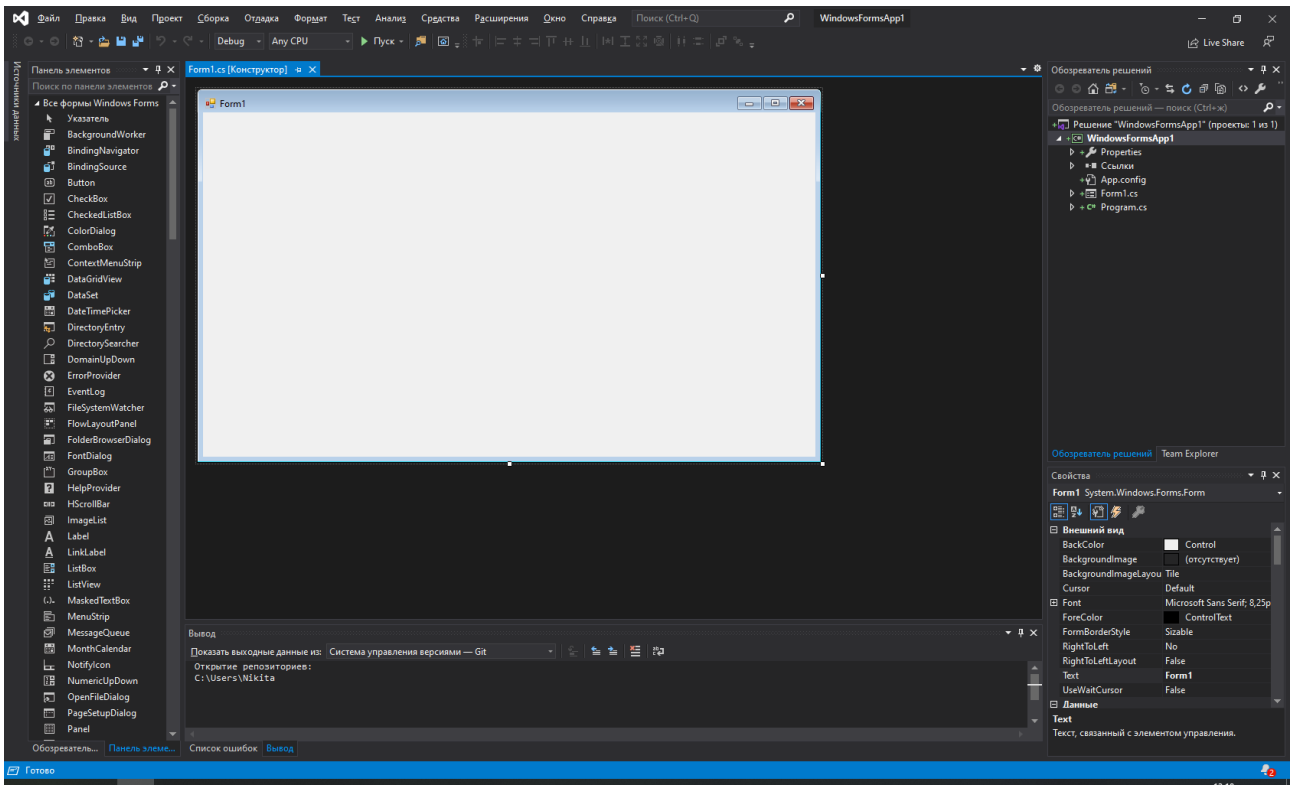

Рис. 2.6. Середовище розробки Microsoft Visual Studio

Для того, щоб перейти до редагування коду програми, потрібно обрати пункт «Program.cs» у вікні Оглядач рішень. Також це вікно містить наступні пункти:

- Налаштування параметрів проекту;
- Список локальних ресурсів проекту;
- Посилання на підключенні бібліотеки та фреймворки;
- Дизайнер середовища Windows Forms.

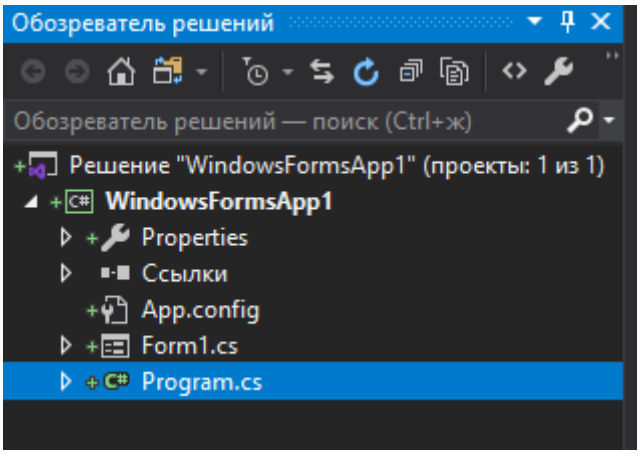

Рис. 2.6. Вікно «Оглядач рішень»

Додавання елементів на середовище Windows Form у Microsoft Visual Studio реалізовано за принципом технології «перетягни та відпусти». У вікні конструктора, в лівій частині екрану, за замовчуванням відкривається вікно «Панель елементів», у якому представлено список усіх елементів, які можуть бути застосовані у даному проекті.

Елементи, додані до середовища через підключення сторонніх плагінів також з'являються у цьому вікні. Також це вікно надає можливість перейменовувати елементи, клонувати елементи та видаляти їх за бажанням користувача.

Елементи у вікні відсортовані по типу у різні категорії задля надання максимальної зручності при користуванні.

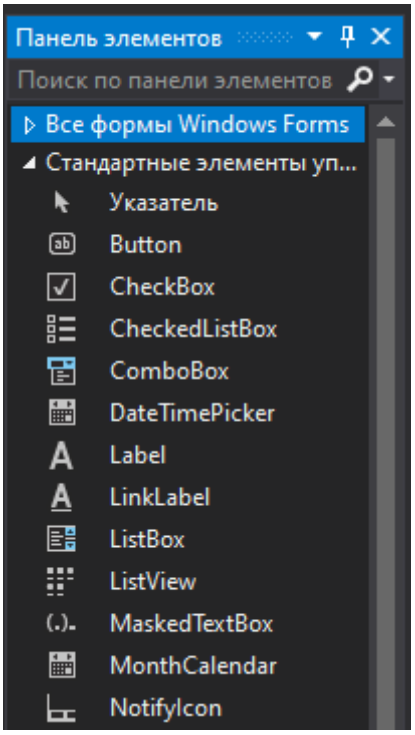

Рис. 2.6. Вікно «Панель елементів»

Також, додавання елементів на середовище Windows Form може бути здійснене безпосередньо кодуванням потрібного елементу.

Проект типу «Додаток Windows Forms» реалізується за допомогою мови програмування C#.

C# (Сі шарп) — це об'єктно-орієнтована мова, яка дозволяє розробникам створювати різноманітні безпечні та надійні програми для виконання на платформі .NET. За допомогою цієї мови користувач може створювати:

- клієнтські програми Windows;
- веб-сервіси на мові XML;
- розподілені компоненти;
- клієнт-серверні програми;
- програми баз даних.

Мова C# підтримує такі технології, як:

- інкапсуляція;
- успадкування;
- поліморфізм.

C# підтримує покажчики та небезпечний код для доступу до пам'яті, а також підтримує функцію взаємодії. Він зручний, та допомагає користувачеві з легкістю виконувати всю потрібну йому роботу, подібно до програм, написаних з застосуванням мови C++.

Основні можливості C#:

- він використовується для спрощення операторів і покажчиків, які використовуються в інших мовах програмування;
- він має функцію динамічного пошуку, яка усуває необхідність посилатися на тип об’єктів, що повертаються у операторі присвоєння;
- іменовані аргументи допомагають користувачеві вказувати параметри методу за їх іменами;
- додаткові параметри визначаються в кінці списку параметрів після усіх необхідних;
- він є типобезпечним, що за замовчуванням забезпечує неявні перетворення типу даних у похідний тип;
- члени переліку поміщаються в область його дії;
- збір сміття відбувається автоматично через процес завершення;

- функція дисперсії дозволяє вказувати вхідні та вихідні параметри для загальних типів;
- здійснює виклики до методів COM і Interop з меншою кількістю коду в системі;
- код C# може бути скомпільований на різних платформах.

Фреймворк .NET відповідає за виконання програми C# в системі. Framework — це комбінація загальномовного середовища виконання та набору бібліотек класів. Реалізація Common Language Infrastructure (CLI) здійснюється за допомогою Common Language Runtime (CLR).

Платформа Microsoft .NET складається з таких основних компонентів:

- Common Language Runtime;
- Бібліотека базового класу;
- Інтерфейси програм користувача.

Діаграма архітектури для фрейморку .NET зі зв'язком із C# наведена нижче (Рис. 2.7.).

Common Language Runtime (CLR) є найважливішим компонентом платформи .NET. Це середовище, де виконуються всі програми, що використовують технологію .NET.

Послуги, що надаються CLR, включають:

- компіляцію коду;
- виділення пам’яті;
- збір сміття.

Код перекладається на мову Intermediate (IL). Цей код може виконуватися на різних платформах.

IL перетворюється на машинну мову під час виконання компілятором Just – In – Time. Під час компіляції JIT код перевіряється на типобезпеку. CLR складається із загального набору правил, яким дотримуються всі мови платформи .NET. Цей набір правил відомий як Common Language Specification (CLS). CLS допомагає об’єктам і додаткам, створеним певною мовою, взаємодіяти з об’єктами та програмами іншої мови.

