

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

ДЛЯ ВІДТВОРЕННЯ ВІДЕОФАЙЛІВ

SOFTWARE FOR VIDEO PLAYBACK

Виконав(ла): студент(ка) 4 курсу, групи КНТ-141

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ІВАНІН Д.М.

(ПРИЗВИЩЕ та ініціали)

Керівник КОЦУР М.І.

(ПРИЗВИЩЕ та ініціали)

Рецензент ІЛЬЯШЕНКО М.Б.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ІВАНІНА Дениса Максимовича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення для відтворення відеофайлів.
Software for Video Playback

керівник проєкту (роботи) к.т.н., доцент, КОЦУР Михайло Ігорович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 03 червня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	КОЦУР М.І., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання “ 28 ” квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка структури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділи 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Денис ІВАНІН
(підпис) (Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Михайло КОЦУР
(підпис) (Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
95 с., 3 табл., 22 рис., 3 дод., 18 джерел.

C++, QT CREATOR, ВІДЕОФАЙЛ, МОВА ПРОГРАМУВАННЯ,
ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, СЕРЕДОВИЩЕ РОЗРОБКИ.

Об'єкт дослідження – процес розробки програмного забезпечення для відтворення відеофайлів.

Предмет дослідження – програмні засоби для підтримки процесу відтворення відеофайлів.

Мета роботи – розробка програмного забезпечення для відтворення відеофайлів.

Матеріали, методи та технічні засоби: мова програмування C++, середовище розробки Qt Creator.

Результати. Розроблено проєктні рішення для створення програмного забезпечення для підтримки процесу відтворення відеофайлів. Створено програмне забезпечення для підтримки відтворення відеофайлів. Проведено тестування розробленої програмної системи для підтримки процесу відтворення відеофайлів.

Висновки. Мету роботи досягнуто. Розроблено програмне забезпечення для підтримки процесу відтворення відеофайлів за допомогою мови програмування C++ та середовища розробки Qt Creator.

Галузь використання – відтворення відеофайлів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 95 pages, 3 tables, 22 figures, 3 appendixes, 18 sources.

C++, QT CREATOR, VIDEO FILE, PROGRAMMING LANGUAGE, SOFTWARE, DEVELOPMENT ENVIRONMENT.

The object of research is the process of developing software for working with video files.

The subject of the research is software for supporting the process of playing video files.

The purpose of this work is to develop the software for playing video files.

Materials, methods and technical tools: C++ programming language, Qt Creator development environment.

Results. Project solutions for the creation of software to support the process of playing video files has been designed. The software to support the process of playing video files has been developed. The developed software system to support the process of playing video files has been tested.

Conclusions. The goal of the work has been achieved. The software tools to support the process of playing video files using the C++ programming language and the Qt Creator development environment have been developed.

Scope of use – playing video files.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок	8
Вступ	9
1 Аналіз предметної області	11
1.1 Програмне забезпечення для відтворення відеофайлів	11
1.2 Аналіз програмного забезпечення для відтворення відеофайлів	22
1.3 Висновки за розділом 1	30
2 Матеріали і методи	32
2.1 Вибір мови програмування.....	32
2.2 Вибір середовища розробки для створення програмного забезпечення для відтворення відеофайлів	35
2.3 Висновки за розділом 2	38
3 Опис програми	39
3.1 Структура програмного забезпечення для відтворення відеофайлів	39
3.2 Функціонування програмного забезпечення для відтворення відеофайлів	41
3.3 Опис основних алгоритмів програмного забезпечення для відтворення відеофайлів	46
3.4 Проектування інтерфейсу програмного забезпечення для відтворення відеофайлів	51
3.5 Висновки за розділом 3	53
4 Експлуатація, тестування та експериментальне дослідження програми.....	54
4.1 Призначення й умови застосування програми	54
4.2 Характеристики програми для відтворення відеофайлів.....	55
4.3 Інструкція по експлуатації програми.....	56
4.3.1 Звернення до програми.....	56
4.3.2 Вхідні й вихідні дані.....	56
4.3.3 Повідомлення.....	57
4.4 Виконання програмного забезпечення для відтворення відеофайлів	57

4.5 Тестування програмного забезпечення для відтворення відеофайлів	59
4.6 Висновки за розділом 4	59
Висновки.....	60
Перелік джерел посилань.....	63
Додаток А Технічне завдання.....	65
Додаток Б Фрагмент тексту програми	69
Додаток В Слайди презентації	89

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

PTS – presentation time stamp;

БД – база даних;

ВВ – відтворення відеофайлів;

ПЗ – програмне забезпечення.

ВСТУП

Актуальність програмного забезпечення для відтворення відеофайлів зумовлена значним зростанням споживання мультимедійного контенту, що включає не лише розважальні матеріали, а й освітні, бізнесові та наукові відео. Через це потреба у стабільних і універсальних рішеннях для перегляду відео стає дедалі більш важливою [1], [2].

Сьогодні існує велика кількість відеоформатів, а також різноманітні кодеки, тому програмне забезпечення має підтримувати ці формати, аби забезпечити сумісність і комфорт користувачів. Крім того, відео переглядають на різних пристроях, тому важливо, щоб відповідне програмне забезпечення могло працювати на різних операційних системах [3]-[5].

Безкоштовні та відкриті рішення користуються особливою популярністю, оскільки вони дозволяють не лише безкоштовно дивитися відео, а й отримують регулярні оновлення завдяки активній спільноті розробників. Для професіоналів, наприклад у відео монтажі чи кіноіндустрії, наявність надійного програмного забезпечення для точного відтворення матеріалів також є критичною. Аналогічно, воно необхідне для відео спостереження, віртуальної реальності та різноманітних презентацій [6], [7].

Проте деякі програми для відтворення відеофайлів мають свої недоліки, які можуть впливати на якість користувацького досвіду. По-перше, не всі відеоплеєри підтримують усі існуючі формати, що іноді призводить до проблем із відтворенням певних файлів або необхідності встановлення додаткових кодеків. По-друге, деякі програми можуть бути ресурсомісткими, споживаючи багато оперативної пам'яті та процесорного часу, що особливо помітно на слабких або старих пристроях. Також варто зазначити, що деякі програми для відтворення відеофайлів мають складний або незручний інтерфейс, через що користувачам важко швидко знайти потрібні функції чи налаштувати параметри відтворення. Крім того, деякі популярні безкоштовні програми для відтворення відеофайлів можуть містити рекламу або спроби

встановлення додаткового програмного забезпечення, що знижує комфорт користування. Важливим недоліком також є недостатній рівень безпеки в окремих програмах, що потенційно може призвести до проблем з безпекою усієї системи користувача.

Тому актуальною є розробка програмного забезпечення для відтворення відеофайлів. У дипломній кваліфікаційній роботі бакалавра розв'язується актуальне завдання розробки програмного забезпечення для відтворення відеофайлів.

Для досягнення поставленої мети у кваліфікаційній роботі бакалавра необхідно розв'язати такі задачі:

- виконати аналіз предметної області та програмних засобів для відтворення відеофайлів;
- здійснити проєктування програмного забезпечення для відтворення відеофайлів;
- створити програмне забезпечення для відтворення відеофайлів;
- виконати тестування розробленого програмного забезпечення для відтворення відеофайлів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Програмне забезпечення для відтворення відеофайлів

Програмне забезпечення для відтворення відеофайлів дозволяє запускати мультимедійний контент на різних типах пристроїв без складних налаштувань або спеціального обладнання. Сьогодні такі програмні інструменти замінили традиційні пристрої зберігання та програвання відео, і тепер доступ до фільмів, кліпів або записів можливий як на комп'ютерах, так і на мобільних платформах чи у веббраузерах [1]-[3].

Цифрові відеоплеєри використовуються як для перегляду потокового відео, так і для роботи з локальними файлами. Браузерні рішення або автономні програми надають розширені можливості — наприклад, підтримку різних форматів, налаштування субтитрів, контроль якості відтворення. Завдяки цим функціям можна адаптувати відео до різних типів пристроїв та умов перегляду [1]-[3].

Щоб ділитися відео з іншими, можна застосовувати програмні сервіси, які дозволяють зберігати матеріали в хмарі або забезпечують завантаження на локальний пристрій для подальшого перегляду без доступу до мережі. Це зручно для організацій, які використовують відео як навчальний ресурс або інструмент комунікації в межах команд [1]-[3].

Такі програми також сприяють поширенню інформації в корпоративному середовищі. Завантаження інструкцій, презентацій або записів зустрічей дозволяє колегам переглядати їх у зручний час. Водночас використання відеоконтенту на сайтах сприяє зростанню зацікавленості відвідувачів і може покращити сприйняття компанії чи продукту [1]-[3].

Відеоплеєри стали універсальним засобом для доступу до аудіовізуального контенту, що охоплює як розважальні, так і професійні цілі. Але важливо пам'ятати і про ризики — наприклад, неконтрольоване розповсюдження авторського відео або копіювання без дозволу [1]-[3].

Коли розглядаються засоби для перегляду відео, варто розрізняти поняття медіапрогравача та відеопрогравача, хоча іноді ці назви використовуються як синоніми. Основна відмінність полягає в підході до обробки контенту: один тип орієнтовано на доступ до вже завантажених файлів, а інший — на негайне їх відтворення у потоковому режимі [2], [3].

Медіапрогравач зазвичай асоціюється з функціями, що дозволяють зберігати відео для подальшого перегляду, формувати списки, керувати колекцією файлів без постійного підключення до мережі. У свою чергу, відеопрогравачі створені для моментального відтворення відео, включно з можливістю перегляду в реальному часі з Інтернет джерел [2], [3].

Функціональність таких програм полягає в обробці цифрового відео: перетворення закодованого файлу у візуальне та звукове представлення на екрані. Для цього застосовуються спеціальні програмні алгоритми, відомі як кодеки. Їх використання забезпечує декодування відео- та аудіоданих, збережених у різних форматах [2], [3].

Формат відео безпосередньо впливає на сумісність із тим чи іншим плеєром. Наприклад, файли у форматі MP4 часто використовують стиснення через H.264, тоді як MKV-файли можуть містити складнішу структуру з декількома доріжками, що вимагає розширеної підтримки з боку програвача [2], [3].

Відеопрогравач не лише відповідає за виведення зображення, а й синхронізує аудіо та відео потоки, забезпечуючи цілісне та плавне відтворення. У більшості сучасних програм ці алгоритми вбудовані, хоча в окремих випадках потрібне додаткове встановлення відповідних компонентів для забезпечення повної функціональності [2], [3].

З розвитком цифрових медіа дедалі ширше застосування отримують відеопрогравачі, які здатні безпосередньо транслювати відеоконтент з Інтернету. Такий підхід усуває потребу в попередньому збереженні файлів, оскільки відтворення відбувається одразу з мережевого джерела. Програми, подібні до VLC, здатні працювати не лише з онлайн потоками, а й з

локальними ресурсами або конкретними мережевими адресами, що забезпечує більшу свободу при роботі з відео [2], [3].

Під час вибору відповідного програмного засобу для перегляду відео зазвичай береться до уваги сукупність характеристик, які впливають на зручність і якість користування. Однією з важливих умов є підтримка великої кількості відеоформатів і кодеків. Це має особливе значення тоді, коли використовуються нестандартні або рідше вживані типи файлів. Якщо плеєр здатен відкривати такі формати без додаткових налаштувань, користування ним стає значно простішим [2], [3].

Ще одним аспектом, що впливає на ефективність, є здатність програми до відтворення відео високої роздільної здатності. Для уникнення затримок або збоїв у такому разі доцільно застосовувати програвачі, які використовують апаратне прискорення. Це особливо актуально при роботі з 4K або навіть 8K контентом [2], [3].

Не менш важливою є й візуальна складова інтерфейсу. Простий, інтуїтивно зрозумілий дизайн або, навпаки, гнучкість у налаштуваннях — усе це залежить від уподобань користувача. Якщо потрібні більші можливості налаштувань під потреби користувача, вибір зазвичай припадає на більш функціональні програми [2], [3].

Додаткові можливості також відіграють роль. Деякі програвачі передбачають інтегровану підтримку субтитрів або зручні інструменти для їх синхронізації. Також часто доступна можливість змінювати зовнішній вигляд програми, працювати з онлайн потоками або застосовувати розширені параметри налаштувань [3], [4].

Щоб забезпечити безперебійну роботу відеоплеєра, важливо слідкувати за актуальністю програмного забезпечення. Регулярне оновлення дозволяє уникнути проблем із сумісністю нових форматів. Апаратне прискорення бажано активувати на системах із нижчими технічними характеристиками для стабільного відтворення. Якщо відео супроводжується субтитрами, варто перевірити, що їхнє відображення синхронізовано з відеорядом [3], [4].

Розуміння особливостей кожної програми допомагає обрати найпридатніший варіант, виходячи з власних потреб і технічних умов. Чи то перегляд особистої колекції, чи трансляція фільму з онлайн сервісу, відповідний інструмент здатен істотно покращити досвід взаємодії з відео контентом. Завдяки різноманіттю доступних рішень можливо знайти саме той варіант, який буде максимально відповідати очікуванням — незалежно від рівня користувача чи типу пристрою [3], [4].

Відео програвачі використовуються для відтворення мультимедійного вмісту, що включає відео, аудіо та інтерактивні елементи. Їх основною функцією є обробка цифрових даних, закодованих у спеціалізованих форматах, і перетворення цих даних у вигляд, який можна побачити або почути. Весь процес передбачає поетапне вилучення інформації з файлів-контейнерів, розпакування вмісту з використанням відповідних алгоритмів і передавання сигналів для візуального або звукового відтворення [3], [4].

Усередині типового програвача реалізовано цілу низку функціональних модулів. Один із таких компонентів відповідає за те, щоб зчитувати потоки даних із контейнерів і розмежовувати окремі типи контенту, як-от відео, аудіо чи субтитри. Інший елемент займається перетворенням стислих цифрових форматів у стандартні некомпресовані дані, які вже можуть бути виведені пристроями. За виведення графічних і звукових сигналів відповідає система рендерингу: вона адаптує зображення до потрібного формату, масштабує кадри, синхронізує звук, а також контролює такі параметри, як частота відображення чи звукові ефекти. Взаємодія з користувачем забезпечується через візуальну оболонку — зазвичай це набір елементів керування, що дозволяє запускати відео, змінювати гучність, перемикатися між доріжками та вмикати субтитри [3], [4].

Підхід до побудови таких систем значною мірою залежить від призначення програвача. Для онлайн передачі зазвичай використовується схема взаємодії, коли користувачеві надходить запитаний контент безпосередньо з віддаленого джерела, що забезпечує відтворення в реальному

часі. Крім того, можна зустріти рішення, засновані на децентралізованих мережах, де обмін даними відбувається між окремими учасниками без центрального сервера. Також не втратили актуальності варіанти, орієнтовані виключно на локальне зберігання й використання медіа, коли доступ до мережі взагалі не потрібен. Залежно від конкретного сценарію застосування вибирається найбільш доречна архітектура [3], [4].

Проаналізуємо види та характеристики програмного забезпечення для відтворення відеофайлів [3], [4]. Відеоплеєри реалізуються як програмні або апаратні рішення, залежно від середовища застосування. У випадку програмного підходу реалізація виконується у вигляді комп'ютерних застосунків, що функціонують на універсальних пристроях. Обробка мультимедійного потоку при цьому здійснюється засобами програмного коду, часто з опорою на відкриті програмні бібліотеки або інтерфейси, які забезпечують сумісність із великою кількістю форматів і стандартів. Завдяки таким підходам забезпечується широке охоплення типів контенту та можливість адаптації до різних джерел даних [3], [4].

Апаратний варіант передбачає використання спеціалізованих електронних модулів, завдяки яким обчислювальні навантаження з декодування передаються від основного процесора на виділені блоки. Це дозволяє досягати вищої швидкодії та знижувати споживання енергії. Відеокарти, що підтримують паралельну обробку, беруть участь у прискоренні декодування складних або ресурсомістких форматів, що забезпечує стабільне відтворення навіть за високої роздільності [3], [4].