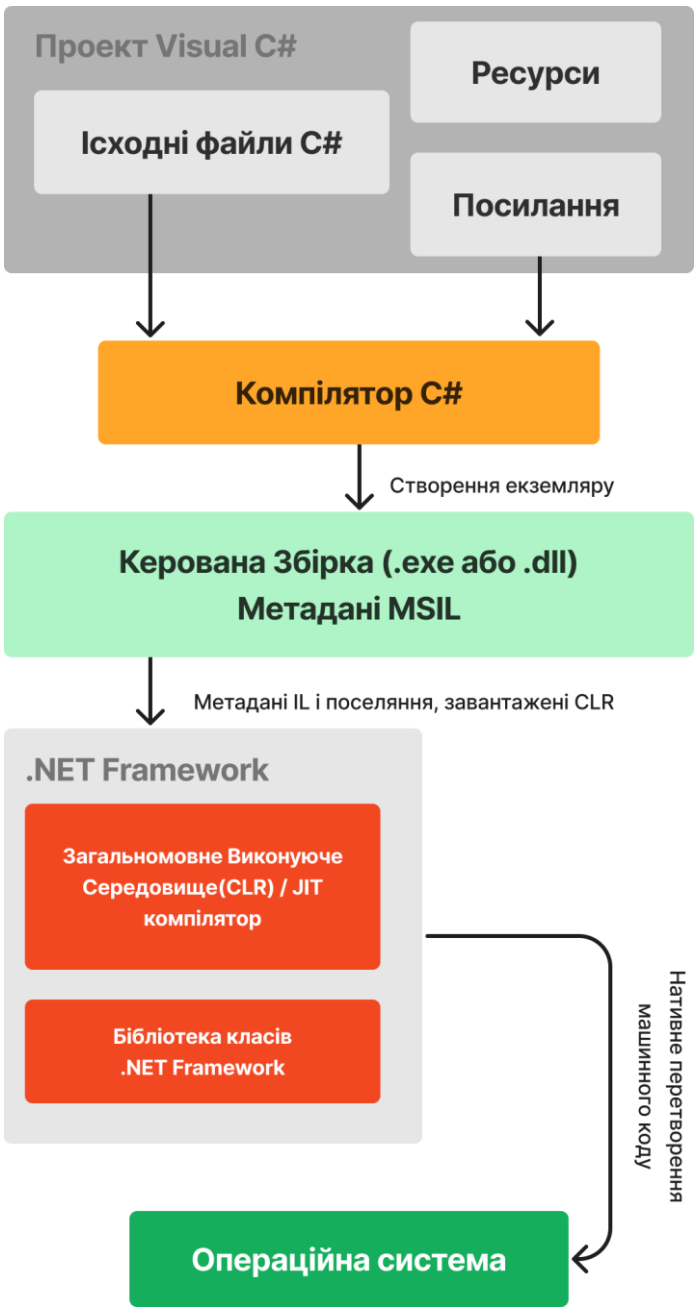

Рис. 2.7. Діаграма архітектури для фрейморку .NET зі зв'язком із C#

CLS має специфікацію, визначену як Common Type System (CTS). Він забезпечує систему типів, спільну для всіх мов. CTS використовується для:

- визначення типів даних оголошення;
- використання та керування коду під час виконання.

Під час виконання, CLR відповідає за такі завдання:

- Завантаження збірок та визначення просторів імен: збірки завантажуються в пам'ять. Після завантаження збірок CLR визначає простір імен для коду в збірках. Простіри імен — це колекція класів.
- Компіляція JIT: перед виконанням коду, код формату IL перетворюється на машинну мову компілятором JIT. У процесі верифікації, код IL перевіряється на наявність місць у пам’яті, вільних для доступу методів, викликаних за допомогою правильно визначених типів.
- Збір сміття: процес збирання сміття виконується після того, як JIT-компіляція керує виділенням і звільненням пам'яті для програми.

Бібліотека базових класів — бібліотека класів .NET Framework працює з будь-якою мовою .NET, як-от VB.NET, VC#, VC++.NET. Бібліотека класів надає класи, які можна використовувати в коді для виконання загальних завдань програмування. Бібліотека класів .NET Framework складається з просторів імен, що містяться в збірках.

Простіри імен — це простіри, які допомагають користувачеві створювати логічні групи пов'язаних класів та інтерфейсів. Це дозволяє користувачеві організовувати класи, які можуть використовуватися для доступу іншими програмами. Їх можна використовувати, щоб уникнути конфліктів між класами, які мають однакові імена.

Збірки — це єдиний блок розгортання, який містить всю інформацію щодо реалізації класів, структур та інтерфейсів. Збірка зберігає інформацію про себе. Інформація відома як метадані.

На рівні презентації .NET надає користувачеві три типи інтерфейсів. Вони такі, як зазначено нижче:

- Windows Forms: використовуються в програмах на базі ОС Windows.
- Веб-форми: використовуються у веб-додатках для реалізації інтерфейсу користувача. Вони надають користувачеві інтерфейс веб-браузера.
- Консольні програми: використовуються для створення консольних програм на основі символів, які виконуються через командний рядок.

Фреймворк .NET надає інтерфейс під назвою Web Services для зв’язку з віддаленими компонентами.

Нижче наведено список можливостей платформи .NET:

- ADO.NET: включає підтримку типів, які визначає користувач, операцій з базою даних, типів даних XML, підтримку кількох активних наборів результатів;
- .NET Remoting: підтримує обмін загальними типами. Це покращує продуктивність віддалених кластерів балансу навантаження мережі;
- XML: користувач може перевірити дерево DOM, що зберігається в екземплярі XmlDocument, через XPathNavigator;
- Windows Presentation Foundation (WPF): використовується для створення клієнтської програми Windows. Він включає такі функції, як прив’язка даних, макет, 2-D і 3-D графіка, анімація, стилі, шаблони, документи, медіа, текст;
- Windows Communication Foundation (WCF): це уніфікує модель програмування, яка використовується для створення сервіс-орієнтованих програм;
- Windows WorkFlow Foundation (WWF): використовується для забезпечення моделі програмування, механізму виконання, інструментів для створення програм робочого процесу;
- Паралельні обчислення: це модель програмування для написання багатопоточного та асинхронного коду, що спрощує роботу розробників додатків;
- Керування розширюваністю Framework: це нова бібліотека в платформі .NET, яка допомагає створювати розширювані програми;
- Сумісність програм і розгортання: .NET Framework сумісний із програмами, створеними з попередніми версіями .NET Framework;
- Автоматичне керування ресурсами: керування ресурсами для будь-якої програми здійснюється автоматично через фреймворк;

- Неявні перетворення: фреймворк .NET неявно перетворює тип локальних змінних, масивів та інших компонентів, оголошених у програмі;

2.4. Опис структури програми та алгоритмів її функціювання

Спроектований та розроблений у рамках кваліфікаційної роботи додаток представляє собою аудіоплеєр, що комбінує у собі зручність простого дизайну та функціональність багатьох популярних конкурентів на ринку. Увесь функціонал додатку реалізовано в одному вікні, що дозволяє швидше та ефективніше працювати з інтерфейсом навігації, та відразу дає доступ до повного списку можливостей. Вигляд додатку представлено на Рис. 2.8.

Рис. 2.8. Загальний вигляд інтерфейсу додатку

Задля того, щоб почати використовувати програму, користувачу необхідна наявність файлів аудіоформату, що підтримує дана програма, на своєму дисковому носії. Програма працює тільки з локальними файлами та не має функції підключення та синхронізації з хмарними сховищами.

Алгоритм функціонування програми простий: після завантаження файлів, програма автоматично починає програвати перший у списку. Програма автоматично зупинить програвання, коли список завантажених треків дійде до кінця. За бажанням, користувач може тимчасово приривати програвання треку, натиснувши на кнопку «Пауза». Продовжити програвання треку можна натисканням на кнопку «Плей». Також, користувач може повністю привати програвання файлу натисканням на кнопку «Стоп», що автоматично очистить виведену на екран інформацію про трек, який було програно останнім.

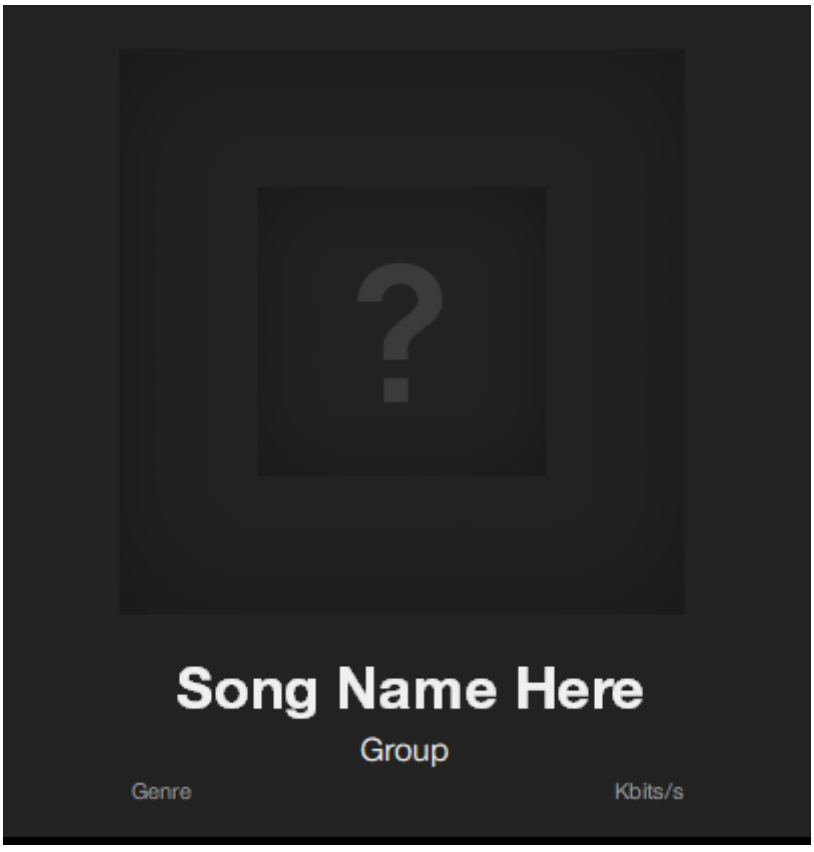

Рис. 2.9. Стан програми після очистки, натисканням кнопки «Стоп»

Алгоритм натискання на кнопки реалізовано за допомогою елементу «bunifuImageButton». Його застосованно у:

- кнопка «Плей/Пауза»;
- кнопка «Наступний трек»;
- кнопка «Попередній трек»;
- кнопка «Стоп»;
- кнопка «Зациклювання треку»;
- кнопка «Перемішати програвання»;
- кнопка «Відкрити/закрити плейліст»;
- кнопка «Обрати файли»;
- кнопка «Видалити трек»;
- кнопка «Закрити програму»;
- кнопка «Мінімізувати програму».