Гібридний підхід полягає у комбінуванні програмної гнучкості з ефективністю апаратного прискорення. Подібні рішення застосовуються в інтелектуальних пристроях, де взаємодія з користувачем, потокові протоколи або додаткові функції реалізуються програмно, а ресурсомісткі обчислення делегуються спеціалізованому устаткуванню. Таким чином досягається баланс між продуктивністю та функціональністю [3], [4].

У сучасних умовах функціональність програмного забезпечення для відтворення відеофайлів значною мірою визначається набором підтримуваних кодувальних форматів. Можливість роботи з найпоширенішими алгоритмами стиснення відео та аудіо забезпечує високу якість відтворення при оптимізованому обсязі даних. Крім того, реалізація підтримки численних типів контейнерів гарантує взаємодію з великою кількістю мультимедійних файлів незалежно від джерела [3], [4].

Щоб забезпечити стабільне відтворення за умов нестабільного мережевого середовища, застосовується динамічна адаптація потоку. У таких ситуаціях здійснюється корекція якості в реальному часі, що дозволяє уникати пауз і затримок у трансляції контенту. Водночас дотримується необхідний рівень деталізації, враховуючи технічні можливості пристрою та швидкість з'єднання [3], [4].

Захист інтелектуальної власності впроваджується через механізми цифрового контролю доступу. Відповідні програмні системи взаємодіють із захисними системами, які контролюють легітимність користувача і запобігають несанкціонованому копіюванню чи розповсюдженню. У таких випадках забезпечується підтримка шифрування та обмеженого відтворення відповідно до ліцензійних умов [3], [4].

З метою покращення інклюзивності підтримується обробка субтитрів і текстових анотацій у різних форматах, що дозволяє розширити доступність контенту для осіб із порушеннями слуху або тих, хто споживає інформацію іншими мовами. Крім базового відображення, забезпечується можливість налаштування вигляду, синхронізації та навігації за допомогою текстових міток [3], [4].

Контроль за процесом відтворення виконується за рахунок інтегрованих механізмів перемотування, паузи, переходу до вказаних часових міток. Щоб уникнути порушення синхронності між відео- і аудіодоріжками, здійснюється узгоджене керування затримками та обробкою потоку, що також

важливо для багатодоріжкових записів. Це забезпечує цілісність сприйняття навіть у випадках складної внутрішньої структури відеофайлу [3], [4].

Під час реалізації програмного забезпечення для відтворення відеофайлів виникає потреба враховувати широкий спектр технічних параметрів, які впливають на якість роботи та ефективність функціонування системи. Важливим чинником є здатність до обробки великої кількості мультимедійних форматів, що включає як підтримку популярних кодеків, так і використання апаратного прискорення з метою зниження навантаження на центральний процесор. У середовищах, де критичною є мінімальна затримка або відтворення відео у високій роздільності, пріоритет надається системам із високим рівнем оптимізації продуктивності [4], [5].

З боку інженерних рішень необхідно передбачати відкритість інтерфейсів, можливість розширення функціоналу, а також підтримку модульної архітектури. Вибір рішень з відкритим програмним кодом дозволяє адаптувати програвач до специфічних потреб без суттєвих обмежень у масштабуванні або інтеграції в уже існуючі цифрові платформи. Це особливо актуально в умовах гнучких робочих процесів, де потрібна глибоке налаштування інструментів відтворення [4], [5].

Для забезпечення стабільної роботи на різних пристроях необхідно враховувати особливості кожної цільової платформи. Відмінності в обчислювальних можливостях мобільних телефонів та настільних комп'ютерів вимагають окремого підходу до оптимізації, особливо у частині використання графічних процесорів і сумісності з конкретними версіями операційних систем. Універсальність продукту, в тому числі підтримка менш поширених ОС, здатна суттєво розширити потенційну аудиторію [4], [5].

Сучасні вимоги до функціональності програмних засобів для відтворення відеофайлів диктують необхідність у спеціалізованій підтримці нових відеокодеків, таких як AV1 і VVC. Завдяки вдосконаленим алгоритмам стиснення, AV1 забезпечує високу якість зображення за меншого обсягу передаваних даних, що критично у випадках обмеженої пропускної здатності.

Проте, складність декодування цього формату вимагає відповідних внутрішніх оптимізацій, зокрема залучення апаратного прискорення та зниження впливу на системні ресурси. Ще більшу ефективність демонструє VVC, який орієнтований на обслуговування відео з надвисокою роздільністю, зокрема у форматі 4K та 8K. Для реалізації такого функціоналу важливо забезпечити надійність декодування навіть у нестабільних мережових умовах [4], [5].

Поряд із технологіями обробки відео дедалі більшого значення набуває збір та аналіз даних, пов'язаних із відтворенням. Реєстрація подій, пов'язаних із буферизацією, дозволяє виявляти вузькі місця у ланцюжку доставки контенту, тоді як фіксація помилок відтворення може вказувати на технічні несправності як на серверній, так і на клієнтській стороні. Поведінкові шаблони аудиторії, такі як перегортання, зупинки або перемотування, дають змогу сформувати уявлення про інтереси користувачів та налаштовувати подачу контенту відповідно до їхніх очікувань. Ці дані можуть оброблятися в реальному часі, надаючи розробникам інструменти для оперативного реагування на зміни у поведінці аудиторії [4], [5].

Наявність вбудованих аналітичних засобів і підтримка сучасних потокових технологій забезпечує відповідним програмним засобам здатність до надання стабільного користувацького досвіду в будь-яких умовах. Програмні рішення, орієнтовані на адаптивну передачу даних, низьку затримку та гнучкість у налаштуванні, демонструють найбільшу ефективність у сучасному середовищі. Такі програмні засоби поєднують підтримку новітніх стандартів кодування, функції реального моніторингу продуктивності та широкі можливості налаштування, створюючи надійну платформу для надання якісного відео контенту. У контексті швидкого розвитку технологій трансляцій, саме такі рішення дозволяють зберігати конкурентоспроможність і забезпечувати високі стандарти якості [4], [5].

У сучасному цифровому середовищі якісний відео контент є одним із досить затребуваних форматів взаємодії з аудиторією. Розробка інструментів для перегляду відео з розширеними мультимедійними функціями відкриває

для бізнесу нові горизонти та дозволяє зайняти помітну нішу на ринку. Такий підхід сприяє зміцненню бренду, адже унікальні особливості додатка здатні привернути увагу широкої аудиторії. Бренд, асоційований із зручністю, новизною та якістю, швидше входить у довіру до споживачів і стає об'єктом позитивної репутації [5], [6].

Впровадження механізмів монетизації в межах мультимедійного продукту дозволяє не лише покривати витрати на розробку, а й формувати стабільні джерела прибутку. Можливості платного доступу, інтеграція реклами, покупка додаткових функцій чи контенту забезпечують гнучкі сценарії для прибуткової експлуатації додатку [5], [6].

Інтеграція систем збору та обробки користувацької аналітики відкриває шлях до глибшого розуміння цільової аудиторії. Завдяки даним про поведінку, вподобання, час активності та географію переглядів з'являється можливість адаптувати продукт до реальних запитів користувачів, пропонуючи персоналізований досвід та релевантні пропозиції [5], [6].

Оптимізація взаємодії з кінцевим користувачем відбувається за рахунок інтуїтивно зрозумілого інтерфейсу та продуманої структури додатку. Удосконалення функціоналу програмного забезпечення, що відповідає сучасним очікуванням (наприклад, швидке перемотування, вибір аудіо доріжки, збереження позиції перегляду), підвищує задоволеність і частоту повторного використання [5], [6].

На ринку, де щодня з'являються нові додатки, важливо створити рішення, що вигідно відрізняється на фоні конкурентів. Власне програмне забезпечення для роботи з відеоданими із унікальними можливостями здатен забезпечити компанії технологічну перевагу та зміцнити її авторитет у галузі [5], [6].

Проаналізуємо етапи створення програмного забезпечення для відтворення відеофайлів. Реалізація проєкту зі створення застосунку для перегляду відео потребує системного підходу, що включає кілька важливих фаз. Першочергово проводиться аналітична підготовка, під час якої

визначається потенційна аудиторія, досліджуються актуальні тенденції ринку та оцінюється конкурентне середовище. Це дозволяє сформулювати чітке бачення майбутнього продукту та його позиціонування [6], [7].

На наступному етапі формується перелік ключових функцій, що будуть реалізовані. Пріоритет надається базовим можливостям перегляду – запуск, зупинка, перемотування, налаштування субтитрів та гучності. Уникнення надмірно складного функціоналу допомагає зосередитися на стабільній роботі програми та економії ресурсів на старті [6], [7].

Подальше просування проєкту передбачає вибір технологій. Підбір мови програмування, платформи для розгортання, систем зберігання даних і засобів безпеки визначає надійність та масштабованість майбутнього рішення. Успішність розробки значною мірою залежить від правильного технологічного стеку [6], [7].

Одночасно з технічним проєктуванням здійснюється створення інтерфейсу користувача. На цьому етапі важливо забезпечити естетику дизайну та зручність навігації. Увага приділяється логічності структури, адаптивності, швидкодії та візуальному стилю [6], [7].

Після завершення основної частини розробки, функціонал поступово реалізується в коді. Команда розробників інтегрує окремі модулі у єдину систему, забезпечуючи їхню коректну взаємодію. При цьому важливо зберігати стабільність навіть при збільшенні навантаження або додаванні нових опцій [6], [7].

Необхідною фазою є також всебічне тестування: перевірка продуктивності, виявлення потенційних помилок, адаптація під різні пристрої та умови використання. Всі недоліки, знайдені в процесі перевірки, мають бути усунені до моменту запуску [6], [7].

Фінальна частина – розгортання продукту на обраних платформах (наприклад, Google Play чи App Store) і організація маркетингової кампанії. Просування включає використання соціальних мереж, цільової реклами, email-

розсилок та партнерських проєктів. Саме цей етап визначає, наскільки ефективно продукт вийде на ринок і знайде свого користувача [6], [7].

Щоб реалізувати застосунок для перегляду відео, подібний до VLC, необхідно забезпечити його певним набором можливостей, які визначають комфорт користувача та багатофункціональність інтерфейсу. Основною вимогою є підтримка великої кількості аудіо- та відеоформатів, зокрема поширених та менш стандартних, таких як MP4, MKV, AVI чи FLV. Для зручності відтворення передбачається реалізація керувань, що дозволяють зупиняти, запускати, перемотувати відео або вимикати звук у будь-який момент [6], [7].

Регулювання гучності аудіо також повинно бути адаптоване під індивідуальні потреби, включаючи можливість її тимчасового відключення. Крім того, важливою є наявність підтримки субтитрів кількома мовами, що дозволяє краще зрозуміти контент незалежно від мови озвучення. Щоб надати користувачеві ще більше контролю, система має передбачати налаштування зміни контрасту, яскравості чи звукового профілю [6], [7].

Однією з ключових функцій є управління списками відтворення: користувач повинен мати змогу створювати, змінювати чи перемішувати списки, забезпечуючи безперервний перегляд контенту. Не менш важливою є можливість локального перегляду — навіть без доступу до Інтернету програма має функціонувати повноцінно, надаючи попередньо збережений контент. Інтерфейс має адаптуватися до різних екранів і режимів. Для швидкої навігації має бути реалізований пошук за контентом, який забезпечує легкий доступ до потрібного відео. Також варто подбати про зворотний зв'язок — користувачі повинні мати змогу звернутися до служби підтримки в будь-який час через різні канали зв'язку [6], [7].

Визначено, що програмне забезпечення для відтворення відеофайлів дозволяє запускати мультимедійний контент на різних типах пристроїв без складних налаштувань або спеціального обладнання. Проаналізовано види та характеристики програмного забезпечення для відтворення відеофайлів.

1.2 Аналіз програмного забезпечення для відтворення відеофайлів

Проаналізуємо програмне забезпечення для відтворення відеофайлів [8]-[14].

Програмне забезпечення Plex (рис. 1.1) являє собою потужне рішення для організації та відтворення мультимедійного контенту, яке забезпечує зручний доступ до особистих відео, музики, фотографій, серіалів та фільмів. Це програмне забезпечення надає можливість централізованого зберігання медіаданих, які потім можуть бути легко відтворені на різних пристроях, включаючи телевізійні платформи від сторонніх виробників. Завдяки гнучкій архітектурі, Plex дозволяє не лише працювати з власною медіатекою, а й отримувати доступ до потокового відео, включаючи телепередачі в реальному часі та контент за запитом без додаткової оплати [8], [12].

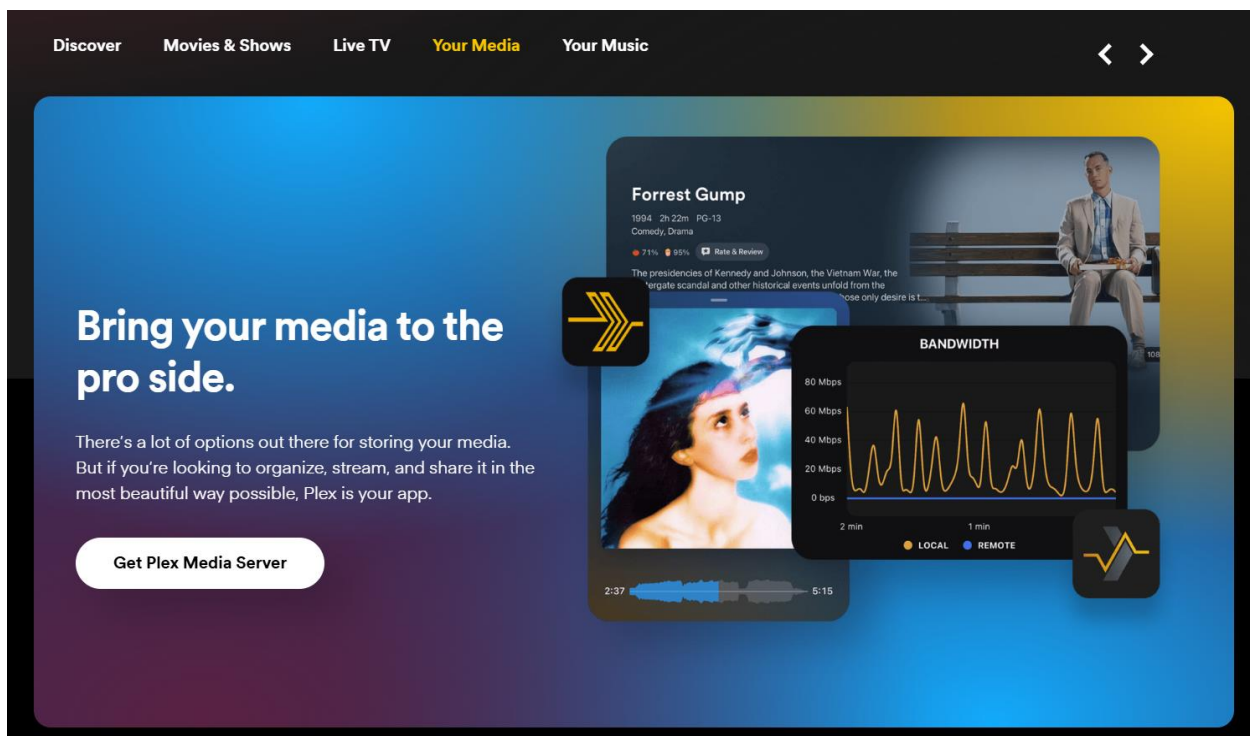


Рисунок 1.1 – Програмне забезпечення Plex [12]

Можливість розширення функціональності за допомогою спеціального підписного плану забезпечує доступ до таких інструментів, як запис

телепередач, пропуск вступних сцен у серіалах, а також раннє тестування нових можливостей. Сервіс сумісний із широким переліком платформ, завдяки чому споживання контенту не прив'язується до конкретного пристрою. Передбачено підтримку користувачів через електронну пошту, базу знань і спільноту, що дозволяє швидко знаходити рішення технічних питань. Програмне забезпечення підтримує широкий спектр форматів і стандартів, хоча відсутність підтримки відео надвисокої роздільної здатності може обмежити досвід відтворення на сучасних дисплеях. Незважаючи на це, Plex зберігає свою популярність завдяки поєднанню зручності, гнучкості та розвиненого функціоналу у своїй безкоштовній версії [8], [12].