Алгоритм регулювання гучності та навігації позиції у треку реалізовано за допомогою елементу «bunifuSlider». Усі ці елементи входять до Bunifu Framework.

Плейліст, чи список програвання треків, використовує стандартний елемент Visual Studio – listBox.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

В даній кваліфікаційній роботі оброблюються дані користувача, які він завантажує до додатку у форматі аудіофайлів. Вони є вхідними даними.

Оскільки додаток призначений тільки для відтворення файлів, ніяких маніпуляцій з ними не відбувається. Вихідними даними є відтворені додатком звукові хвилі, та графічне зображення цих хвиль за допомогою «волюметра».

2.6. Опис розробленого додатку

2.6.1. Використані технічні засоби

При розробці та тестуванні системи була використана клієнтська персональна ЕОМ з наступними мінімальними характеристиками:

– процесор Intel Core i2 Duo 2.5 GHz;
– відеокарта Nvidia GeForce 8600;
– монітор 60Hz;
– не менше 2000Мб ОЗУ;
– 50Мб вільного місяця для розробленого додатку;
– клавіатура;
– комп’ютерна миша;
– інтегрована аудіокарта;

2.6.2. Використані програмні засоби

Для розробки додатку було використано безкоштовний IDE від Microsoft – Visual Studio. Версія IDE – Community 2019. Дизайн програми було розроблено за допомогою програм Adobe Photoshop 2021 та онлайн сервісу Figma.

2.6.3. Виклик та завантаження програми

Для того, щоб почати роботу з програмою, потрібно запустити .exe файл у кореневій папці «Program». Назва файлу – «MusicPlayer.exe» (Рис. 2.10.)

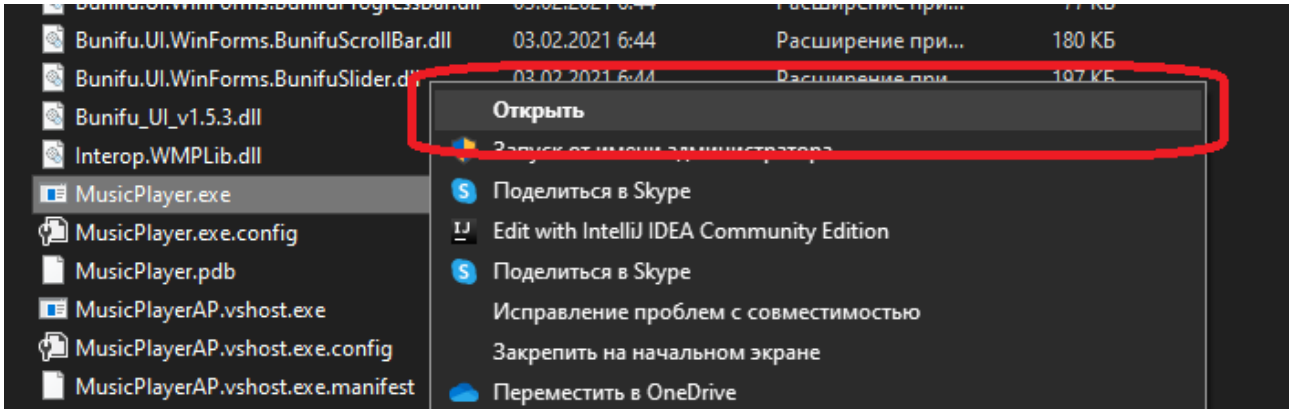

Рис. 2.10. Приклад запуску програми

2.6.4. Опис інтерфейсу користувача

Результатом роботи є спроектований та розроблений додаток-аудіоплеєр, для якого також було розроблено унікальний користувацький інтерфейс.

При першому запуску програми, користувача привітає наступне вікно:

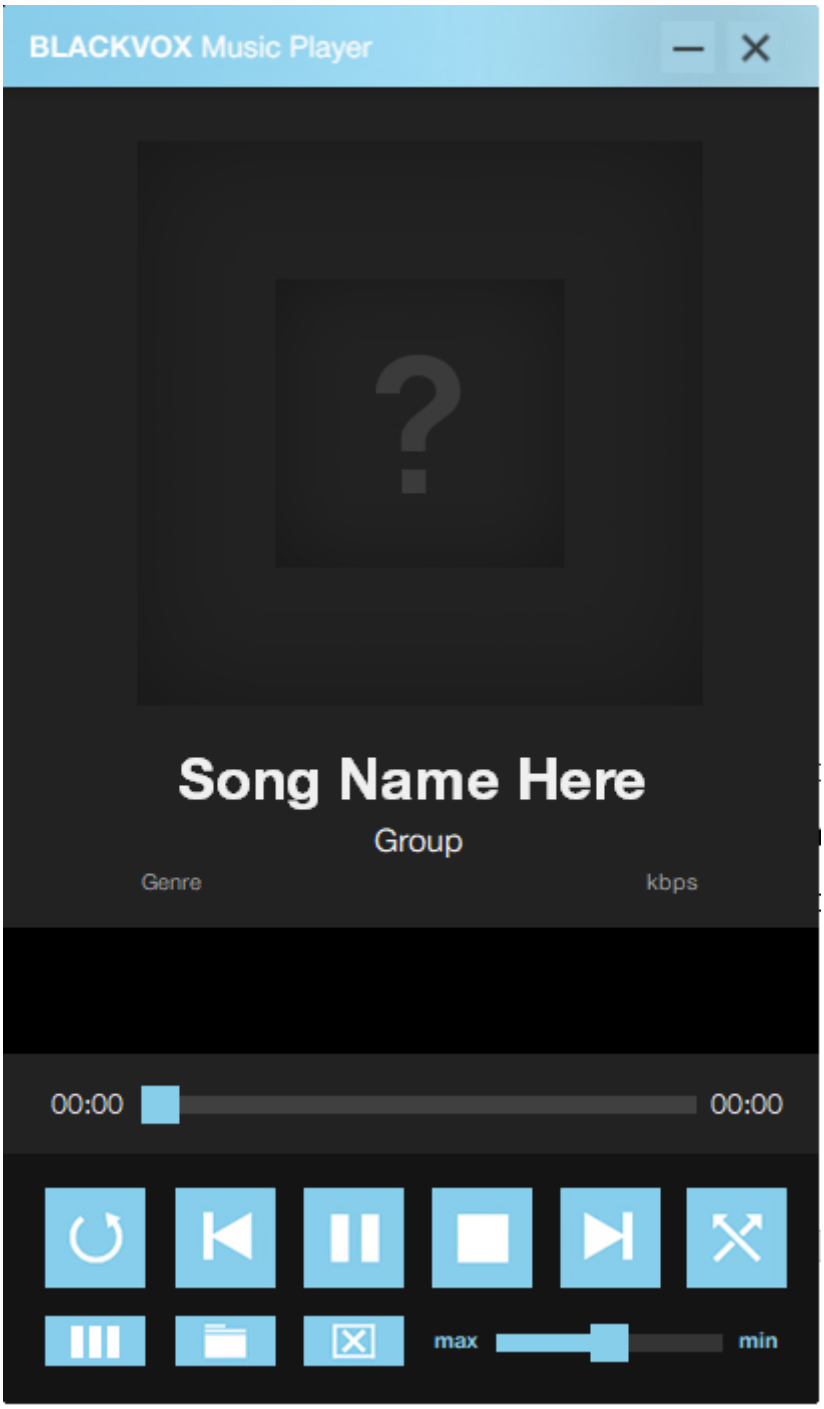

Рис. 2.11. Вигляд інтерфейсу додатку при першому запуску

Щоб завантажити файли до програми, користувачу потрібно натиснути кнопку «Обрати файли» (Рис. 2.12.) та через вікно діалогу відкриття файлів обрати потрібні йому треки, після чого натиснути «Відкрити».

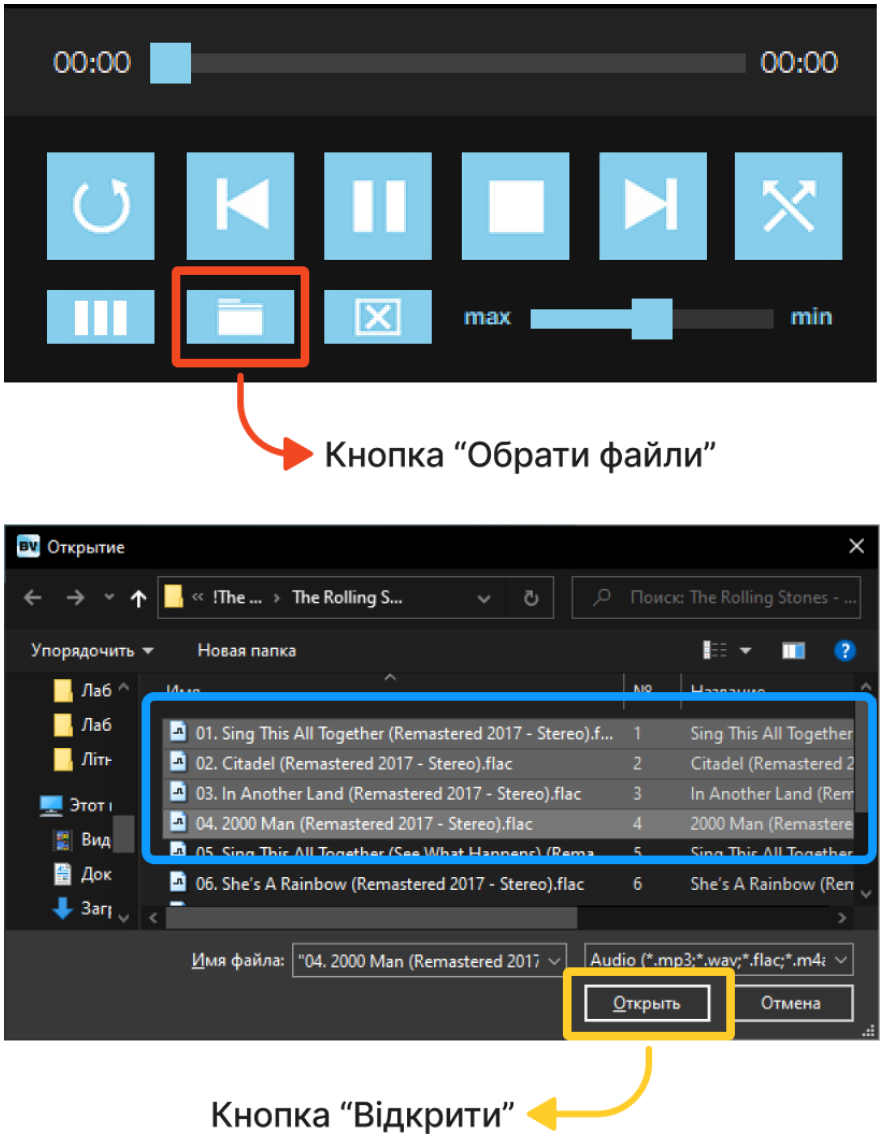

Рис. 2.12. Приклад роботи з алгоритмом додавання файлів

Після пророблених кроків, треки з’являться у вікні плейлісту, і перший обраний при додаванні буде автоматично відтворено.

Альтернативно, у додатку реалізовано функцію «Drag & Drop», яка дозволяє користувачу перетаскувати файли прямо у вікно програми. Для використання цієї функції, потрібно натиснути на кнопку «Відкрити плейліст» на нижній панелі програми (Рис. 2.13.), після чого перетягнути файли у зону

плейлісту, який з’явиться у верхній частині програми. Додані файли з’являться у списку, або у кінці списку, якщо до цього в ньому вже були додані треки. Функція працює як з одним файлом, так і з декількома одночасно.

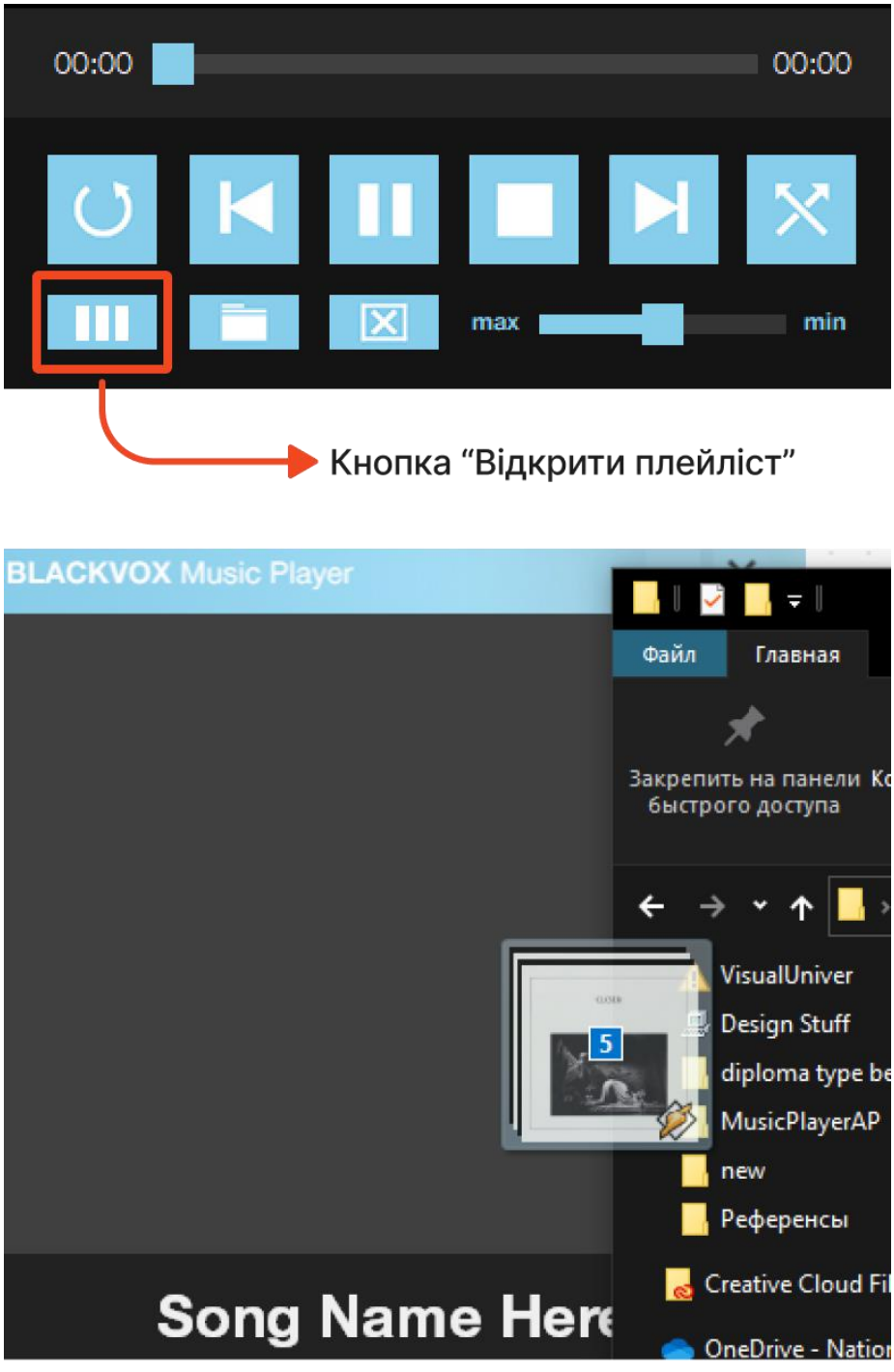

Рис. 2.13. Приклад роботи з алгоритмом додавання файлів «Drag & Drop»

Кнопка «Відкрити плейліст» також функціонує для його приховання. Якщо плейліст знаходиться у відкритому стані, повторне натискання на кнопку тимчасово переведе його у скритий стан. Таке рішення значно впливає на ергономіку програми, та робить її компактнішою, ніж додавання окремого вікна зі списком треків, що завантажені в програму.

Якщо потрібно видалити трек зі списку програвання (плейліста), то треба виділити трек у списку натиснувши ЛКМ і потім натиснути на кнопку «Видалити трек». Трек буде видалено зі списку, але він залишиться на дисковому носії користувача.

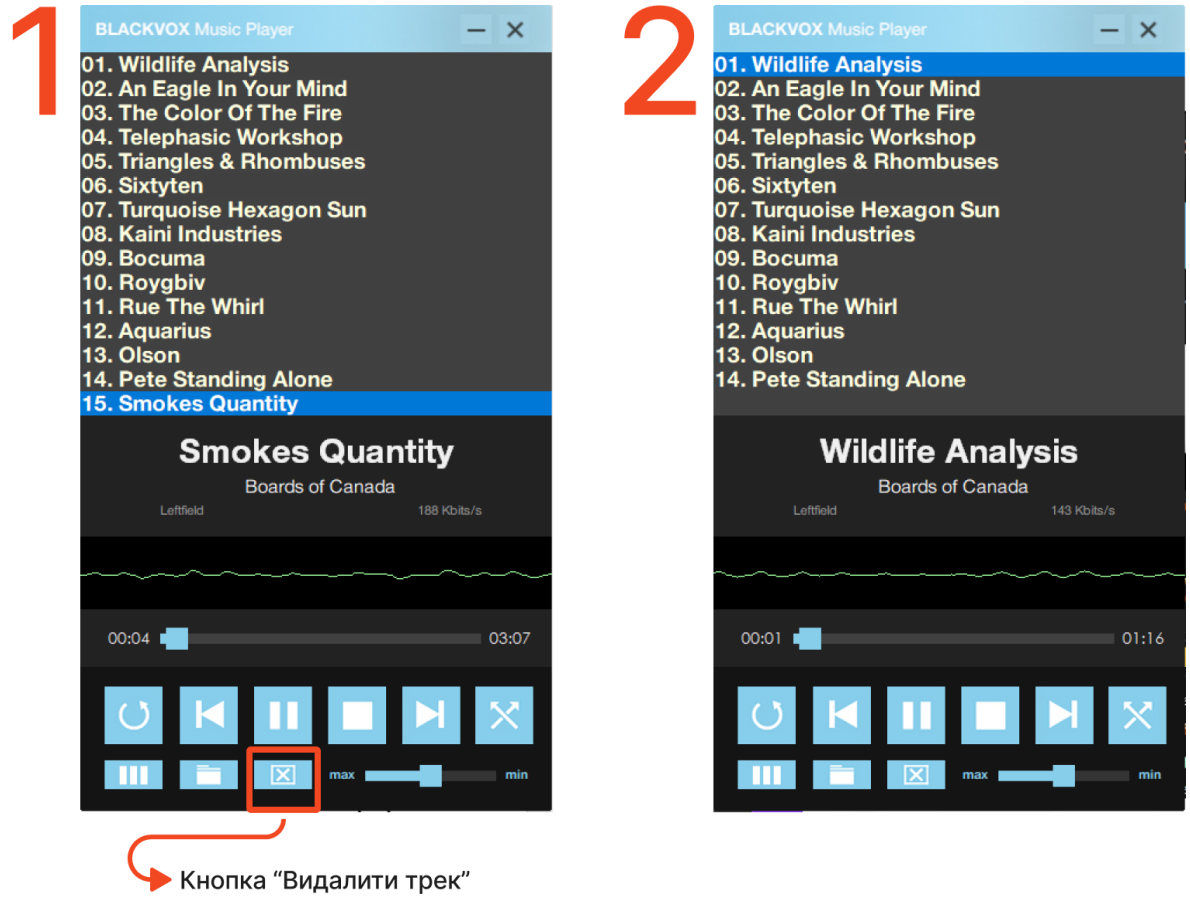

Рис. 2.14. Алгоритм роботи видалення треку

Для навігації по доданим композиціям, у додатку реалізовані усі базові навігаційні функції додатку-плеєра, та декілька додаткових можливостей. Основні функції:

- кнопка «Програвання/Пауза» (1);

- кнопка «Попередній трек» (2);
- кнопка «Наступний трек» (3);
- кнопка «Стоп» (4).