Основними характеристиками програмного забезпечення Plex є такі [8], [12]:

- підтримка персональної медіатеки: програмне забезпечення Plex дозволяє організовувати, зберігати та відтворювати власні відео, музику, фото та інший мультимедійний контент з локальних або мережевих джерел [8], [12];
- безкоштовний доступ до контенту: програмне забезпечення Plex забезпечує можливість перегляду прямих телеканалів, фільмів і телешоу за запитом без додаткових витрат [8], [12];
- багатоплатформність: програмне забезпечення Plex сумісне з популярними потоковими пристроями, включаючи Roku, Amazon Fire TV, Apple TV, а також доступне на персональних комп'ютерах, мобільних телефонах та розумних телевізорах [8], [12];
- розширені можливості через підписку: за додаткову плату у програмному забезпеченні Plex відкривається доступ до функцій DVR-запису, пропуску вступів у серіалах і пріоритетного доступу до оновлень [8], [12];
- автоматичне оновлення метаданих: програмне забезпечення Plex підтримує зручну каталогізацію файлів із завантаженням обкладинок, описів і рейтингів з онлайн баз [8], [12];
- інтуїтивний інтерфейс: зручна навігація та швидкий доступ до улюбленого контенту завдяки простій і логічній структурі меню [8], [12];

– обмеження безкоштовної версії: деякі функції програмного забезпечення Plex, зокрема підтримка нових форматів або трансляція на мобільні пристрої, доступні лише з активною підпискою [8], [12].

Серед недоліків програмного забезпечення Plex можна відзначити обмеження у підтримці найновіших форматів відео, зокрема відсутність повноцінної роботи з надвисокою роздільною здатністю. Деякі функції, які можуть бути важливими для користувача, доступні лише в межах платного підписного плану, що звужує можливості безкоштовної версії. Також іноді виникають труднощі з автоматичним розпізнаванням метаданих, через що контент може бути некоректно каталогізований або позначений. У випадках використання на слабших пристроях можливе зниження швидкості роботи або затримки під час потокового передавання, особливо якщо виконується постійна трансформація файлів [8], [12].

Програмне забезпечення PotPlayer (рис. 1.2) являє собою відеоплеєр, що надає змогу відтворювати мультимедійні файли безкоштовно, не потребуючи придбання ліцензій чи підписки [8], [9], [13].



Рисунок 1.2 – Програмне забезпечення PotPlayer [13]

Програмне забезпечення PotPlayer здатне розпізнавати майже всі поширені формати, а в разі нестачі необхідного програмного забезпечення для декодування автоматично завантажує відповідні кодеки. Платформа орієнтована на високоякісне відтворення відео, включаючи підтримку записів надвисокої роздільної здатності. Вбудовані інструменти для запису екрана і простого редагування відео дають змогу працювати з контентом без потреби стороннього програмного забезпечення [8], [9], [13].

Користувачеві програмного забезпечення PotPlayer пропонується широке налаштування інтерфейсу, гнучке управління через клавіатурні комбінації, а також можливість створювати тимчасові позначки для зручного повернення до окремих моментів. Потужність функціоналу поєднується з плавною роботою навіть на менш продуктивних пристроях. Водночас, доступність програми обмежується платформою Windows, що виключає її використання на пристроях з іншими операційними системами. Інтерфейс може здатися складним для недосвідчених користувачів, однак відкриває широкі можливості для персоналізації [8], [9], [13].

Програмне забезпечення PotPlayer має такі особливості [8], [9], [13]:

- підтримка численних форматів: програмне забезпечення PotPlayer підтримує відтворення майже всіх популярних відео- та аудіоформатів без потреби у встановленні сторонніх кодеків [8], [9], [13];
- автоматичне завантаження кодеків: при відкритті файлів у невідомих форматах потрібні компоненти завантажуються автоматично [8], [9], [13];
- розширені функції перегляду: програмне забезпечення PotPlayer підтримує відтворення відео у різних форматах, зокрема, з високою роздільною здатністю [8], [9], [13];
- вбудовані інструменти: у програмне забезпечення PotPlayer інтегровано засоби для запису екрану та базового редагування відео [8], [13];
- гарячі клавіші та закладки: користувачам програмного забезпечення PotPlayer доступні функції швидкого керування та можливість створення позначок у певних моментах відтворення [8], [9], [13];

- налаштування інтерфейсу: програмне забезпечення PotPlayer підтримує зміну тем інтерфейсу користувача, кольорів і стилів відповідно до його потреб [8], [9], [13];

- висока продуктивність: програмне забезпечення PotPlayer вирізняється швидкою обробкою файлів та низьким споживанням системних ресурсів [8], [9], [13];

- доступність лише для Windows: програмне забезпечення PotPlayer працює виключно в середовищі цієї операційної системи [8], [9], [13].

Серед недоліків програмного забезпечення PotPlayer можна відзначити його доступність лише для операційної системи Windows, що обмежує можливості використання на інших платформах. Крім того, наявність великої кількості параметрів і налаштувань може ускладнити роботу для недосвідчених користувачів, які не знайомі з тонкощами відеовідтворення. Інтерфейс програми PotPlayer, хоча й гнучкий у налаштуванні, може здаватися перевантаженим і не зовсім інтуїтивним. Також варто згадати про відсутність офіційної мобільної версії, що обмежує зручність перегляду на портативних пристроях [8], [9], [13].

Програмне забезпечення KMPlayer (рис. 1.3) представляє собою гнучке мультимедійне рішення, яке дає змогу комфортно працювати з відеофайлами різної якості, включно з дуже високою роздільною здатністю. У програму вже вбудовано основні кодеки, тож вона може одразу запускати більшість поширених форматів, хоча при потребі користувачеві дозволено додавати інші. Програма не обмежується лише відео — вона також підтримує аудіофайли та дозволяє зручно впорядковувати улюблені треки й кліпи у власній добірці для швидкого доступу. Відтворення медіа не потребує локального збереження, адже програма дає змогу транслювати вміст безпосередньо з хмарних платформ, завдяки чому медіафайли можна переглядати без завантаження [8], [9], [14].

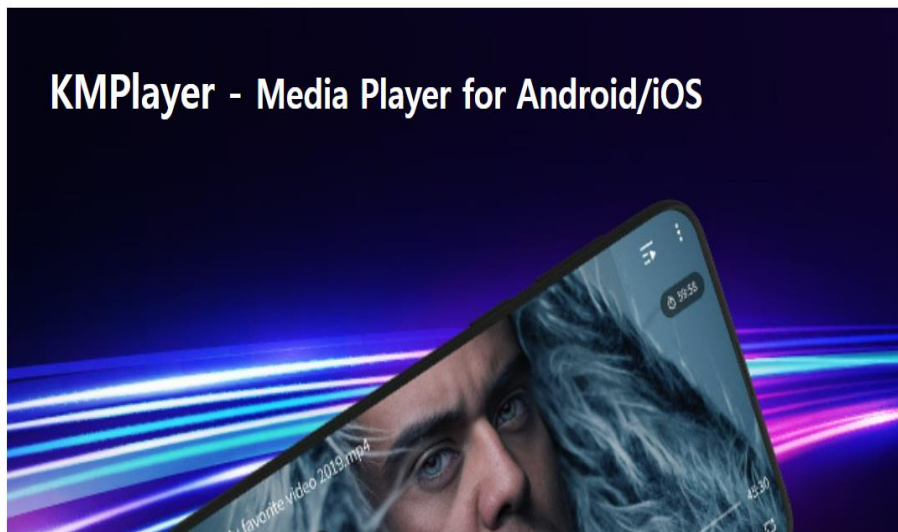
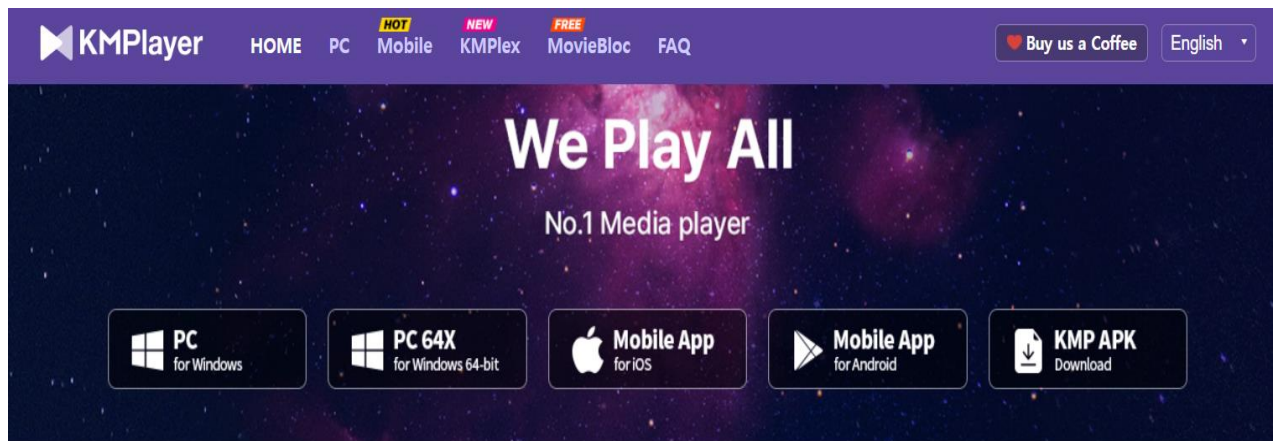


Рисунок 1.3 – Програмне забезпечення KMPlayer [14]

Програмне забезпечення KMPlayer підтримує різноманітні формати субтитрів та пропонує розширені можливості керування процесом відтворення, забезпечуючи зручність і гнучкість у користуванні [8], [9], [14].

Інтерфейс програмного забезпечення KMPlayer виглядає сучасно та функціонально, що дозволяє адаптувати робоче середовище до індивідуальних уподобань. Програма KMPlayer створена для плавної роботи навіть з ресурсомісткими відеоформатами, залишаючись водночас доступною без обов'язкових оплат [8], [9], [14].

Програмне забезпечення KMPlayer має такі особливості [8], [9], [14]:

- підтримка високої якості відео: програмне забезпечення KMPlayer відтворює відеофайли з роздільною здатністю до ultra HD та 8K без втрати якості зображення [8], [9], [14];

- сумісність із багатьма форматами: програмне забезпечення KMPlayer підтримує відтворення широкого спектра аудіо- та відеоформатів без потреби в додаткових кодексах [8], [9], [14];

- можливість потокової трансляції: контент можна переглядати безпосередньо з хмарних сервісів, таких як Google Drive та Dropbox, без попереднього завантаження [8], [9], [14];

- гнучке керування відтворенням: програмне забезпечення KMPlayer надає доступні інструменти для точного налаштування процесу перегляду, включно з регулюванням швидкості, розміру вікна та положення субтитрів у екрані відображення відео [8], [9], [14];

- зручна організація медіафайлів: користувачі програмного забезпечення KMPlayer підтримує можуть створювати власні списки улюблених відео й музики для швидкого доступу до контенту [8], [9], [14];

- сучасний інтерфейс: програмне забезпечення KMPlayer має інтуїтивно зрозумілий вигляд з можливістю змінювати теми й оформлення інтерфейсу в залежності від бажань користувача [8], [9], [14];

- безкоштовна модель з доплатами: основний функціонал програмного забезпечення KMPlayer підтримує доступний безкоштовно, однак окремі розширення вимагають покупок у застосунку [8], [9], [14].

Серед недоліків програмного забезпечення KMPlayer можна відзначити наявність рекламного контенту в безкоштовній версії, що може відволікати під час користування. Крім того, інтерфейс із великою кількістю налаштувань і функцій іноді сприймається як перевантажений, особливо для тих, хто очікує простоти. Іноді також спостерігаються труднощі зі стабільністю під час відтворення окремих форматів або при роботі з великими файлами через мережу [8], [9], [14].

Порівняльну характеристику програмного забезпечення для відтворення відеофайлів наведено у таблиці 1.1.

Таблиця 1.1 – Порівняння програмного забезпечення для відтворення відеофайлів

Критерій порівняння	Plex [12]	PotPlayer [13]	KMPlayer [14]
Підтримка багатьох форматів файлів	+—	+	+
Можливість редагування відео	—	+	+—
Сумісність із хмарними сервісами	+	—	+
Наявність мобільної версії	+	—	+
Інтеграція з онлайн контентом	+	—	+—
Простота використання	+	+—	+—

За результатами проведеного аналізу можна зробити висновок, що у наш час існує досить багато програмних засобів для відтворення відеофайлів. Проте деякі програмні засоби для відтворення відеофайлів не підтримують усі існуючі формати, що іноді призводить до проблем із відтворенням певних файлів або необхідності встановлення додаткових кодеків. Крім того, деякі програми можуть бути ресурсомісткими, споживаючи багато оперативної пам'яті та процесорного часу, що особливо помітно на слабких або старих пристроях. Також варто зазначити, що деякі програми для відтворення відеофайлів мають складний або незручний інтерфейс, через що користувачам важко швидко знайти потрібні функції чи налаштувати параметри відтворення. Деякі популярні безкоштовні програми для відтворення відеофайлів можуть містити рекламу або спроби встановлення додаткового програмного забезпечення, що знижує комфорт користування. Важливим недоліком також є недостатній рівень безпеки в окремих програмах, що потенційно може призвести до проблем з безпекою усієї системи користувача. Тому актуальною є розробка програмного забезпечення для відтворення відеофайлів.

При розробці програмного забезпечення для відтворення відеофайлів необхідно забезпечити такі функціональні вимоги:

- підтримка можливості завантаження відеофайлу для подальшого відтворення;
- підтримка можливості відтворення відеофайлів;
- можливість зупинення процесу відтворення відеофайлів;
- можливість регулювання гучності у процесі відтворення відеофайлу;
- підтримка можливості перемотування відео у процесі відтворення.

1.3 Висновки за розділом 1

Визначено, що програмне забезпечення для відтворення відеофайлів дозволяє запускати мультимедійний контент на різних типах пристроїв без складних налаштувань або спеціального обладнання. Проаналізовано види та характеристики програмного забезпечення для відтворення відеофайлів.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато програмних засобів для відтворення відеофайлів. Проте деякі програмні засоби для відтворення відеофайлів не підтримують усі існуючі формати, що іноді призводить до проблем із відтворенням певних файлів або необхідності встановлення додаткових кодеків. Крім того, деякі програми можуть бути ресурсомісткими, споживаючи багато оперативної пам'яті та процесорного часу, що особливо помітно на слабких або старих пристроях. Також варто зазначити, що деякі програми для відтворення відеофайлів мають складний або незручний інтерфейс, через що користувачам важко швидко знайти потрібні функції чи налаштувати параметри відтворення. Деякі популярні безкоштовні програми для відтворення відеофайлів можуть містити рекламу або спроби встановлення додаткового програмного забезпечення, що знижує комфорт користування. Важливим недоліком також є недостатній рівень безпеки в окремих програмах, що потенційно може призвести до проблем з безпекою усієї системи користувача.

Тому актуальною є розробка програмного забезпечення для відтворення відеофайлів.

Сформульовано функціональні вимоги до програмного забезпечення для відтворення відеофайлів.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Вибір мови програмування

При розробці програмного забезпечення для відтворення відеофайлів було використано мову програмування C++ [15], [16].