Додаткові функції:

- кнопка «Зациклювання треку» (5);
- кнопка «Перемішати програвання» (6);
- кнопка «Відкрити/Закрити плейліст» (7);
- кнопка «Обрати файли» (8);
- кнопка «Видалити трек» (9).

Рис. 2.15. Навігаційні та функціональні кнопки

Кнопки додаткової функціональності «Зациклювання треку» та «Перемішати програвання» містять у собі дві функції. Одне натискання на кнопку переводить функцію у активований стан та міняє колір кнопки з синього на зелений, звітуючи про активацію функції. Повторне натискання, відповідно, переводить функцію у вимкнений стан.

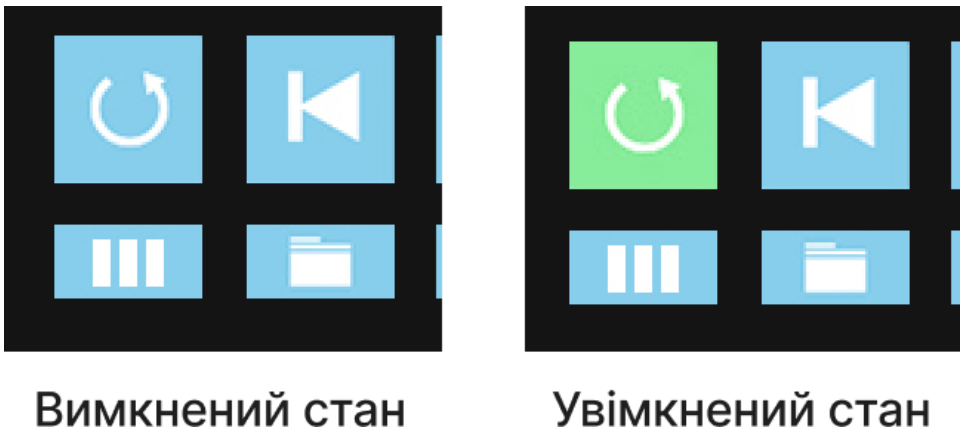

Рис. 2.16. Приклад роботи кнопки з двома станами

Для навігації позиції по треку що програвається, в програмі розроблений спеціальний слайдер навігації позиції треку (Рис. 2.17.). Якщо в програму було додано трек і він знаходиться у стані програвання, час від його початку буде зображений зліва від навігаційного слайдеру (у хвилинах). Загальна довжина треку зображена справа від навігаційного слайдеру. Якщо користувач бажає переключитися у бажане місце треку, він можете зробити це натисканням ЛКМ по навігаційному показнику слайдера, та, не відпускаючи, може перетягнути його у бажане місце.

Альтернативно, перейти до потрібної позиції у треці можна натисканням ЛКМ на потрібну позицію слайдера, після чого показник автоматично переміститься у потрібну позицію.

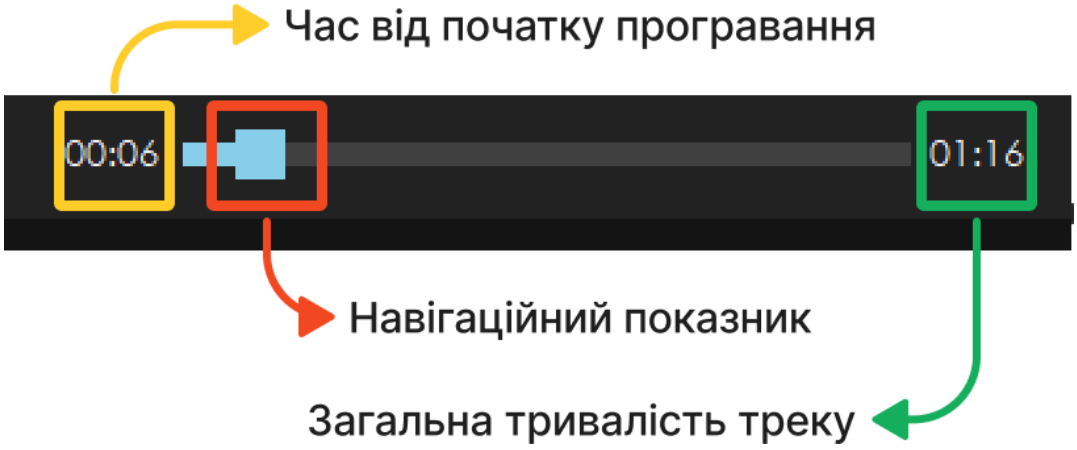

Рис. 2.17. Будова слайдеру позиції треку

Слайдер гучності працює за схемою, що є аналогічною до схеми функціонування слайдеру позиції треку. Перетаскування слайдеру у ліву позицію буде зменшувати гучність треку. Перетаскування слайдера у праву позицію, відповідно, буде збільшувати гучність треку.

Рис. 2.18. Будова слайдеру гучності треку

Верхня частина плеєру містить інформацію про програваємий трек, яка зчитується з метаданих аудіофайлу. Користувачу, за наявністю цих даних у файлі, буде виводитися наступне:

- графічна обложка треку;
- назва треку;
- автор треку;
- жанр треку;
- бітрейт треку.

Також плеєр має функцію графічного відображення звукових хвиль, або «волюметр». Він відмальовує амплітуди звукової хвилі треку у та представляє їх у візуальному зображенні. «Волюметр» служить як елемент, потрібний як при професійній роботі з аудіофайлами, так і як елемент візуальної естетики.

Нижче представлена будова верхньої частини плеєру.

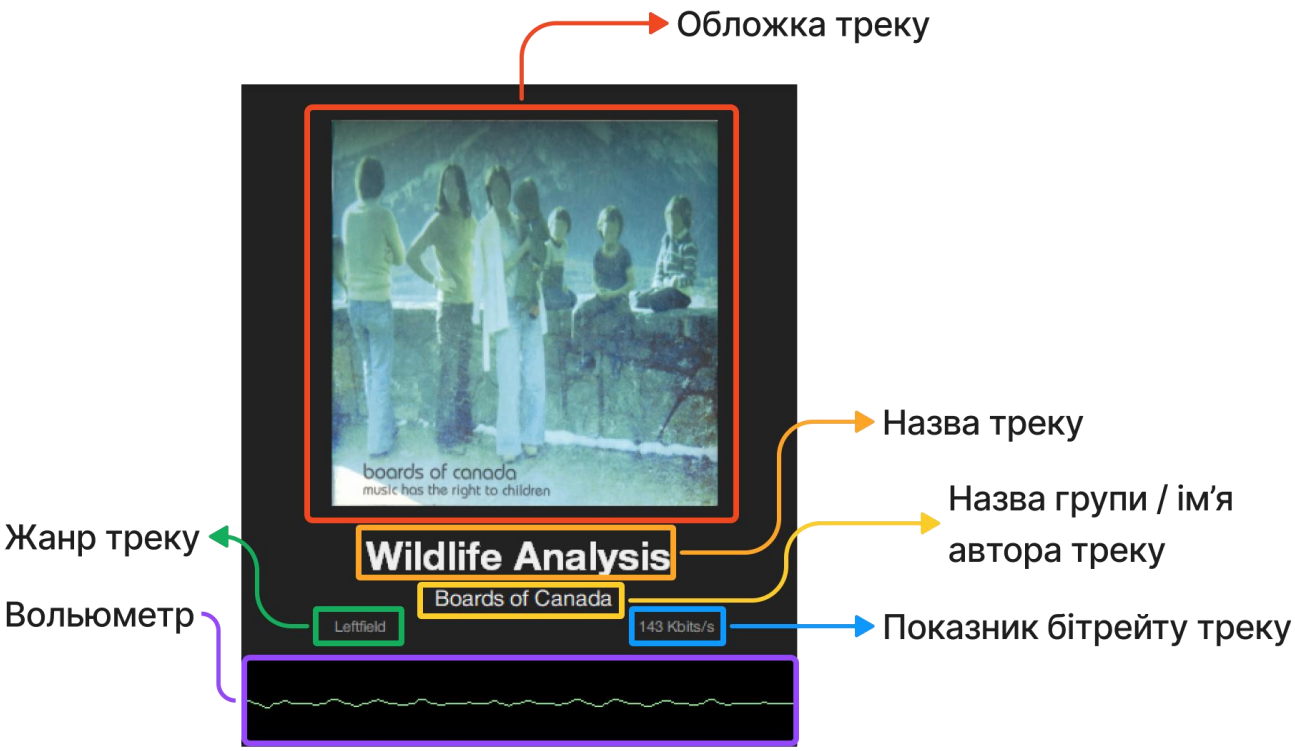

Рис. 2.19. Будова верхньої частини плеєру

Шапка плеєру функціонує як зона, за яку можна перетаскувати додаток по екрану. Для цього треба зажати ЛКМ у любій точці шапки, та переміщати курсор у потрібне місце.

Також, у шапці містяться дві кнопки: «Мінімізувати додаток» та «Зачинити додаток». Вони реалізують відповідні до їх назви функції.