C++ належить до мов програмування, які поєднують широкі можливості управління ресурсами з високим рівнем універсальності при побудові програм будь-якої складності. Завдяки цьому забезпечується підвищена ефективність обчислень та можливість створення оптимізованих рішень із чітко структурованою архітектурою. Мову програмування C++ часто обирають для розробки застосунків, де критично важливими є швидкодія, контроль над пам'яттю та адаптивність до різних середовищ виконання програми [15], [16].

Завдяки прямому доступу до апаратних ресурсів мова програмування C++ дозволяє розробникам реалізовувати алгоритми з високим рівнем продуктивності, що особливо цінується в системах з інтенсивною обробкою даних. Її застосування не обмежується лише мультимедійними рішеннями: мова програмування C++ активно використовується в розробці драйверів, ігрових рушіїв, складних обчислювальних систем і програмного забезпечення для вбудованих платформ. Це зумовлено можливістю детального контролю над потоками, кешуванням, управлінням пам'яттю та іншими низькорівневими процесами [15], [16].

Мова програмування C++ перебуває в активному розвитку вже багато десятиліть, отримуючи регулярні оновлення відповідно до вимог сучасної індустрії. Завдяки таким нововведенням, як автоматичне визначення типів, лямбда-вирази, уніфіковане ініціалізування чи нові шаблонні механізми, створення складного програмного забезпечення стало менш трудомістким і більш гнучким. При цьому C++ зберігає сумісність із попередніми версіями, що дає змогу масштабувати існуючі проєкти без потреби повного переписування коду [15], [16].

Сьогодні мова програмування C++ розглядається не просто як інструмент для створення програм, а як потужна технологічна основа для реалізації складних інженерних рішень, де поєднуються надійність, швидкодія й можливість тонкого налаштування функціоналу програми під конкретні завдання користувача [15], [16].

У сфері розробки програмного забезпечення, орієнтованого на роботу з відео, мову C++ доцільно застосовувати через її здатність ефективно взаємодіяти з апаратними компонентами системи. Забезпечується висока швидкість обробки відеоданих, що критично важливо при декодуванні, трансформації кадрів та виведенні зображення на екран у режимі реального часу. Обчислення можуть виконуватись із мінімальними затримками завдяки прямому доступу до процесорних та графічних ресурсів, що особливо помітно під час використання складних відеокодеків [15], [16].

Мова програмування C++ дозволяє без труднощів інтегрувати широке коло зовнішніх бібліотек, серед яких — FFmpeg, OpenCV, DirectX і Vulkan. Таке поєднання відкриває широкі можливості для реалізації мультимедійних функцій без надмірних витрат на обслуговування або складну оптимізацію. Крім того, завдяки підтримці об'єктно-орієнтованого підходу вдається чітко структурувати програмну архітектуру, забезпечуючи зрозумілість і логічну організацію коду [15], [16].

Доступ до великої кількості прикладів, шаблонів і технічної документації значно полегшує процес проєктування та прискорює розробку застосунків, орієнтованих на відеообробку. Сукупність цих факторів робить C++ ефективним інструментом для створення продуктивного, стабільного та масштабованого програмного забезпечення, що здатне обробляти відеофайли різних форматів із максимальною точністю й швидкодією [15], [16].

Обґрунтування вибору мови програмування для розробки програмного забезпечення для відтворення відеофайлів наведено у таблиці 2.1.

Таблиця 2.1 – Обґрунтування вибору мови програмування для розробки програмного забезпечення для відтворення відеофайлів

Критерій порівняння мов програмування	Мова програмування		
	C++	Python	Java
Висока продуктивність	+	—	+—
Підтримка мультимедійних бібліотек	+	+—	+—
Оптимізація апаратних ресурсів	+	—	+
Зручність для створення ПЗ для відтворення відеофайлів	+	—	—
Масштабованість і підтримка	+	+—	+

Отже, для реалізації програмного забезпечення для відтворення відеофайлів обрано мову програмування C++, яка забезпечує високу продуктивність і ефективність роботи, що критично для обробки відеоданих у режимі реального часу, адже відтворення відео вимагає швидкої обробки потоків даних. Крім того, C++ дає можливість низькорівневого контролю над апаратними ресурсами, що дозволяє максимально оптимізувати використання процесора, пам'яті та відеокарти, особливо при роботі з різними кодеками і форматами. Важливо перевагою є те, що мова програмування C++ підтримує об'єктно-орієнтований підхід, що сприяє структурованій архітектурі програмного забезпечення для відтворення відеофайлів та полегшує його розширення і підтримку. Завдяки широкому поширенню C++ та великій спільноті розробників, існує багато готових рішень, документації і прикладів, що прискорює розробку. Отже, вибір C++ для створення програми для відтворення відеофайлів є обґрунтованим через її високу продуктивність, гнучкість, широкі можливості оптимізації та сумісність з мультимедійними технологіями, що робить її однією з найкращих мов для таких задач.

2.2 Вибір середовища розробки для створення програмного забезпечення для відтворення відеофайлів

В якості середовища розробки для створення програмного забезпечення для відтворення відеофайлів було обрано середовище Qt Creator [17], [18].

Qt Creator розглядається як повноцінне середовище для створення програмного забезпечення, де поєднуються зручність, гнучкість та функціональна насиченість. Завдяки глибокій інтеграції з бібліотеками фреймворку Qt забезпечується стабільна підтримка міжплатформного програмування, що дозволяє створювати застосунки, які успішно працюють у різних операційних середовищах без необхідності суттєвої переробки коду. Це, у свою чергу, позитивно впливає на терміни розробки та оптимізує витрати на подальше обслуговування [17], [18].

Використання внутрішньої мови, що активно підтримується всередині Qt Creator, сприяє створенню адаптивних та інтуїтивно зрозумілих інтерфейсів користувача. У процесі проєктування значно полегшується реалізація візуальних елементів, анімації та динамічної взаємодії. Вбудовані засоби керування проєктами дозволяють впорядковувати структуру розробки, що особливо важливо при роботі з великими програмними системами, зокрема мультимедійного призначення [17], [18].

Середовище підтримує розширення функціональності за допомогою додаткових модулів, що відкриває можливість адаптації до специфіки різних команд та типів програмних рішень. Засоби аналізу продуктивності дозволяють детально дослідити виконання програмного коду та виявити вузькі місця, які можуть впливати на ефективність застосунку, зокрема у проєктах, де обробляються великі обсяги відеоданих. Активна спільнота забезпечує постійну підтримку та обмін досвідом [17], [18].

Отже, Qt Creator слугує не лише як середовище програмування, а як універсальний інструментальний простір, орієнтований на професійну

розробку сучасних програмних продуктів, де поєднується технологічна досконалість, продуктивність і зручність реалізації ідей [17], [18].

Для створення програм, пов'язаних із відеобробкою, особливу увагу приділяється вибору відповідного інструментарію, і середовище розробки Qt Creator у цьому контексті розглядається як практичне рішення завдяки можливості використання на різних платформах та зручному графічному середовищу. Його функціональність охоплює написання, редагування та тестування коду з можливістю швидкої реалізації візуальних елементів, що особливо важливо для програм із розвиненим інтерфейсом. Забезпечується сумісність із різними операційними системами, включно з настільними й мобільними платформами, що дозволяє запускати один і той самий застосунок на різних пристроях без суттєвої адаптації [17], [18].

Середовище розробки активно використовується там, де потрібна підтримка мультимедійних форматів, зокрема аудіо та відео, що дає змогу значно спростити реалізацію програм зі складною медіа архітектурою. Використання мови C++ у межах Qt Creator забезпечує достатній рівень продуктивності, що критично важливо для швидкої обробки графічних даних. Інструменти, вбудовані у середовище, дозволяють оптимізувати процес керування проєктами, спрощуючи процеси збирання, тестування та розгортання програмного забезпечення [17], [18].

Низький поріг входження для новачків пояснюється інтуїтивно зрозумілим інтерфейсом і добре організованою структурою редактора. Доступ до численних навчальних матеріалів, прикладів та технічних рішень забезпечується активною спільнотою розробників. Усе це сприяє створенню сучасних застосунків для роботи з відео та зображеннями, використовуючи актуальні можливості програмування у зручному й продуктивному середовищі [17], [18].

Обґрунтування вибору середовища розробки для створення програмного забезпечення для відтворення відеофайлів наведено у таблиці 2.2.

Таблиця 2.2 – Обґрунтування вибору середовища розробки для створення програмного забезпечення для відтворення відеофайлів

Критерій порівняння середовищ розробки	Середовища розробки		
	Qt Creator	Visual Studio	CLion
Кросплатформеність	+	+—	+
Зручність візуального проектування інтерфейсів	+	+—	—
Підтримка багатоплатформених проєктів	+	+—	+
Швидкість роботи	+	+—	+—
Інтеграція з системами контролю версій	+	+	+—

Таким чином, для створення програмного забезпечення для відтворення відеофайлів обрано середовище розробки Qt Creator через його потужний набір інструментів, які значно спрощують створення кросплатформених застосунків з графічним інтерфейсом користувача. Qt Creator інтегрує в собі редактор коду, засоби налагодження та візуального дизайну, що дозволяє швидко і ефективно розробляти як прості, так і складні програми. Особливо важливою є те, що Qt має власний потужний фреймворк для роботи з мультимедіа, включно з підтримкою відео та аудіо, що полегшує інтеграцію відтворення відеофайлів і робить розробку більш гнучкою і швидкою. Крім того, Qt Creator підтримує роботу з C++, що є оптимальним вибором для програм, які потребують високої продуктивності. Вбудовані інструменти дозволяють легко керувати проєктами, автоматизувати збірку, тестування і розгортання програмного забезпечення.

2.3 Висновки за розділом 2

Для реалізації програмного забезпечення для відтворення відеофайлів обрано мову програмування C++, яка забезпечує високу продуктивність і ефективність роботи, що критично для обробки відеоданих у режимі реального часу, адже відтворення відео вимагає швидкої обробки потоків даних. Крім того, C++ дає можливість низькорівневого контролю над апаратними ресурсами, що дозволяє максимально оптимізувати використання процесора, пам'яті та відеокарти, особливо при роботі з різними кодеками і форматами. Важливо перевагою є те, що мова програмування C++ підтримує об'єктно-орієнтований підхід, що сприяє структурованій архітектурі програмного забезпечення для відтворення відеофайлів та полегшує його розширення і підтримку.

Для створення програмного забезпечення для відтворення відеофайлів обрано середовище розробки Qt Creator через його потужний набір інструментів, які значно спрощують створення кросплатформених застосунків з графічним інтерфейсом користувача. Qt Creator інтегрує в собі редактор коду, засоби налагодження та візуального дизайну, що дозволяє швидко і ефективно розробляти як прості, так і складні програми. Особливо важливою є те, що Qt має власний потужний фреймворк для роботи з мультимедіа, включно з підтримкою відео та аудіо, що полегшує інтеграцію відтворення відеофайлів і робить розробку більш гнучкою і швидкою.

3 ОПИС ПРОГРАМИ

3.1 Структура програмного забезпечення для відтворення відеофайлів

Структуру програмного забезпечення для відтворення відеофайлів наведено на рис. 3.1.

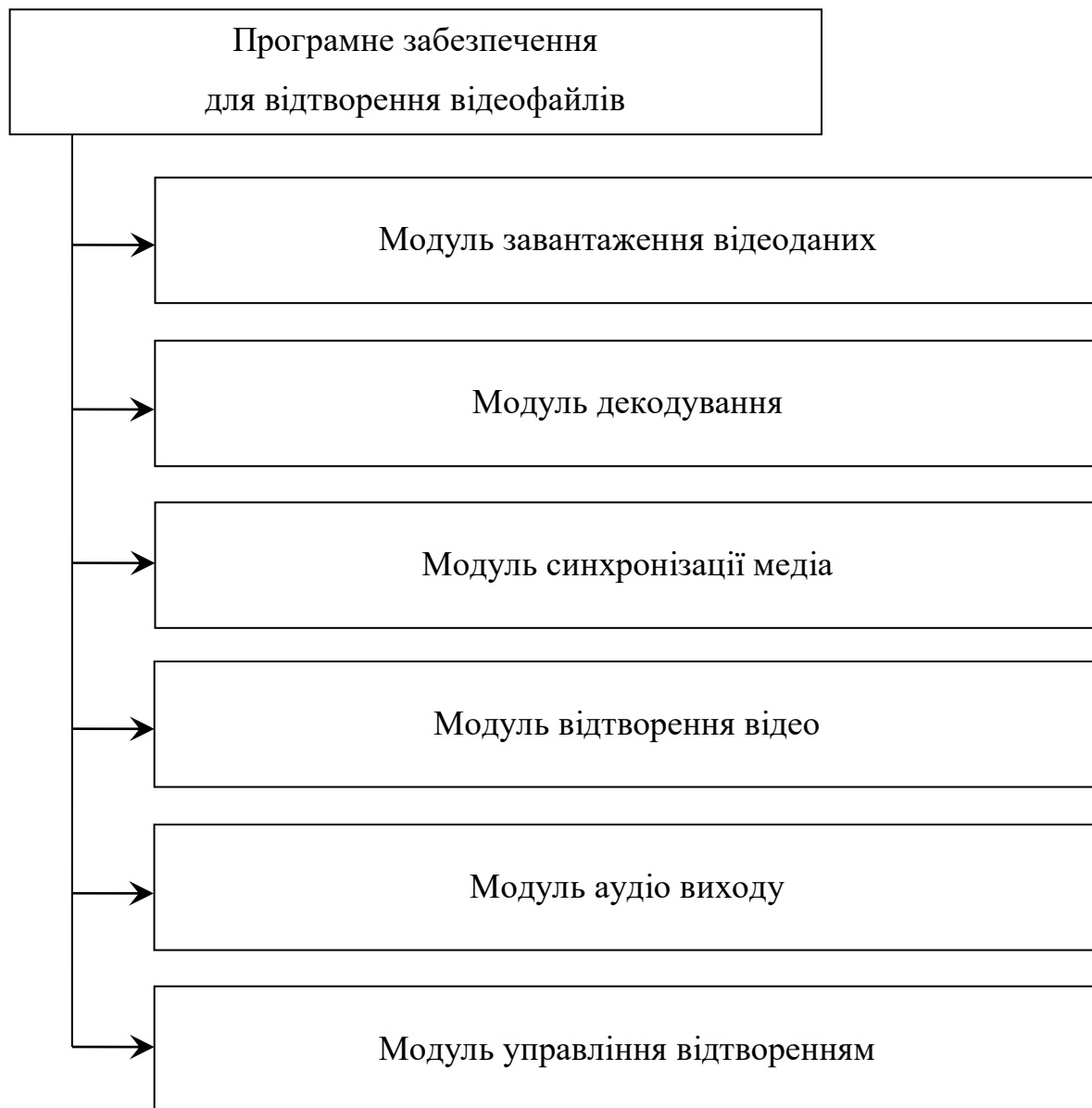


Рисунок 3.1 – Структура програмного забезпечення для відтворення відеофайлів

Архітектура програмного забезпечення для відтворення відеофайлів ґрунтується на поділі всієї системи на шість окремих функціональних модулів, кожен з яких відповідає за певний етап обробки та виведення мультимедійного контенту.

Початкову роль у процесі виконує модуль завантаження відеоданих. Його завдання полягає у відкритті файлу, ініціалізації потоків і підготовці даних до обробки. Саме цей компонент здійснює аналіз контейнера, виявляє наявні відео- та аудіодоріжки, встановлює параметри для наступних етапів обробки, а також передає отримані пакети до відповідних черг.

Другим є модуль декодування, що відповідає за перетворення стиснених даних у придатну до подальшого використання форму. Цей компонент приймає закодовані пакети з черг і розпаковує їх у фрейми, які вже містять зображення чи аудіосигнали у сирому вигляді. Саме тут здійснюється взаємодія з бібліотеками низького рівня, що реалізують підтримку різноманітних форматів стиснення.