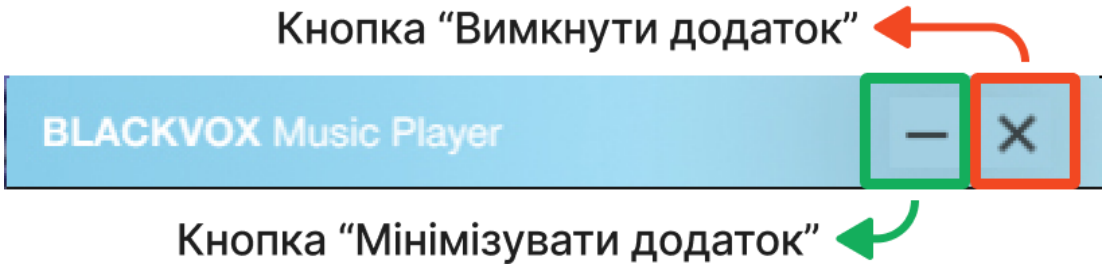

Рис. 2.20. Будова шапки плеєру

РОЗДІЛ 3 ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

1. передбачуване число операторів програми - 1165;
2. коефіцієнт складності програми – 1,3;
3. коефіцієнт корекції програми в ході її розробки – 0,06;
4. годинна заробітна плата програміста – 150 грн/год;
5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;
6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,2;
7. вартість машино-години ЕОМ – 14 грн/год

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \qquad (3.1)$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u– витрати праці на дослідження алгоритму рішення задачі;

t_a– витрати праці на розробку блок-схеми алгоритму;

$t_п$– витрати праці на програмування по готовій блок-схемі;

$t_{отл}$– витрати праці на налагодження програми на ЕОМ;

$t_д$– витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q=q\cdot C\cdot(1+p), \text{ де} \qquad (3.2)$$

q– передбачуване число операторів;

C– коефіцієнт складності програми;

p– коефіцієнт кореляції програми в ході її розробки.

$$Q= 1165\cdot 1,2\cdot(1+0,06) = 1481,88;$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \bullet B}{(75 \dots 85) \bullet K}, \text{ людино-годин}, \qquad (3.3)$$

де *B* – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

K – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

$$t_u = \frac{1481,88 \cdot 1,2}{78 \cdot 1,2} = 18,99, \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25)\cdot k}, \text{ людино-годин.} \qquad (3.4)$$

де *Q* – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.4), людино-годин:

$$t_a = \frac{1481,88}{20 \cdot 1,2} = 61,74, \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)\cdot k}, \text{ людино-годин.} \qquad (3.5)$$

$$t_a = \frac{1481{,}88}{25 \cdot 1{,}2} = 49{,}39, \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

- за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5)\cdot k}, \text{ людино-годин.} \quad (3.6)$$

$$t_n = \frac{1481{,}88}{5 \cdot 1{,}2} = 246{,}98, \text{ людино-годин,}$$

- за умови комплексного налагодження завдання:

$$t^{к}_{отл} = 1{,}2 \cdot t_{отл}; \quad (3.7)$$

$$t^{к}_{отл} = 1{,}2 \cdot 246{,}98 = 296{,}376, \text{ людино-годин}$$

Витрати праці на підготовку документації:

$$t_д = t_{дp} + t_{до}; \quad (3.8)$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису

$$t_д = \frac{Q}{(15...20)\cdot K}; \quad (3.9)$$

$$t_д = \frac{1481{,}88}{18 \cdot 1{,}2} = 68{,}60, \text{ людино-годин.}$$

$t_{до}$– трудомісткість редагування, печатки й оформлення документації

$$t_{до} = 0{,}75 \cdot t_{др}; \quad (3.10)$$

$$t_{до} = 0{,}75 \cdot 68{,}60 = 51{,}45, \text{ людино-годин.}$$

$$t_д = 68{,}60 + 51{,}45 = 112{,}05, \text{ людино-годин.}$$

Отримаємо трудомісткість розробки програмного забезпечення:

t =50+18,99+61,74+246,98+296,376+112,05=768,15, людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 768,15 людино-годин для розробки даного програмного забезпечення.

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \quad (3.11)$$

де $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн,} \quad (3.12)$$

де t – загальна трудомісткість, людино-годин;

$C_{ПР}$– середня годинна заробітна плата програміста, грн/година

$$З_{ЗП} = 768{,}15 \cdot 150 = 115\,222{,}5, \text{ грн.}$$

$З_{МВ}$– Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot C_{М}, \text{ грн,} \quad (3.13)$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год.

$C_{МЧ}$– вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 768{,}15 \cdot 14 = 10\,745{,}1, \text{ грн.}$$

$$К_{ПО} = 115222{,}5 + 10745{,}1 = 125\,976{,}6, \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес.} \quad (3.14)$$

де B_k- число виконавців;

$F_р$– місячний фонд робочого часу (при 40 годинному робочому тижні Fр=176 годин).

$$T = \frac{786{,}15}{1 \cdot 176} = 4{,}46 \text{ міс.}$$

Висновки. На розробку даного програмного забезпечення піде 786,15 людино-годин. Тобто, ймовірна очікувана тривалість розробки складатиме 4,46 місяці при стандартному 40-годинному робочому тижні і 176-годинному робочому місяці. Очікувані витрати на створення програмного забезпечення складатимуть 125 976,6 грн.

ВИСНОВКИ

В рамках даної кваліфікаційної роботи було спроектовано та розроблено додаток-аудіоплеєр на мові програмування C#, з застосуванням додаткового фреймворку Bunifu UI.

Призначенням розробленого програмного забезпечення є надання користувачеві комфортного та функціонального додатку, який дозволяє прослуховувати файли усіх найпопулярніших аудіоформатів. Також додаток надасть можливість візуалізації звукової хвилі програваємого треку, та автоматично збереже зібраний плейліст після завершення своєї роботи.

Під час виконання даного кваліфікаційної роботи були виконані наступні задачі:

- проаналізовано предметну область задачі, що розв’язується;
- з’ясовані вимоги та підстави для розробки додатку даної кваліфікаційної роботи;
- проаналізовано можливості та стилі інтерфейсу подібних застосунків, що є представлиним на ринку;
- обрано раціональну структуру і параметри програми;
- створено користувацький інтерфейс додатку;
- написано програмний код додатку;
- розроблено рекомендації щодо використання програми.

Додаток написано у інтегрованому середовищі розробки Microsoft Visual Studio на високорівневій мові кодування C#. Графічний інтерфейс додатку був реалізований за допогою технології Windows Forms та стороннього фреймворку Bunifu UI. Застосування цих технологій дозволило досянути добрих результатів в оптимізації роботи програми, та надання їй сучасного та функціонального інтерфейсу, що значно впливає на загальний досвід користування (UX). Додаток призначенний для керуваня під операційною системою Windows.

На завершення, у «Економічному розділі» кваліфікаційної роботи було визначено трудомісткість розробленого програмного продукту (786,15 люд-год), проведений підрахунок вартості роботи по створенню програми (125976,6) грн. та розраховано час на його створення (4,46 міс)

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development: Build applications with C# / Mark J. Price – 818 p.
2. Head First C#: A Learner's Guide to Real-World Programming with C# and .NET Core 4th Edition / Andrew Stellman – 800 p.
3. C#: Learn C# in One Day and Learn It Well. C# for Beginners with Hands-on Project. / Jamie Chan – 161 p.
4. Starting out with Visual C# 5th Edition / Tony Gaddis – 960 p.
5. The C# Programming Yellow Book / Rob Miles – 224 p.
6. Visual C# How to Program (Deitel Series) 6th Edition / Paul Deitel – 1056p.
7. C# in Depth: Fourth Edition / Jon Skeet – 526 p.
8. Programming C# 8.0: Build Windows, Web, and Desktop Applications / Ian Griffiths – 779 p.
9. Murach's C# / Anne Boehm – 914 p.
10. Pro C# 7: With .NET and .NET Core / Andrew Troelsen – 1437 p.
11. MICROSOFT VISUAL C#. ПОДРОБНОЕ РУКОВОДСТВО / Джон Шарп – 848 с.
12. .NET Framework Essentials (O'Reilly Programming Series) / Thuan L. Thai – 320 p.
13. Mastering C# and .NET Framework / Marino Posadas – 560 p.
14. Язык программирования C# 6.0 и платформа .NET 4.6 7-е изд. / Эндрю Троелсен, Филипп Джепикс – 1440 с.
15. Visualstudio .Net: The .Net Framework Black Book / Julian Templeman – 816 p.
16. Entity Framework Core in Action / Jon P Smith – 520 p.
17. Professional C# and .NET, 2021 Edition / Christian Nagel – 1008 p.
18. Programming C# 5.0: Building Windows 8, Web, and Desktop Applications for the .NET 4.5 Framework 1st Edition / Ian Griffiths – 886 p.

19. Программирование на C# для начинающих. Основные сведения / Алексей Васильев – 592 с.

20. C#. Практическое руководство / Евдокимов П.В. – 416 с.

21. C# для чайников / Билл Семпф, Джон Пол Мюллер, Чак Сфер – 608 с.

22. TagLib API Documentation [Електронний ресурс]. Режим доступу: https://taglib.org/api/index.html

23. A COMPARISON OF MODEL VIEW CONTROLLER AND MODEL VIEW PRESENTER. [Електронний ресурс]. Режим доступу: https://arxiv.org/ftp/arxiv/papers/1408/1408.5786.pdf

24. Bunifu UI .Net Framework Tutorial 5 [Електронний ресурс]. Режим доступу: https://www.youtube.com/watch?v=PfAJNXimP_s

25. Bunifu UI .Net Framework Tutorial 6 [Електронний ресурс]. Режим доступу: https://www.youtube.com/watch?v=gDMLcaNHMD0

ДОДАТОК А

КОД ПРОГРАМИ

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
using TagLib;

namespace MusicPlayerAP
{
   public partial class Form1 : Form
   {
      // Ініціалізуємо списки імен та путів до них
      List<string> TrackNames = new List<string>();
      List<string> TrackPaths = new List<string>();
      bool next;
      bool repeat = false;
      bool random = false;
      Random rand = new Random();
      int iScroll = -3;
      int overflow = 0;
      string songName = "Song Name Here";
      const string overflowPrefix = " *** ";
      const string sPath = "playlist.txt";
      public static readonly string[] allowedExtentions = {".mp3", ".wav", ".flac", ".m4a", ".aac",
".ogg", ".aif", ".aiff", ".midi"};
      public Form1()
      {
         InitializeComponent();
         System.Windows.Forms.Control.CheckForIllegalCrossThreadCalls = false;

         // Збереження та завантаження плейлісту

         string line;
         try
         {
            System.IO.StreamReader Load = new System.IO.StreamReader(sPath);
            while ((line = Load.ReadLine()) != null)
            {
               try
               {
                  if(System.IO.File.Exists(line)) {
                     string fileName = Path.GetFileNameWithoutExtension(line);
```

```
                TrackNames.Add(fileName);
                Tracks.Items.Add(fileName);
                TrackPaths.Add(line);
            }
        } catch { }
    }
    Load.Close();
    System.IO.StreamWriter Save = new System.IO.StreamWriter(sPath);
    foreach (var item in TrackPaths)
    {
        Save.WriteLine(item);
    }
    Save.Close();
  } catch { }
}

// Функція переходу на наступний трек у списку
private void NextTrack()
{
  if(TrackPaths.Count != 0)
  {
    if (Tracks.SelectedIndex == TrackPaths.Count - 1)
    {
      if (random)
      {
        axWindowsMediaPlayer1.URL = TrackPaths[0];
        Tracks.SelectedIndex = 0;
      } else
      {
        // Якщо трек є останнім, вимикаємо програвання та обнуляємо показники
        axWindowsMediaPlayer1.URL = TrackPaths[0];
        Tracks.SelectedIndex = 0;
        axWindowsMediaPlayer1.Ctlcontrols.stop();
        timer1.Stop();
        label1.Text = "Song Name Here";
        label2.Text = "Group";
        label3.Text = "00:00";
        label4.Text = "00:00";
        label5.Text = "Genre";
        label6.Text = "kbps";
        bunifuSlider1.Value = 0;
        albumCover.Image = null;
        play_pauseButton.Image = MusicPlayer.Properties.Resources.play;
      }
    }
    else
    {
      // Перевірка на те, чи ввімкнена функція довільного порядку терків
      if(random)
      {
        int rnd;
        do
```

```
                {
                  rnd = rand.Next(TrackPaths.Count - 1);
                } while (rnd == Tracks.SelectedIndex - 1);
                axWindowsMediaPlayer1.URL = TrackPaths[rnd];
                Tracks.SelectedIndex = rnd;
            }
            // Інакше вмикається наступній трек
            axWindowsMediaPlayer1.URL = TrackPaths[Tracks.SelectedIndex + 1];
            Tracks.SelectedIndex += 1;
        }
    }
}

// Функція переходу на попередній трек у списку
private void PreviousTrack()
{
    if (TrackPaths.Count != 0)
    {
        if (Tracks.SelectedIndex == 0)
        {
            // Якщо трек є першим, то вмикаємо останній трек
            axWindowsMediaPlayer1.URL = TrackPaths[0];
            Tracks.SelectedIndex = TrackPaths.Count - 1;
        }
        else
        {
            // Інакше, вмикаємо попередній
            axWindowsMediaPlayer1.URL = TrackPaths[Tracks.SelectedIndex - 1];
            Tracks.SelectedIndex -= 1;
        }
        play_pauseButton.Image = MusicPlayer.Properties.Resources.pause;
    }
}

// Пересування слайдеру відповідно до позиції у треку
private void bunifuSlider1_ValueChanged(object sender, EventArgs e)
{
    if (axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.wmppsPlaying)
    {
        // Трек на паузі
        axWindowsMediaPlayer1.Ctlcontrols.pause();
        // Переходимо на потрібну нам точку у треці
        double position = axWindowsMediaPlayer1.currentMedia.duration / ((double)100 /
(bunifuSlider1.Value + 1));
        axWindowsMediaPlayer1.Ctlcontrols.currentPosition = position;
        // Щоб уникнути звуків "хрускоту" при перемотуванні, зупиняємо виконання
програми на 50 мс
        System.Threading.Thread.Sleep(50);
        // Продовжуємо програвання треку
        axWindowsMediaPlayer1.Ctlcontrols.play();
    }
    else
```

```
        {
            // Якщо не програвається жодний трек, виключаємо спроможність пересувати
слайдер
            if (axWindowsMediaPlayer1.currentMedia != null) {
                double position = axWindowsMediaPlayer1.currentMedia.duration / ((double)100 /
(bunifuSlider1.Value + 1));
                axWindowsMediaPlayer1.Ctlcontrols.currentPosition = position;
            } else {
            bunifuSlider1.Value = 0;
            }
        }
    }
    private void Tracks_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (Tracks.SelectedIndex != -1)
        {
            // При виборі треку у списку, включаємо цей трек
            axWindowsMediaPlayer1.URL = TrackPaths[Tracks.SelectedIndex];
            play_pauseButton.Image = MusicPlayer.Properties.Resources.pause;

        }
    }
    // Збереження плейлісту та вихід з програми
    private void bunifuFlatButton1_Click(object sender, EventArgs e)
    {
        System.IO.StreamWriter Save = new System.IO.StreamWriter(sPath);
        foreach (var item in TrackPaths)
        {
            Save.WriteLine(item);
        }
        Save.Close();
        Application.Exit();
    }

    // Обирання файлів
    private void bunifuImageButton2_Click(object sender, EventArgs e)
    {
        if(openFileDialog1.ShowDialog() != DialogResult.Cancel)
        {
            // З кожного вибраного файлу дістаємо його назву, заносимо до списку
            foreach (String file in openFileDialog1.FileNames)
            {
                string fileName = Path.GetFileNameWithoutExtension(file);
                TrackNames.Add(fileName);
                Tracks.Items.Add(fileName);
                TrackPaths.Add(file);
            }
            if (axWindowsMediaPlayer1.playState != WMPLib.WMPPlayState.wmppsPlaying)
            {
                // Вибираємо перший трек і вмикаємо його
                axWindowsMediaPlayer1.URL = openFileDialog1.FileName;
                axWindowsMediaPlayer1.Ctlcontrols.play();
```

```
            play_pauseButton.Image = MusicPlayer.Properties.Resources.pause;
        }
        // Вибираємо перший за списком трек
        if (Tracks.SelectedIndex == -1)
        {
            Tracks.SelectedIndex = 0;
        }
    }
}

// Кнопка "Play" та "Pause"
private void play_pauseButton_Click(object sender, EventArgs e)
{
    if (axWindowsMediaPlayer1.URL != "")
    {
        // Увімкнення треку
        if (axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.wmppsPaused)
        {
            axWindowsMediaPlayer1.Ctlcontrols.play();
            // Приховуємо кнопку "Play", показуємо кнопку "Pause"
            play_pauseButton.Image = MusicPlayer.Properties.Resources.pause;
        }
        else if(axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.wmppsPlaying)
        {
            // Приховуємо кнопку "Pause", показуємо кнопку "Play"
            play_pauseButton.Image = MusicPlayer.Properties.Resources.play;
            axWindowsMediaPlayer1.Ctlcontrols.pause();
        }
        // Увімкнення треку, якщо їх кількість більша за нуль
        else if (axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.wmppsReady)
        {
            if (TrackNames.Count > 0)
            {
                axWindowsMediaPlayer1.Ctlcontrols.play();
                play_pauseButton.Image = MusicPlayer.Properties.Resources.pause;
            }
        }
        else if(axWindowsMediaPlayer1.playState == WMPLib.WMPPlayState.wmppsStopped)
        {
            if(TrackNames.Count>0)
            {
                axWindowsMediaPlayer1.Ctlcontrols.play();
                play_pauseButton.Image = MusicPlayer.Properties.Resources.pause;
            }
        }
    }
}

// Кнопка "Stop"
private void bunifuImageButton4_Click(object sender, EventArgs e)
{
    axWindowsMediaPlayer1.Ctlcontrols.stop();
```

```
            play_pauseButton.Image = MusicPlayer.Properties.Resources.play;
        }

        // Слайдер налаштування гучності
        private void bunifuSlider2_ValueChanged(object sender, EventArgs e)
        {
            axWindowsMediaPlayer1.settings.volume = bunifuSlider2.Value;
        }

        // Виводження назви, автора та тривалості треку
        private void axWindowsMediaPlayer1_PlayStateChange(object sender,
AxWMPLib._WMPOCXEvents_PlayStateChangeEvent e)
        {
            if (e.newState == 3)
            {
                if (axWindowsMediaPlayer1.currentMedia != null)
                {
                    var file = TagLib.File.Create(axWindowsMediaPlayer1.currentMedia.sourceURL);

                    // Початок відліку таймера, який відповідає за тимчасову позицію в треку
                    timer1.Start();
                    // Привласнення довжини треку
                    label3.Text = axWindowsMediaPlayer1.currentMedia.durationString;

                    // Виведення назви треку
                    if (file.Tag.Title != null)
                    {
                        if(file.Tag.Title.Length >= 19)
                        {
                            label1.Text = file.Tag.Title.Substring(0, 19);
                        } else {
                            label1.Text = file.Tag.Title;
                        }
                        songName = file.Tag.Title;
                        overflow = 0;
                        iScroll = -3;
                    } else
                    {
                        if (axWindowsMediaPlayer1.currentMedia.name.Length >= 19)
                        {
                            label1.Text = axWindowsMediaPlayer1.currentMedia.name.Substring(0, 19);
                        } else {
                            label1.Text = axWindowsMediaPlayer1.currentMedia.name;
                        }
                        songName = axWindowsMediaPlayer1.currentMedia.name;
                        overflow = 0;
                        iScroll = -3;
                    }

                    // Виведення назви виконавця треку
                    label2.Text = file.Tag.FirstPerformer;
```

```
            // Виведення жанру треку
            label5.Text = file.Tag.FirstGenre;

            // Виведення значення бітрейта треку
            label6.Text = file.Properties.AudioBitrate.ToString() + " Kbits/s";

            // Отримання зображення обкладинки треку
            if (file.Tag.Pictures.Length >= 1)
            {
               var bin = (byte[])(file.Tag.Pictures[0].Data.Data);
               albumCover.Image = Image.FromStream(new
MemoryStream(bin)).GetThumbnailImage(282, 282, null, IntPtr.Zero);
            }
            else
            {
               albumCover.Image = null;
            }
         }
      }
      else if (e.newState == 8)
      {
         // Після закінчення треку дозволяємо перехід на наступний трек
         next = true;
      }
      else if (e.newState == 9)
      {
         // Якщо перехід дозволено – увімкнути наступний трек
         if (next)
         {
            next = false;
            if (repeat)
            {
               axWindowsMediaPlayer1.URL = TrackPaths[Tracks.SelectedIndex];
            }
            else
            {
               NextTrack();
            }
         }
      }