Подальша обробка відбувається в модулі синхронізації медіа, який узгоджує час відтворення звукових і відеофрагментів. Цей модуль визначає, коли саме фрейм має бути передано на екран або до аудіосистеми, і коригує їх послідовність відповідно до часових міток. У разі виявлення відставання чи випередження одного потоку відносно іншого виконується відповідна компенсація, що забезпечує цілісне та плавне сприйняття.

Модуль відтворення відео займається безпосереднім відображенням кадрів на екрані. Він приймає фрейми з декодера або буфера синхронізації, масштабує їх до розмірів вікна, виконує необхідну кольорокорекцію та передає дані для візуалізації. У його межах також реалізовано механізм обробки подій, пов'язаних із графічним інтерфейсом — наприклад, перемальовування кадру у випадку зміни розмірів вікна чи активації повноекранного режиму.

Окремо функціонує модуль аудіо виходу, який відповідає за передачу аудіофреймів до звукового пристрою. Після декодування звукові дані

буферизуються, а потім поступово зчитуються апаратною частиною. Завдання модуля — підтримувати рівномірне надходження звуку, обробляти зміни гучності та повідомляти інші компоненти системи про завершення відтворення окремих фрагментів.

Завершує архітектуру модуль управління відтворенням. Він координує взаємодію всіх інших частин системи, приймає команди користувача (наприклад, відтворення, пауза, перемотування), забезпечує ініціалізацію й завершення роботи програми, а також відповідає за обробку помилок і відновлення у випадку нестандартних ситуацій. Через нього реалізується логіка програвача як цілісного застосунку.

Відзначимо, що у реальному середовищі можуть траплятися ситуації, коли відеофайл пошкоджено або відсутні деякі ключові кадри. Структура програми передбачає обробку таких виняткових ситуацій через пропуск кадрів, повторне зчитування, попередження користувача або автоматичне перемикання на інше джерело. Крім того, відзначимо, що у програмі використовуються бібліотеки для роботи з різними форматами даних, що дозволяє не змінювати логіку програми при додаванні підтримки нових типів файлів, оскільки бібліотека для обробки відео даних ізольована в окремому модулі, а всі інші компоненти лише взаємодіють із нею через уніфікований інтерфейс.

Отже, структура програмного забезпечення для відтворення відеофайлів реалізується за принципами модульності та слабкого зв'язку між компонентами, що забезпечує легкість супроводу, оновлення окремих частин системи або заміни одних реалізацій іншими.

3.2 Функціонування програмного забезпечення для відтворення відеофайлів

Функціонування програмного забезпечення для відтворення відеофайлів подамо за допомогою схеми, зображеної на рис. 3.2.

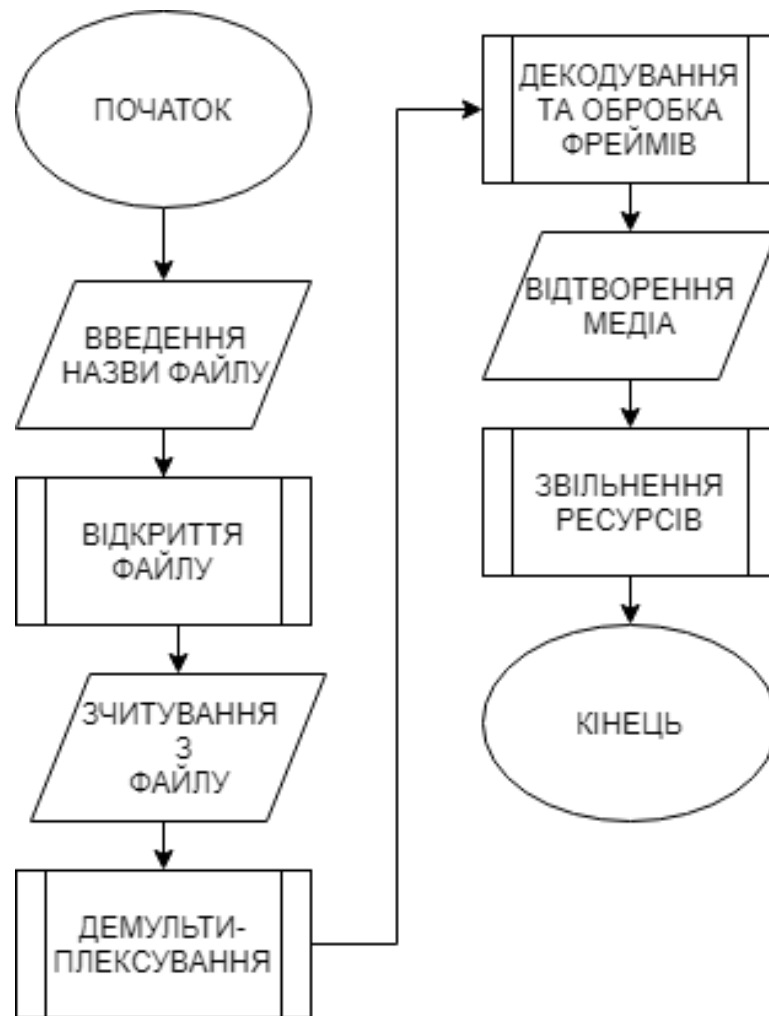


Рисунок 3.2 – Схема функціонування програмного забезпечення для відтворення відеофайлів

Робота програмного забезпечення для відтворення відеофайлів відбувається за певною послідовністю, яка починається з вибору джерела медіа. Через графічний інтерфейс здійснюється звернення до файлової системи, де обирається потрібний файл. Після підтвердження вибору, шлях до відеофайлу передається в програмну логіку, де ініціюється процес відкриття за допомогою низькорівневих інструментів. Після цього формується контекст, що описує структуру файлу, та проводиться його ініціалізація.

Коли відбувається аналіз вмісту файлу, з'ясовується, які саме типи потоків у ньому присутні — відео, аудіо чи інші. На основі цієї інформації визначаються відповідні потоки, кожен із яких описується структурою, що містить технічні характеристики: частоту кадрів, кількість фреймів, формат

даних тощо. Отримані дані дозволяють правильно підготувати декодери для подальшої роботи.

Далі починається обробка потоку, який ще не розділений — для цього залучається механізм декодування. Використовується відповідний модуль, який отримує послідовність пакетів і відокремлює відео- та аудіо-компоненти. У результаті з'являються окремі блоки даних, які передаються для подальшої обробки — кожен за своїм напрямом. Тип кожного пакета визначається під час виконання, і згідно з ним викликається відповідний метод для передачі в систему декодування.

Компоненти, що відповідають за обробку даних, працюють у взаємодії через визначені інтерфейси. Механізм спостереження забезпечує обмін сигналами між джерелом та одержувачем: після отримання нового пакета надсилається повідомлення, і відповідний декодер виконує дію згідно з типом медіа. Контролер, що координує роботу всіх частин системи, містить посилання на декодери, організовуючи взаємодію в межах відповідного шаблону, що приховує складність взаємодії внутрішніх елементів.

Надходження пакета в декодер не означає негайної обробки. Спочатку такий пакет розміщується в черзі. Якщо черга заповнена, виконання відповідного потоку призупиняється до моменту, поки не з'явиться місце. Завдяки цьому контролюється використання пам'яті та попереджується перевантаження системи. Оскільки операції декодування та виводу займають більше часу, ніж зчитування, це рішення дозволяє зберегти стабільність роботи без втрати продуктивності.

Завершення обробки передбачає вивільнення ресурсів, які були використані в ході виконання, що запобігає накопиченню зайвих обсягів у пам'яті та сприяє стабільному функціонуванню програми при повторних зверненнях до інших файлів.

Після запуску програмного забезпечення, кожен з модулів, відповідальний за декодування, починає функціонувати у фоновому режимі, очікуючи надходження закодованих пакетів із вхідної черги. Як тільки

доступний новий медіа-пакет, розпочинається процес його обробки. Декодування здійснюється окремим обчислювальним потоком, що дозволяє уникати блокування основного виконання програми. Після завершення декодування формується структура, яка містить вже розшифровані медіа-дані. Така структура зазвичай представлена у вигляді універсального контейнера, здатного зберігати як відео, так і аудіо інформацію залежно від типу вхідного потоку. Разом із цим у структурі закладена інформація щодо внутрішнього формату даних та параметрів їх синхронного відтворення.

Поява нового фрейму автоматично фіксується внутрішнім механізмом сповіщення, після чого відповідний інтерфейс отримує повідомлення про доступність медіа-даних.

Щойно новий фрейм готовий, відповідне повідомлення про це передається спеціалізованому обробнику декодера, який перехоплює повідомлення про доступний фрейм через виклик перевизначеного методу, після чого передає цей фрейм далі – вже наступному учаснику у ланцюгу відтворення. Цей компонент відповідає за перетворення отриманих даних на візуальні або звукові сигнали, які користувач зможе безпосередньо сприймати. Відповідний зв'язок здійснюється через інтерфейс, який дозволяє приймати нові фрейми й реагувати на зміну стану медіа системи.

Щодо графічного представлення відео, цю функцію виконує компонент, який базується на класі з відповідної бібліотеки. Саме він відповідає за відображення відео на екрані користувача. Попередньо зображення піддається масштабуванню відповідно до геометрії інтерфейсного елемента за допомогою бібліотеки. Малювання здійснюється у межах перевизначеного методу. Важливо, що цей процес має відбуватися у контексті подій. Для ініціювання оновлення зображення викликається метод, подія публікується через функцію із вказівкою цільового об'єкта. Обробник подій у належний момент обробляє її, що й спричиняє оновлення елемента інтерфейсу.

Аудіо ж обробляється окремим компонентом, його задача полягає у збереженні надісланих фреймів у спеціалізованому буфері. З цього буфера компонент здійснює періодичне зчитування даних, автоматично приводячи до відтворення звуку. Параметри, що описують формат аудіопотоку, задаються на етапі ініціалізації, і далі вже сам компонент забезпечує правильну синхронізацію виводу.

У рамках роботи з мультимедійними даними особливу увагу зосереджено на ефективному керуванні оперативною пам'яттю. Через велику кількість фрагментів відео- й аудіоінформації, що обробляються упродовж відтворення, постійне створення і зберігання структур даних без своєчасного їхнього вивільнення може призвести до надмірного навантаження на систему. Тому процес звільнення пам'яті відбувається негайно після завершення опрацювання чергового елемента даних.

Як тільки черговий блок закодованої інформації використовується декодером для побудови фрейму, його присутність у пам'яті втрачає актуальність. У цей момент виконуються спеціальні інструкції, які очищають структуру і звільняють пов'язану з нею оперативну пам'ять. Внутрішньо використовується рекомендований механізм, що спершу скидає вміст буфера, а потім повністю ліквідує залишки структури з оперативної пам'яті. Це забезпечує швидке очищення простору для нових даних, без зайвого накопичення.

Коли фрейм створено, він переходить до подальшої обробки — для відео це відображення на екрані, а для аудіо — передача до відповідного буфера. При роботі з відео характерною є потреба тимчасового збереження останнього кадру, навіть після його первинного використання. Такий підхід дозволяє уникати візуальних збоїв у випадку, коли оновлення зображення викликається повторно, наприклад, унаслідок зміни розміру вікна. У подібній ситуації використовується той самий кадр, який залишався у пам'яті до прибуття наступного.

Натомість аудіо дані мають дещо інший життєвий цикл. З моменту копіювання звукового фрагмента у внутрішній буфер апаратного пристрою, його подальше зберігання втрачає сенс. Тому одразу після підтвердження передачі фрейму в буфер клієнтський компонент інформує центральну частину програвача про завершення опрацювання, що запускає процес знищення структури звукового кадру. Повідомлення передається послідовно, спершу від компонента до керівника процесом, далі до контролера, який і звільняє пам'ять, застосовуючи відповідні функції очищення.

Візуальна частина працює за схожим принципом. Компонент, відповідальний за формування зображення, після завершення відображення повідомляє про це програвач, який ініціює вивільнення ресурсу. Завдяки чіткому розмежуванню обов'язків між модулями, а також каскадному підходу до вивільнення, забезпечується ефективне використання пам'яті, що дозволяє уникати перевантаження системи навіть при тривалому відтворенні.

3.3 Опис основних алгоритмів програмного забезпечення для відтворення відеофайлів

У процесі створення програмного забезпечення для відтворення відеофайлів розроблено декілька алгоритмів.

Робота алгоритму декодування у програмній системі відтворення відеофайлів організована навколо окремого потоку виконання, що відповідає за перетворення стиснених відеоданих у непрямий формат, придатний для подальшого відображення чи відтворення (рис. 3.3). Ініціація цього процесу здійснюється одразу після того, як демультіплексор розпізнає структуру вхідного мультимедійного потоку та визначає, які типи даних у ньому містяться. На етапі налаштування декодера формується відповідне середовище для обробки даних. У якості відправної точки використовується опис потоку, що включає параметри кодування. З отриманих параметрів за допомогою функціональності бібліотеки декодування визначається тип кодека, що

підходить для обробки вхідного потоку. Далі створюється об'єкт контексту, до якого передаються ці параметри, і вже через нього ініціалізується декодер як функціональна одиниця, готова до роботи.



Рисунок 3.3 – Схема ітерації алгоритму декодування у програмному забезпеченні для відтворення відеофайлів

Після завершення ініціалізації запускається спеціалізований потік, призначений для неперервного читання даних із черги вхідних пакетів. Ця черга реалізована як безпечна структура, що дозволяє синхронізувати дії декодера, який додає туди пакети, та декодера, що їх вилучає. Механізм блокування доступу забезпечується за допомогою м'ютексів, що дозволяють уникнути конфліктів при паралельному доступі з різних потоків.

У кожній ітерації обробки потік декодування намагається взяти черговий пакет даних. Якщо такий пакет присутній, він передається до декодера, де запускається процес розпакування. Спочатку пакет передається функції, що відповідає за його обробку — вона поміщає його у внутрішній буфер декодера. Потім викликається друга функція, яка з цього буфера намагається дістати вже розшифрований фрейм. Ці виклики повторюються стільки разів, скільки кадрів виявлено всередині пакета.

Після кожного вдало отриманого фрейму система передає його далі по ланцюжку — зазвичай у модуль синхронізації чи безпосередньо у відео- чи аудіобуфер для відтворення. У разі, якщо кадрів у пакеті більше немає або обробка завершена, декодер звільняє ресурси, пов'язані з цим пакетом, і переходить до наступного елемента в черзі.

Весь процес триває доти, поки не буде вичерпано всі вхідні дані, або поки не буде надано сигнал про завершення відтворення. Завдяки тому, що операції з чергою та сам процес декодування відбуваються у власному потоці, інші частини системи можуть паралельно виконувати свої функції, що забезпечує плавне відтворення навіть при великих обсягах оброблюваних відеоданих.

У процесі відтворення відеофайлів у мультимедійному програмному забезпеченні питанням виняткової важливості є синхронність аудіо та відео потоків. З огляду на те, що аудіопотік здатний відтворюватися без розривів, а відео — піддається повторному показу або затримкам, виникає необхідність ретельної координації моментів відображення кадрів. Спиратися виключно на формальні значення частоти кадрів не є надійною стратегією, оскільки вони можуть бути неточними або нестабільними, а сама апаратна основа системи — не гарантувати постійної продуктивності.

У кожному відеопакеті міститься інформація щодо часу, у який повинен бути відтворений відповідний фрейм. Ці часові мітки, а саме параметр PTS (presentation time stamp), дозволяють прив'язати момент візуалізації кадру до конкретної позиції на шкалі часу. В основі реалізації

синхронізації закладено принцип вирівнювання відеоряду під потік аудіо, який виступає основною часовою віссю. З появою нового фрейму виконується обчислення інтервалу очікування до моменту його виведення.

Для забезпечення точного контролю над часом було використано монотонний таймер, здатний відображати реальний хід часу незалежно від системних змін. Запуск таймера відбувається на етапі ініціалізації, після чого на його базі будується вся логіка синхронізації. Ключовий компонент, відповідальний за координацію, – це модуль-синхронізатор, до якого надходить кадр для обробки. Після отримання фрейму, обчислюється затримка, необхідна для його відображення згідно з часовою міткою, та коригується у відповідності до поточної швидкості відтворення аудіо.

Частота відео не сприймається як константа, а підлягає динамічному регулюванню. З урахуванням повторюваності кадрів, яка фіксується відповідним параметром, відбувається оновлення темпу виведення зображення. Для стартового кадру затримка обчислюється як обернена величина до номінальної частоти кадрів. Подальші фрейми потребують точнішого розрахунку на основі дійсної різниці між поточними значеннями PTS аудіо та відео.

Алгоритм корекції затримки передбачає початкове визначення шляхом віднімання попередньої мітки часу кадру з поточної. Далі, враховуючи часовий розрив між аудіо- і відеопотоками, перевіряється, чи перевищує він допустимий діапазон. У разі суттєвого розбігу затримка скасовується, а за надмірно малої різниці – подвоюється. Наступним кроком стає оновлення значення внутрішнього відеотаймера, після чого визначається фактична тривалість паузи до наступного кадру. Якщо обчислена пауза не перевищує встановленого мінімального порогу, вона замінюється на допустиме значення. Таким чином, кожне відображення кадру виконується з урахуванням зміщення в часі, а відеопотоки підтримують узгодженість.

У процесі створення програмного забезпечення для відтворення відеофайлів виникла потреба в реалізації певних допоміжних механізмів,

необхідних для забезпечення коректної роботи системи відтворення. Серед таких рішень окреме значення має процедура масштабування зображення без втрати пропорційності, а також алгоритм, що відповідає за визначення координат зображення в межах області його виводу. Для оптимізації аудіопроцесів також було реалізовано структуру буферизованого накопичення та вибірки даних, що організована за принципом кільцевого буфера.

Буфер виконує роль проміжної пам'яті для тимчасового зберігання аудіоданих, які надходять із потоку декодування. Принцип його роботи полягає в тому, що нові блоки інформації записуються в область, яка логічно визначається як кінець буфера, а вибірка здійснюється з іншої його частини, що відповідає поточній точці читання. У міру зчитування зсувається відповідна позиція, а дані, які вже опрацьовані, можуть бути звільнені для повторного використання пам'яті. Таким чином забезпечується неперервність обробки звукового потоку без перевантаження оперативної пам'яті. Обсяг буфера обмежується наперед встановленим значенням, що дозволяє уникнути переповнення системи при різниці в темпах декодування й фактичного відтворення.

Для обробки відео в умовах змінної області виводу довелося впровадити механізм, який розраховує допустимі розміри кадру, що помістяться в екранну область без спотворення пропорцій. Оскільки використовуваний інструментарій не підтримує автоматичного збереження співвідношення сторін, обрахунок відповідних розмірів здійснюється окремо. Для цього визначаються коефіцієнти масштабування по ширині та висоті, обирається мінімальний з них, після чого обчислюються нові габарити кадру. Ураховується також специфічна вимога до кратності ширини певному числу, як правило чотирьом, що зумовлено особливостями деяких алгоритмів кодування зображень.

Після визначення розмірів формується контекст масштабування, де зазначаються характеристики як вихідного, так і майбутнього кадру. Далі виконується безпосереднє масштабування, що трансформує фрейм до нових

параметрів, а отриманий результат копіюється у певну структуру. Побайтове перенесення відбувається стандартними засобами мови програмування, що дозволяє передати графічний кадр у компоненти графічного інтерфейсу.

На завершальному етапі відбувається розрахунок позиції, з якої кадр має бути виведений на екран. Центрування здійснюється через визначення залишкових розмірів вільного простору по горизонталі та вертикалі. Залежно від того, яка з цих величин більша, обирається відповідний напрям для зміщення. Таким чином зображення розміщується в межах області відображення максимально естетично та без спотворення.

3.4 Проєктування інтерфейсу програмного забезпечення для відтворення відеофайлів

Проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для відтворення відеофайлів виконано з урахуванням вимог технічного завдання. При розробленні програмного забезпечення для відтворення відеофайлів враховані функціональні вимоги до програми:

- підтримка можливості завантаження відеофайлу для подальшого відтворення;
- підтримка можливості відтворення відеофайлів;
- можливість зупинення процесу відтворення відеофайлів;
- можливість регулювання гучності у процесі відтворення відеофайлу;
- підтримка можливості перемотування відео у процесі відтворення.

Інтерфейс головної форми програмного забезпечення для відтворення відеофайлів наведено на рис. 3.4.

Регулювання гучності у процесі відтворення відеофайлів наведено на рис. 3.5.

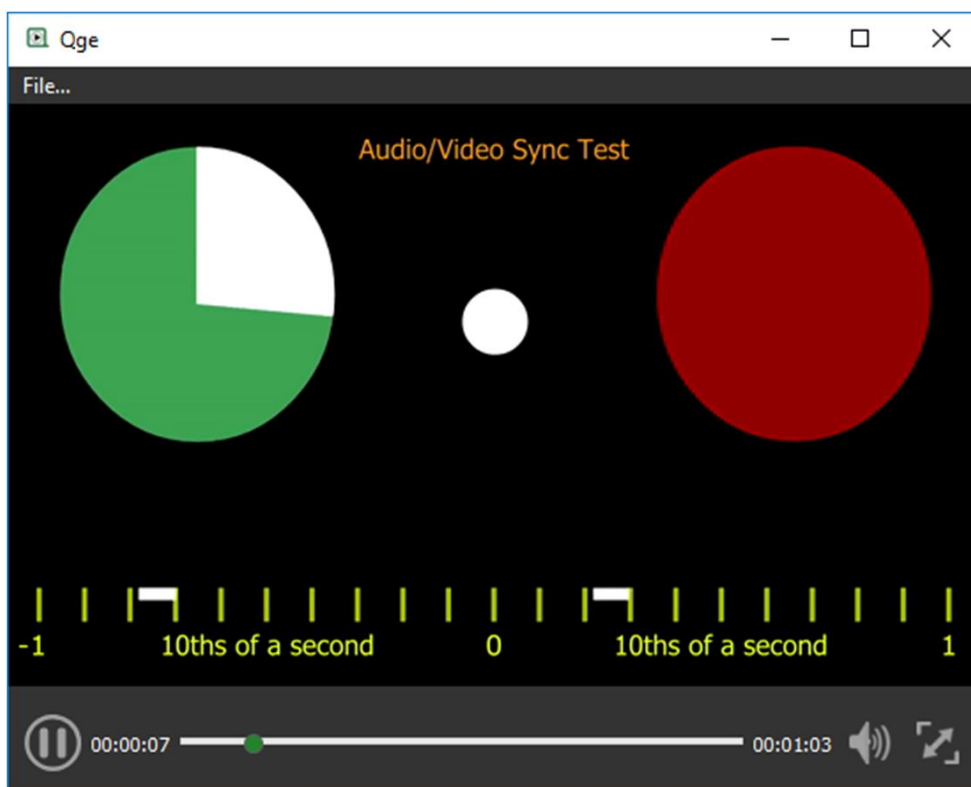


Рисунок 3.4 – Інтерфейс головної форми програмного забезпечення для відтворення відеофайлів

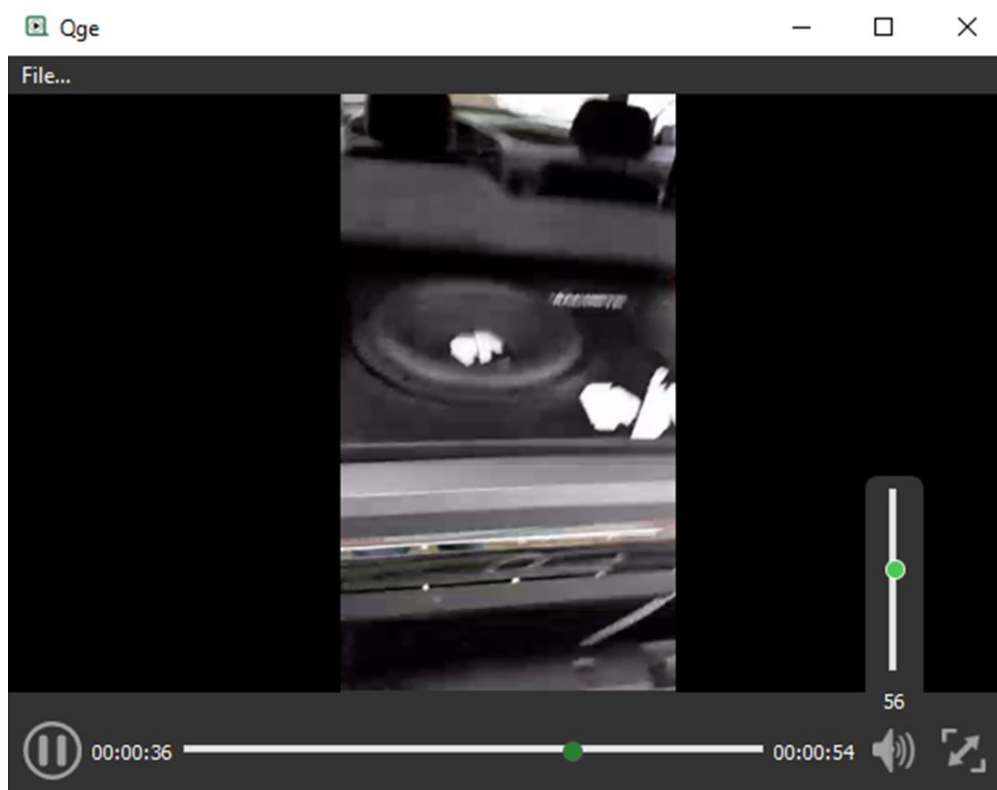


Рисунок 3.5 – Регулювання гучності у процесі відтворення відеофайлів

3.5 Висновки за розділом 3

Розроблено програмне забезпечення для відтворення відеофайлів.

Запропоновано структуру програмного забезпечення для відтворення відеофайлів. Архітектура програмного забезпечення для відтворення відеофайлів ґрунтується на поділі всієї системи на шість окремих функціональних модулів (модуль завантаження відеоданих, модуль декодування, модуль синхронізації медіа, модуль відтворення відео, модуль аудіо виходу, модуль управління відтворенням), кожен з яких відповідає за певний етап обробки та виведення мультимедійного контенту.

Описано функціонування програмного забезпечення для відтворення відеофайлів, робота програмного забезпечення для відтворення відеофайлів відбувається за певною послідовністю, яка починається з вибору джерела медіа, через графічний інтерфейс здійснюється звернення до файлової системи, де обирається потрібний файл. Після підтвердження вибору, шлях до відеофайлу передається в програмну логіку, де ініціюється процес відкриття за допомогою низькорівневих інструментів, після цього формується контекст, що описує структуру файлу, та проводиться його відтворення.

Описано особливості реалізації програмного забезпечення для відтворення відеофайлів. Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для відтворення відеофайлів.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Програма призначена для відтворення відеофайлів. Програма написана на мові C++, яка забезпечує високу продуктивність і ефективність роботи, що критично для обробки відеоданих у режимі реального часу, адже відтворення відео вимагає швидкої обробки потоків даних. В якості середовища розробки для створення програмного забезпечення для відтворення відеофайлів обрано Qt Creator, що інтегрує в собі редактор коду, засоби налагодження та візуального дизайну, що дозволяє швидко і ефективно розробляти як прості, так і складні програми.

Програмне забезпечення для відтворення відеофайлів забезпечує виконання таких функцій:

- підтримка можливості завантаження відеофайлу для подальшого відтворення;
- підтримка можливості відтворення відеофайлів;
- можливість зупинення процесу відтворення відеофайлів;
- можливість регулювання гучності у процесі відтворення відеофайлу;
- підтримка можливості перемотування відео у процесі відтворення.

Для роботи програмного забезпечення для відтворення відеофайлів потрібні такі технічні та програмні ресурси: наявність процесора з тактовою частотою не нижче 2,5 ГГц із підтримкою багатопотокової обробки, що дозволяє забезпечити своєчасне декодування та відображення відеопотоків у реальному часі. Мінімальний обсяг оперативної пам'яті має становити не менше 4 ГБ, хоча для стабільної роботи з відео високої чіткості рекомендовано щонайменше 8 ГБ. Відеокарта з підтримкою апаратного прискорення відео (наприклад, через технології NVDEC, VAAPI або Quick Sync) значно підвищує продуктивність, особливо при роботі з форматом H.264 або новішими кодеками. У якості програмного середовища слід використовувати операційну

систему з підтримкою багатозадачності. Для забезпечення належної роботи всіх компонентів необхідна наявність встановленого мультимедійного фреймворку FFmpeg, який відповідає за обробку, декодування та масштабування відео. Крім того, обов'язковою є підтримка бібліотек Qt версії не нижче 5.12, оскільки інтерфейс користувача реалізовано з використанням цього фреймворку. Наявність вільного простору на жорсткому диску, щонайменше 500 МБ, потрібна для встановлення програми та збереження тимчасових файлів, включно з буферами декодованого відео та аудіо.

Функціональні характеристики розробленого програмного забезпечення для відтворення відеофайлів наведено у технічному завданні (додаток А). Фрагмент тексту програмного забезпечення для відтворення відеофайлів наведено у додатку Б.

4.2 Характеристики програми для відтворення відеофайлів

Програмне забезпечення для відтворення відеофайлів складається з шести окремих функціональних модулів (модуль завантаження відеоданих, модуль декодування, модуль синхронізації медіа, модуль відтворення відео, модуль аудіо виходу, модуль управління відтворенням), кожен з яких відповідає за певний етап обробки та виведення мультимедійного контенту.

Програмне забезпечення для відтворення відеофайлів характеризується здатністю обробляти мультимедійні потоки різних форматів у режимі реального часу. Його основною функцією є декодування аудіо- та відеоданих, а також їх синхронізоване відтворення із мінімальними затримками. Забезпечується підтримка широкого спектра кодеків, що дає змогу працювати з файлами у форматах, які найчастіше зустрічаються у користувацькому середовищі.

Програмне забезпечення має гнучку систему масштабування зображення з урахуванням пропорцій кадру, що дозволяє коректно адаптувати відео до розмірів області відтворення без втрати якості. Передбачено

механізми буферизації звукових даних, що забезпечує безперервне та плавне звучання навіть при незначних затримках зчитування.

Графічна складова реалізується з використанням програмних бібліотек, завдяки чому забезпечується висока продуктивність при мінімальному навантаженні на систему. Інтерфейс адаптовано для інтеграції в більші застосунки або для автономного використання, з урахуванням потреб кінцевого користувача. Особливу увагу приділено ефективному використанню апаратних ресурсів та підтримці апаратного прискорення при наявності відповідного обладнання.

Вся логіка взаємодії між модулями побудована таким чином, щоб гарантувати модульність і масштабованість, що спрощує подальшу розробку або інтеграцію з іншими системами.

4.3 Інструкція по експлуатації програми

4.3.1 Звернення до програми

Для запуску програмного забезпечення для відтворення відеофайлів необхідно запустити виконуваний файл.

4.3.2 Вхідні й вихідні дані

Вхідними даними до програмного забезпечення для відтворення відеофайлів є відеофайли різних форматів, які містять закодовані аудіо- та відеопотоки.

Вихідними даними програмного забезпечення для відтворення відеофайлів є декодовані зображення кадрів, які передаються графічному інтерфейсу для виведення на екран, а також звукові дані, що направляються в аудіосистему пристрою для відтворення.

4.3.3 Повідомлення

Користувач програмного забезпечення для відтворення відеофайлів може отримати такі повідомлення: повідомлення про некоректність відеофайлу та про неможливість його відображення.