      // Обнулення всіх інформаційних вікон під час зупинки треку
      else if (e.newState == 1)
      {
         timer1.Stop();
         label1.Text = "Song Name Here";
         label2.Text = "Group";
         label3.Text = "00:00";
         label4.Text = "00:00";
         label5.Text = "Genre";
         label6.Text = "kbps";
         bunifuSlider1.Value = 0;
```

```
            songName = label1.Text;
            overflow = 0;
            iScroll = -3;
            albumCover.Image = null;
        }
    }

    // Таймер переміщення позиції треку
    private void timer1_Tick(object sender, EventArgs e)
    {
        if (axWindowsMediaPlayer1.Ctlcontrols != null)
        {
            if (songName.Length > 19)
            {
                iScroll++;
                if(iScroll >= 0)
                {
                    int iLmt = songName.Length - iScroll;
                    if (iLmt < 19)
                    {
                      overflow++;
                      if (overflow <= overflowPrefix.Length)
                      {
                          string str = songName + overflowPrefix.Substring(0, overflow);
                          str = str.Substring(iScroll, 19);
                          label1.Text = str;
                      } else {

                          string str = songName + overflowPrefix + songName.Substring(0, overflow -
overflowPrefix.Length);
                          str = str.Substring(iScroll, 19);
                          label1.Text = str;
                          if(iScroll >= songName.Length + overflowPrefix.Length)
                          {
                             iScroll =-3;
                             overflow = 0;
                          }
                       }

                    } else {
                      string str = songName.Substring(iScroll, 19);
                      label1.Text = str;
                      overflow = 0;
                    }
                }
            }

            // Розраховуємо позицію треку за формулою
            double position = 1 / (axWindowsMediaPlayer1.currentMedia.duration /
axWindowsMediaPlayer1.Ctlcontrols.currentPosition);
            position *= 100;
            int pos = (int)position;
```

```
        if (pos < 0)
        {
            pos = 0;
        }
        if (axWindowsMediaPlayer1.Ctlcontrols.currentPositionString == "")
        {
            // Якщо позиція відсутня, обнуляємо її
            label4.Text = "00:00";
        }
        else
        {
            // Інакше виставляємо поточну позицію
            label4.Text = axWindowsMediaPlayer1.Ctlcontrols.currentPositionString;
        }
        bunifuSlider1.Value = pos;
    }
}

// Кнопка "Мінімізувати"
private void bunifuFlatButton2_Click(object sender, EventArgs e)
{
    if(WindowState == FormWindowState.Normal)
    {
        WindowState = FormWindowState.Minimized;
    }
}

// Кнопка перемикання на наступний трек
private void bunifuImageButton6_Click(object sender, EventArgs e)
{
    NextTrack();
}

// Кнопка перемикання на попередній трек
private void bunifuImageButton3_Click(object sender, EventArgs e)
{
    PreviousTrack();
}

// Кнопка видалення вибраного треку
private void bunifuImageButton7_Click(object sender, EventArgs e)
{
    if (TrackNames.Count > 0)
    {
        if (Tracks.SelectedIndex == -1) return;
        TrackNames.RemoveAt(Tracks.SelectedIndex);
        TrackPaths.RemoveAt(Tracks.SelectedIndex);
        Tracks.Items.RemoveAt(Tracks.SelectedIndex);
        if (TrackNames.Count > 0)
        {
            NextTrack();
        }
```

```
        else
        {
            axWindowsMediaPlayer1.Ctlcontrols.stop();
        }
    }
}
// Кнопка "Повтор треку"
private void repeat_Click(object sender, EventArgs e)
{
    if (repeat)
    {
        repeat = false;
        repeatButton.Image = MusicPlayer.Properties.Resources.replay;
    } else {
        repeat = true;
        repeatButton.Image = MusicPlayer.Properties.Resources.replayON;
    }
}
// Кнопка "Довільний порядок"
private void shuffle_Click(object sender, EventArgs e)
{

    if (random)
    {
        random = false;
        shuffleButton.Image = MusicPlayer.Properties.Resources.shuffle;
    }
    else
    {
        random = true;
        shuffleButton.Image = MusicPlayer.Properties.Resources.shuffleON;
    }

}

// Функція "Drag & Drop"
private void Tracks_DragEnter(object sender, DragEventArgs e)
{
    e.Effect = DragDropEffects.All;
}
private void Tracks_DragDrop(object sender, DragEventArgs e)
{
    string[] DroppedFiles = (string[])e.Data.GetData(DataFormats.FileDrop, false);

    foreach (String file in DroppedFiles)
    {
        string fileName = Path.GetFileNameWithoutExtension(file);
        string fileExtention = Path.GetExtension(file);
        if (allowedExtentions.Contains(fileExtention))
        {
            TrackNames.Add(fileName);
            Tracks.Items.Add(fileName);
```

```
                TrackPaths.Add(file);
            }
        }
        if (axWindowsMediaPlayer1.playState != WMPLib.WMPPlayState.wmppsPlaying)
        {
            foreach (String file in DroppedFiles)
            {
                string fileExtention = Path.GetExtension(file);
                if (allowedExtentions.Contains(fileExtention))
                {
                    axWindowsMediaPlayer1.URL = DroppedFiles[0];
                    axWindowsMediaPlayer1.Ctlcontrols.play();
                    play_pauseButton.Image = MusicPlayer.Properties.Resources.pause;
                    break;
                }
            }
        }
        if (Tracks.SelectedIndex == -1 && Tracks.Items.Count > 0)
        {
            Tracks.SelectedIndex = 0;
        }
    }
    // Кнопка "Плейліст"
    private void playlist_Click(object sender, EventArgs e)
    {
        if (Tracks.Visible == false)
        {
            Tracks.Visible = true;
        } else
        {
            Tracks.Visible = false;
        }
    }
  }
}
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

Перелік файлів на диску

Перелік документів на магнітному носії

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Мірошник.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Мірошник.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Мірошник.rar	Архів. Містить коди програми і скомпільовану програму
Презентація	
Мірошник.ppt	Презентація кваліфікаційної роботи

ВІДГУК

на кваліфікаційну роботу бакалавра
на тему:
"Розробка додатку «аудіо-плеєр» на мові програмування C#"
студента групи 122-18-3 Мірошника Микити Вадимовича

Розроблений в кваліфікаційній роботі додаток призначений для відтворення файлів усіх популярних аудіоформатів задля розваги користувача, або для використання у професійних цілях.

Актуальність розробленого програмного продукту полягає в створенні такого додатку, що комбінує в собі функціональність інших додатків подобного типу, представлених на ринку, та зручний і сучасний дизайн інтерфейсу користувача.

Програмне забезпечення реалізовано на високорівневій мові програмування C#. Основним фреймворком для реалізації користувацького інтерфейсу було обрано фреймворк Bunifu UI, побудований над інтерфейсом програмування Windows Forms.

Додаток було розроблено у інтегрованому середовищі розробки Microsoft Visual Studio 2019.

Працездатність представленої програми підтверджена налагоджувальними випробуваннями та тестуванням програми.

Для вирішення поставлених задач при розробці додатку були здійснені наступні дії:

1. Дослідження додатків-конкурентів.
2. Розробка логічної моделі програми.
3. Розробка графічного дизайну інтерфейсу користувача.
4. Проектування та розробка програмного забезпечення.

Практична значимість створення даного програмного продукту полягає в наданні користувачу сучасного та простого в користуванні аудіоплеєра

У економічному розділі визначено трудомісткість розробленого додатку, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Тема кваліфікаційної роботи безпосередньо пов’язана з об’єктом діяльності бакалавра за напрямом підготовки 122 Комп’ютерні науки.

Оформлення пояснювальної записки до кваліфікаційної роботи виконано відповідно до стандартів на програмну документацію.

Кваліфікаційна робота була виконана самостійно та заслуговує оцінки «добре», а студенту Мірошнику Микиті Вадимовичу присвоєння кваліфікації бакалавра за спеціальністю «фахівець з розробки та тестування програмного забезпечення».

Керівник кваліфікаційної роботи

ас. каф. ПЗКС, к.т.н. **Приходченко С.Д**

РЕЦЕНЗІЯ
на кваліфікаційну роботу бакалавра
на тему:
"Розробка додатку «аудіо-плеєр» на мові програмування C#"
студента групи 122-18-3 Мірошника Микити Вадимовича

Кваліфікаційна роботана тему «Розробка додатку «аудіо-плеєр» на мові програмування C#» виконаний в повному обсязі, відповідно до технічного завдання.

Мета кваліфікаційної роботи: написання додатку, який надає користувачу аудіо-плєер з можливістю програвання усіх найпопулярніших аудіоформатів, та комбінує в собі унікальний та практичний дизайн.

У пояснювальній записці розглянуто необхідність створення і сфера застосування розробленої, виконано постановку завдання, опис вхідних і вихідних даних, розроблено інформаційне забезпечення системи, наведені загальні відомості про додаток та його функціональне призначення, зазначені використовувані технічні засоби, визначені джерела, використані при розробці.

Для реалізації інформаційної системи була використана мова програмування C#. Інтерфейс було розроблено за допомогою фреймворку Bunifu UI.

Вважаю завдання і зміст кваліфікаційної роботи відповідним для перевірки ступеня підготовленості Мірошника М.В. за напрямом 122 «Комп'ютерні науки».

Список літератури, наведений в роботі, налічує більше 20 джерел, що свідчить про вміння автора працювати з літературою та іншими джерелами інформації. Якість оформлення кваліфікаційної роботи можна визнати задовільною, присутня достатня кількість малюнків. В роботі є заключні висновки. Працездатність додатка підтверджується випробуваннями та тестуваннями.

Кваліфікаційна робота виконана самостійно та заслуговує оцінки «добре», а студент Мірошник Микита Вадимович заслуговує присвоєння йому кваліфікації бакалавра за спеціальністю «фахівець з розробки та тестування програмного забезпечення».

Рецензент кваліфікаційної роботи
к.т.н., доцент каф. БІТ **Герасіна О.**