4.4 Виконання програмного забезпечення для відтворення відеофайлів

Після запуску програмного забезпечення для відтворення відеофайлів користувачу відображається головна форма (рис. 4.1). Для відкриття відеофайлу необхідно у головному меню File обрати пункт Відкрити та за допомогою стандартного діалогового вікна операційної системи відкрити потрібний файл. Для запуску процесу відтворення відеофайлу необхідно натиснути кнопку з позначкою трикутника у лівій нижній частині екрану. Для призупинення відтворення потрібно натиснути ту ж кнопку (яка у процесі відтворення відображається як дві вертикальні риски). Інтерфейс програми для відтворення відеофайлів є подібним до аналогічного програмного забезпечення.

У процесі відтворення також відображається доріжка з відміткою місця, що відповідає поточному відео кадру. Для перемотування відео необхідно перемістити відповідну позначку на зазначеній доріжці. Регулювання гучності у процесі відтворення відеофайлів відбувається за допомогою кнопки з позначкою динаміка (рис. 4.2) шляхом збільшення чи зменшення відповідного рівня.

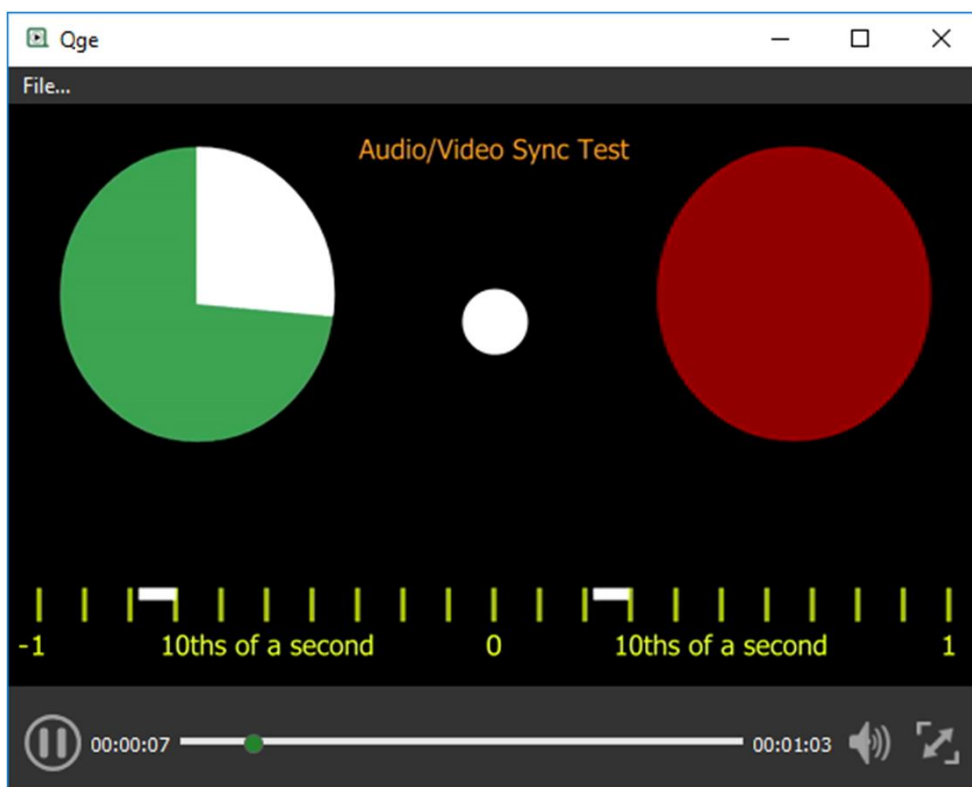


Рисунок 4.1 – Інтерфейс головної форми програмного забезпечення для відтворення відеофайлів

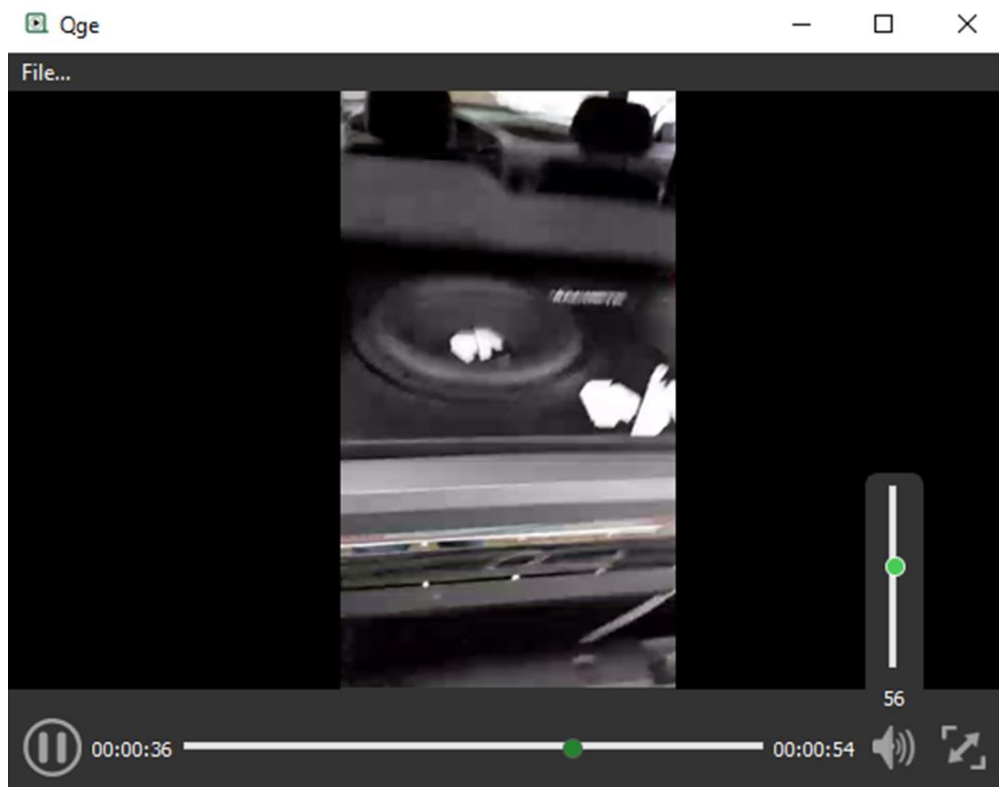


Рисунок 4.2 – Регулювання гучності у процесі відтворення відеофайлів

4.5 Тестування програмного забезпечення для відтворення відеофайлів

Виконано тестування програмного забезпечення для відтворення відеофайлів.

Виявлено, що програма для відтворення відеофайлів функціонує правильно та злагоджено. Вона реалізує всі функціональні вимоги та успішно виконує свою основну задачу підтримки процесу роботи з відеофайлами.

Розроблене програмне забезпечення дозволяє забезпечувати автоматизацію процесів, пов'язаних з підтримкою процесу відтворення відеофайлів.

4.6 Висновки за розділом 4

Описано програмне забезпечення для відтворення відеофайлів. Виконано тестування розробленого програмного забезпечення для відтворення відеофайлів. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє забезпечити автоматизацію процесів, пов'язаних з підтримкою процесу відтворення відеофайлів.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було проаналізовано та досліджено процес розробки програмного забезпечення для відтворення відеофайлів.

Визначено, що програмне забезпечення для відтворення відеофайлів дозволяє запускати мультимедійний контент на різних типах пристроїв без складних налаштувань або спеціального обладнання. Проаналізовано види та характеристики програмного забезпечення для відтворення відеофайлів.

За результатами проведеного аналізу зроблено висновок, що у наш час існує досить багато програмних засобів для відтворення відеофайлів. Проте деякі програмні засоби для відтворення відеофайлів не підтримують усі існуючі формати, що іноді призводить до проблем із відтворенням певних файлів або необхідності встановлення додаткових кодеків. Крім того, деякі програми можуть бути ресурсомісткими, споживаючи багато оперативної пам'яті та процесорного часу, що особливо помітно на слабких або старих пристроях. Також варто зазначити, що деякі програми для відтворення відеофайлів мають складний або незручний інтерфейс, через що користувачам важко швидко знайти потрібні функції чи налаштувати параметри відтворення. Деякі популярні безкоштовні програми для відтворення відеофайлів можуть містити рекламу або спроби встановлення додаткового програмного забезпечення, що знижує комфорт користування. Важливим недоліком також є недостатній рівень безпеки в окремих програмах, що потенційно може призвести до проблем з безпекою усієї системи користувача. Тому актуальною є розробка програмного забезпечення для відтворення відеофайлів.

Сформульовано функціональні вимоги до програмного забезпечення для відтворення відеофайлів.

Для реалізації програмного забезпечення для відтворення відеофайлів обрано мову програмування C++, яка забезпечує високу продуктивність і

ефективність роботи, що критично для обробки відеоданих у режимі реального часу, адже відтворення відео вимагає швидкої обробки потоків даних. Крім того, C++ дає можливість низькорівневого контролю над апаратними ресурсами, що дозволяє максимально оптимізувати використання процесора, пам'яті та відеокарти, особливо при роботі з різними кодексами і форматами. Важливо перевагою є те, що мова програмування C++ підтримує об'єктно-орієнтований підхід, що сприяє структурованій архітектурі програмного забезпечення для відтворення відеофайлів та полегшує його розширення і підтримку.

Для створення програмного забезпечення для відтворення відеофайлів обрано середовище розробки Qt Creator через його потужний набір інструментів, які значно спрощують створення кросплатформених застосунків з графічним інтерфейсом користувача. Qt Creator інтегрує в собі редактор коду, засоби налагодження та візуального дизайну, що дозволяє швидко і ефективно розробляти як прості, так і складні програми. Особливо важливою є те, що Qt має власний потужний фреймворк для роботи з мультимедіа, включно з підтримкою відео та аудіо, що полегшує інтеграцію відтворення відеофайлів і робить розробку більш гнучкою і швидкою.

Розроблено програмне забезпечення для відтворення відеофайлів.

Запропоновано структуру програмного забезпечення для відтворення відеофайлів. Архітектура програмного забезпечення для відтворення відеофайлів ґрунтується на поділі всієї системи на шість окремих функціональних модулів (модуль завантаження відеоданих, модуль декодування, модуль синхронізації медіа, модуль відтворення відео, модуль аудіо виходу, модуль управління відтворенням), кожен з яких відповідає за певний етап обробки та виведення мультимедійного контенту.

Описано функціонування програмного забезпечення для відтворення відеофайлів, робота програмного забезпечення для відтворення відеофайлів відбувається за певною послідовністю, яка починається з вибору джерела медіа, через графічний інтерфейс здійснюється звернення до файлової

системи, де обирається потрібний файл. Після підтвердження вибору, шлях до відеофайлу передається в програмну логіку, де ініціюється процес відкриття за допомогою низькорівневих інструментів, після цього формується контекст, що описує структуру файлу, та проводиться його відтворення.

Описано особливості реалізації програмного забезпечення для відтворення відеофайлів. Виконано проєктування інтерфейсу взаємодії користувача з програмним забезпеченням для відтворення відеофайлів.

Виконано тестування розробленого програмного забезпечення для відтворення відеофайлів. Результати тестування програмного забезпечення показали, що розроблена програма дозволяє забезпечити автоматизацію процесів, пов'язаних з підтримкою процесу відтворення відеофайлів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. What is a video player? Plus, what platforms to consider [Electronic resource]. – Access mode: <https://vimeo.com/blog/post/what-is-a-video-player>.
2. The Essential Guide to Video Players and Video Playback: Everything You Need to Know [Electronic resource]. – Access mode: <https://modspure.com/the-essential-guide-to-video-players-and-video-playback-everything-you-need-to-know/>.
3. What is a media player [Electronic resource]. – Access mode: <https://www.fastpix.io/blog/what-is-a-media-player>.
4. How to build a video playback system with Reeltime Pro [Electronic resource]. – Access mode: <https://pomfort.com/article/how-to-build-a-video-playback-system-with-reeltime-pro/>.
5. How To Develop A Video Player App Like VLC [Electronic resource]. – Access mode: <https://devtechnosys.com/insights/develop-a-video-player-app-like-vlc/>.
6. How to Create a Custom Video Player [Electronic resource]. – Access mode: <https://flowplayer.com/blog/how-to-create-custom-video-player>.
7. Build a custom HTML5 video player [Electronic resource]. – Access mode: <https://www.creativebloq.com/html5/build-custom-html5-video-player-9134473>.
8. The Best Video Players [Electronic resource]. – Access mode: <https://www.bestapp.com/best-video-player-apps/>.
9. Best media players [Electronic resource]. – Access mode: <https://www.av4you.com/blog/best-media-players/>.
10. Best Free Video Players/Media Players [Electronic resource]. – Access mode: <https://www.cyberlink.com/blog/media-player-windows/370/best-free-video-media-player>.
11. Free Video Players [Electronic resource]. – Access mode: <https://www.anymp4.com/player/free-video-player.html>.
12. Plex [Electronic resource]. – Access mode: <https://www.plex.tv/>.

13. PotPlayer [Electronic resource]. – Access mode:
<https://potplayer.daum.net/>.

14. KMPlayer [Electronic resource]. – Access mode:
<https://kmplayer.com/home>.

15. C++ Language [Electronic resource]. – Access mode:
<https://cplusplus.com/doc/tutorial/>.

16. C++ Standard Library [Electronic resource]. – Access mode:
https://en.cppreference.com/w/cpp/standard_library.

17. Qt Framework [Electronic resource]. – Access mode:
<https://www.qt.io/product/framework>.

18. Qt Creator Manual [Electronic resource]. – Access mode:
<https://thinkingintqt.com/doc/qtcreator/index.html>.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення може використовуватися для відтворення відеофайлів.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Програмне забезпечення для відтворення відеофайлів», затверджене наказом Національного університету «Запорізька політехніка» № 209 від 28 квітня 2025 р.

A.2 Призначення розробки

Програмний продукт призначений для перегляду відеофайлів.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для відтворення відеофайлів забезпечує виконання таких функцій:

- підтримка можливості завантаження відеофайлу для подальшого відтворення;
- підтримка можливості відтворення відеофайлів;
- можливість зупинення процесу відтворення відеофайлів;
- можливість регулювання гучності у процесі відтворення відеофайлу;
- підтримка можливості перемотування відео у процесі відтворення.

А.3.2 Вимоги до інтерфейсу програми

Інтерфейс програмного забезпечення для відтворення відеофайлів повинен бути зручним для користувачів. Повинна бути забезпечена можливість роботи в візуальному режимі.

А.3.3 Вимоги до надійності

Програмне забезпечення для відтворення відеофайлів повинно забезпечити надійне функціонування.

А.3.4 Умови експлуатації

Для експлуатації програмного забезпечення для відтворення відеофайлів необхідна наявність персонального комп'ютера.

А.3.5 Вимоги до складу та параметрів технічний засобів

Для роботи програмного забезпечення для відтворення відеофайлів потрібні такі технічні та програмні ресурси: наявність процесора з тактовою частотою не нижче 2,5 ГГц із підтримкою багатопотокової обробки, що дозволяє забезпечити своєчасне декодування та відображення відеопотоків у реальному часі. Мінімальний обсяг оперативної пам'яті має становити не менше 4 ГБ, хоча для стабільної роботи з відео високої чіткості рекомендовано щонайменше 8 ГБ. Відеокарта з підтримкою апаратного прискорення відео (наприклад, через технології NVDEC, VA-API або Quick Sync) значно підвищує продуктивність, особливо при роботі з форматом H.264 або новішими кодеками. У якості програмного середовища слід використовувати операційну

систему з підтримкою багатозадачності. Для забезпечення належної роботи всіх компонентів необхідна наявність встановленого мультимедійного фреймворку FFmpeg, який відповідає за обробку, декодування та масштабування відео. Крім того, обов'язковою є підтримка бібліотек Qt версії не нижче 5.12, оскільки інтерфейс користувача реалізовано з використанням цього фреймворку. Наявність вільного простору на жорсткому диску, щонайменше 500 МБ, потрібна для встановлення програми та збереження тимчасових файлів, включно з буферами декодованого відео та аудіо.

А.3.6 Вимоги до маркування і пакування

Програма для відтворення відеофайлів може бути записана на будь-якому носії інформації.

На пакуванні повинна бути назва програми – «Програмне забезпечення для відтворення відеофайлів».

А.4 Вхідні дані до роботи

Вхідними даними до програмного забезпечення для відтворення відеофайлів є відеофайли різних форматів, які містять закодовані аудіо- та відеопотоки.

Вихідними даними програмного забезпечення для відтворення відеофайлів є декодовані зображення кадрів, які передаються графічному інтерфейсу для виведення на екран, а також звукові дані, що направляються в аудіосистему пристрою для відтворення.

ДОДАТОК Б
Фрагмент тексту програми

```

void QEdgeAufdDkdr::Dkdr(AVPket *pkt)
{
    int rslt = -1;
    rslt = avcdc_send_pkt(mkdc_kntx.get(), pkt);

    if(rslt < 0)
    {
        OnFailed(QString("Помилка: %1").arg(rslt));
        return;
    }

    while(rslt >= 0)
    {
        AVFrem* frem = bvfremalloc();
        rslt = avcdc_receive_frem(mkdc_kntx.get(), frem);

        if(rslt == AERROR(EAGAIN) || rslt == AERROR_EOF)
        {
            bvfremunref(frem);
            bvfremfree(&frem);
            break;
        }

        else if(rslt < 0)
        {
            bvfremunref(frem);
            bvfremfree(&frem);
            QString errtxt = "Помилка";
            OnFailed(errtxt);
            return;
        }

        OnNewFrem(frem);
    }
}

void QEdgeBfrizedKntner::Stop()
{
    std::lck_guard<std::mtx> lck(mrwmtx);
    Q_UNUSED(lck);

    meof = false;
    FreeBfr();
    close();
    mbfsz = 0;
}

void QEdgeBfrizedKntner::FreeBfr()
{
    if(mbfr)
    {
        free(mbfr);
    }
}

```

```

        mbfr = nullptr;
    }
}

qint64 QEdgeBfrizedKntner::readDany(char *dany, qint64 len)
{
    std::lck_guard<std::mtx> lck(mrwmtx);
    Q_UNUSED(lck);

    qint64 ttl = 0;

    if(meof)
    {
        emt eof();
        return 0;
    }

    if(len >= mbfsz)
    {
        ttl = mbfsz;
        mcpy(dany, mbfr, szt(ttl));
        meof = true;
    }

    else if(mbfsz > 0)
    {
        ttl = len;
        mcpy(dany, mbfr, szt(len));
        mbfsz -= len;
        uintot* bfr = (uintot*)malloc(mbfsz);
        mcpy(bfr, mbfr + len, mbfsz);

        FreeBfr();
        mbfr = bfr;
    }

    return ttl;
}

qint64 QEdgeBfrizedKntner::writeDany(const char *dany, qint64 len)
{
    std::lck_guard<std::mtx> lck(mrwmtx);
    Q_UNUSED(lck);

    uintot* bfr = (uintot*)malloc(mbfsz + len);
    mcpy(bfr, mbfr, szt(mbfsz));
    mcpy(bfr + mbfsz, dany, szt(len));

    FreeBfr();

    mbfr = bfr;

    mbfsz += len;
}

```

```

        return len;
    }

qint64 QEdgeBfrizedKntner::bytesAvailable() const
{
    return mbfsz;
}

QEdgeAufdReproducer::QEdgeAufdReproducer() :
    mplr(CNullPlayer::Instance()),
    maudvyhd_initialized(false)
{
    connect(&maud_bfr, SIGNAL(eof()), this, SLOT(OnBfrEof()));
}

QEdgeAufdReproducer::~QEdgeAufdReproducer()
{
    maudvyhd.release();
    mplr.release();
}

void QEdgeAufdReproducer::Stop()
{
    if(maudvyhd.get())
    {
        maudvyhd->stop();
    }

    maudvyhd.release();

    maud_bfr.Stop();

    maudvyhd_initialized = false;
}

void QEdgeAufdReproducer::OnSetVolume(int volume)
{
    if(maudvyhd.get())
    {
        maudvyhd->setVolume(volume / 100.0);
    }
}

void QEdgeAufdReproducer::OnAufd(AVFrem *aufd_frem)
{
    int smplszbts = bvgetbts_per_smpl((AVSmplFormat)aufd_frem->format);
    maud_bfr.writeDany((char*)aufd_frem->dany[0], smplszbts *
aufd_frem->nb_smpls);

    if(!maudvyhd_initialized)
    {
        mfrmt.setSmplRate(aufd_frem->smpl_rate / aufd_frem->channels);
    }
}

```

```

        mfrmt.setChannelCount(aufd_frem->channels);
        mfrmt.setSmplSz((smplszbts * 8));
        mfrmt.setByteOrder(QAufdFormat::LittleEndian);
        mfrmt.setSmplType(QAufdFormat::Float);
        mfrmt.setKdc("aufd/pcm");

        QCoreApplication::postEvent(this, new QAufdVyhdEvent(true));

        maudvyhd_initialized = true;
    }

    mplr->AufdPrcsd(aufd_frem);
}

bool QEdgeAufdReproducer::event(QEvent* ev)
{
    if(ev->type() == QAufdVyhdEvent::s_aufdvyhd_event_type)
    {
        QAufdVyhdEvent* aufd_ev = static_cast<QAufdVyhdEvent*>(ev);

        if(aufd_ev->IsStart())
        {
            maudvyhd.release();
            maudvyhd.reset(new
QAufdVyhd(QAufdDeviceInfo::defaultVyhdDevice(), mfrmt, this));
            maudvyhd->setNotifyInterval(100);
            maudvyhd->start(&maud_bfr);
            connect(maudvyhd.get(), SIGNAL(notify()), this,
SLOT(OnBfrRead()));
        }

        else
        {
            Stop();
        }

        return true;
    }

    return QObject::event(ev);
}

void QEdgeDkdrBase::Init(AVStrm *strm, IDkdrSbscrer* sbscrer)
{
    if(!mrnng)
    {
        Q_ASSERT(strm);
        mstrm = strm;

        if(msbscrer.get())
        {
            msbscrer.release();
        }
    }
}

```

```

msbscrer.reset(sbscrer);

AvcdcParameters* kdc_prmrs = mstrm->kdcpar;
Avcdc* kdc = avcdc_find_dkdr(kdc_prmrs->kdc_id);

if(mkdc_kntx.get())
{
    FreeKdcKntx();
}

mkdc_kntx.reset(avcdc_alloc_kntx3(kdc));
avcdcprmtr_to_kntx(mkdc_kntx.get(), kdc_prmrs);

int erkd = avcdc_open2(mkdc_kntx.get(), kdc, 0);

if(erkd < 0 || !mkdc_kntx.get())
{
    msbscrer->OnDkdrFailed(this, QString("Error").arg(erkd));
    return;
}

mdmxr_fnshd = false;
mrnng = true;

if(mdkdr_thread.get())
{
    mdkdr_thread.release();
}

mdkdr_thread.reset(new std::thread(QEdgeDkdrBase::DkdrThread,
this));
}
}

void QEdgeDkdrBase::StopDkdr(bool intrpt)
{
    if(intrpt)
    {
        mrnng = false;

        if(mdkdr_thread.get())
        {
            mdkdr_thread->join();
        }

        mdkdr_thread.release();
        FreeKdcKntx();
        Frchrg();
    }

    else
    {

```

```

        mdmxr_fnshd = true;
    }
}

void QEdgeDkdrBase::Frchrg()
{
    while(!mpketchrg.IsEmpty())
    {
        AVPket* pkt = mpketchrg.Pop();

        if(pkt)
        {
            bvpketunrf(pkt);
            bvpketfr(&pkt);
        }
    }
}

void QEdgeDkdrBase::OnNewPket(AVPket *pkt)
{
    if(!mrnng)
    {
        Q_ASSERT(0);
        msbscrer->OnDkdrFailed(this, QString("Dkdr not started"));
        bvpketunrf(pkt);
        bvpketfr(&pkt);
        return;
    }

    else
    {
        PushPket(pkt);
    }
}

void QEdgeDkdrBase::PushPket(AVPket* pkt)
{
    while(mpketchrg.Sz() >= QEdgePketQueue::s_maxsz)
    {
        utis::QEdgeSleep(10);
    }

    mpketchrg.Push(pkt);
}

void QEdgeDkdrBase::OnNewFrem(AVFrem *frem)
{
    msbscrer->OnNewFrem(this, frem);
}

void QEdgeDkdrBase::OnFailed(QString errtxt)
{
    msbscrer->OnDkdrFailed(this, errtxt);
}

```

```

    Q_ASSERT(0);
}

void QEdgeDkdrBase::OnThreadFnshd()
{
    mrnng = false;
    msbscrer->OnDkdrFnshd(this);
    QCoreApplication::postEvent(this, new QOchystEvent());
}

bool QEdgeDkdrBase::event(QEvent *ev)
{
    if(ev->type() == QOchystEvent::s_ochyst_event_type)
    {
        StopDkdr(true);
        return true;
    }

    return QObject::event(ev);
}

void QEdgeDkdrBase::FreeKdcKntx()
{
    AvcdcKntx* ctvx = mkdc_kntx.get();
    avcdc_close(ctvx);
    avcdcfr_kntx(&ctvx);
    mkdc_kntx.release();
}

AVPket *QEdgeDkdrBase::PopPket()
{
    if(mpketchg.IsEmpty())
    {
        return nullptr;
    }

    return mpketchg.Pop();
}

void QEdgeDkdrBase::DkdrThread(void *ctvx)
{
    QEdgeDkdrBase* htser = static_cast<QEdgeDkdrBase*>(ctvx);

    while (htser->mrnng)
    {
        AVPket* pket = htser->PopPket();

        if(!pket)
        {
            utis::QEdgeSleep(20);
            continue;
        }
    }
}

```

```

        htser->Dkdr(pkt);

        bvpketunrf(pkt);
        bvpketfr(&pkt);

        if(htser->IsQueueEmpty() && htser->mdmxr_fnshd)
        {
            htser->OnThreadFnshd();
            return;
        }
    }
}

void QEdgeDmxr::Init(IDmxrSbscrer* sbscrer)
{
    msbscrer = sbscrer;
}

bool QEdgeDmxr::Start(QString fgrl, int skms)
{
    if(fgrl.isEmpty() || mrnng)
    {
        Q_ASSERT(0);
        return false;
    }

    mfgrl = fgrl;
    mrnng = true;

    mskms = skms;

    mthrd.reset(new std::thread(QEdgeDmxr::DemuxInThread, this));

    return true;
}

void QEdgeDmxr::Intrpt()
{
    {
        mrnng = false;
        if(mthrd.get())
        {
            mthrd->join();
        }
        mthrd.release();
    }
}

void QEdgeDmxr::Seek(int msec)
{
    Intrpt();
    Start(mfgrl, msec);
}

void QEdgeDmxr::OnFailed(QString errtxt)

```

```

{
    msbscrer->OnDmxrFailed(errtxt);
}

void QEdgeDmxr::OnFnshd()
{
    mrnng = false;
    mthrd.release();
    msbscrer->OnDmxrFnshd();
}

void QEdgeDmxr::DemuxInThread(void *ctvx)
{
    QEdgeDmxr* htsr = static_cast<QEdgeDmxr*>(ctvx);

    std::string trgtrl = htsr->mfgrl.toStdString();
    const char* fgrl = trgtrl.c_str();

    AVFormatKntx* frmtkntx = avfirtalloc_kntx();
    int erkd = avfirtopeninpt(&frmtkntx, fgrl, nullptr, nullptr);

    if(erkd < 0)
    {
        QString errtxt = "error " + QString::number(erkd);
        htsr->OnFailed(errtxt);
        return;
    }

    erkd = avfirtfind_strminfo(frmtkntx, nullptr);

    if(erkd < 0)
    {
        QString errtxt = "error " + QString::number(erkd);
        htsr->OnFailed(errtxt);
        return;
    }

    int vidstrminindex = -1;
    int aufd_strminindex = -1;
    AVStrm* vidstrm = nullptr;
    AVStrm* aufd_strm = nullptr;

    for(uint i = 0; i < frmtkntx->nb_strms; ++i)
    {
        if(frmtkntx->strms[i]->kdcpar->kdc_type == AVMEDIA_TYPE_VIDEO)
        {
            vidstrminindex = int(i);
            vidstrm = frmtkntx->strms[i];
        }

        else if(frmtkntx->strms[i]->kdcpar->kdc_type ==
AVMEDIA_TYPE_AUFD)
        {

```

```

        aufd_strmindex = int(i);
        aufd_strm = frmtkntx->strms[i];
    }
}

if(!vidstrm)
{
    QString errtxt = "помилка";
    htsr->OnFailed(errtxt);
    return;
}

if(!aufd_strm)
{
    QString errtxt = "помилка";
    htsr->OnFailed(errtxt);
    return;
}

if(erkd < 0)
{
    QString errtxt = "помилка";
    htsr->OnFailed(errtxt);
    return;
}

htsr->msbscrer->InitStrm(vidstrm, aufd_strm);

if(htsr->mskms)
{
    int res = 0;

    int64_t strmidx = bvfind_default_strmindex(frmtkntx);
    int64_t fremidx = utis::MscsToTimebaseUnits(frmtkntx->strms[strmidx]->time_base, htsr->mskms);

    res = avfrmtseek_file(frmtkntx, strmidx, 0, fremidx, fremidx,
AVSEEK_FLAG_FREM);

    if(res < 0)
    {
        QString errtxt = "помилка";
        htsr->OnFailed(errtxt);
        return;
    }
}

int res = 0;
while(res >= 0 && htsr->mrnng)
{
    AVPket* pket = bvpket_alloc();
    res = bvread_frem(frmtkntx, pket);
}

```

```

        if(pkt->strmindex == vidstrmindex)
        {
            htser->msbscrer->OnVideoPket(pkt);
        }

        else if(pkt->strmindex == aufd_strmindex)
        {
            htser->msbscrer->OnAufdPket(pkt);
        }

        else
        {
            bvpketunrf(pkt);
            bvpketfr(&pkt);
        }
    }

    avfrmtflush(frmtkntx);
    avfrmtcloseinpt(&frmtkntx);
    avformatfr_kntx(frmtkntx);
    htser->OnFnshd();
}

void QEdgeMediaController::Start(QString flnm)
{
    maud_fnshd = false;
    mvidfnshd = false;
    mpresentationstrd = false;

    mdmxr.Init(this);

    if(!mdmxr.Start(flnm))
    {
        OnDmxrFailed("error");
        return;
    }
}

void QEdgeMediaController::VideoFremPrbsd(AVFrem* frem)
{
    bvfremunref(frem);
    bvfremfree(&frem);
}

void QEdgeMediaController::AufdFremPrbsd(AVFrem* frem)
{
    bvfremunref(frem);
    bvfremfree(&frem);
}

void QEdgeMediaController::VideoPrsnt()
{

```

```

        msynchrnznr.OnVideoPrsnt();
    }

void QEdgeMediaController::AufdPrsnt(long long aufd_dany_remains)
{
    msynchrnznr.OnAufdPrsnt(aufd_dany_remains);
}

void QEdgeMediaController::Stop()
{
    mdmxr.Intrpt();
    maud_dkdr. StopDkdr(true);
    mviddkdr. StopDkdr(true);
}

void QEdgeMediaController::Seek(int64_t msec)
{
    maud_fnshd = false;
    mvidfnshd = false;
    mdmxr.Seek(msec);
}

void QEdgeMediaController::OnDkdrFnshd(IDkdr *sender)
{
    if(sender == &maud_dkdr)
    {
        maud_fnshd = true;
    }

    if(sender == &mviddkdr)
    {
        mvidfnshd = true;
    }

    if(mvidfnshd && maud_fnshd)
    {
        msbscrer->OnPlayFnshd();
    }
}

void QEdgeMediaController::OnDkdrFailed(IDkdr *sender, QString
errtxt)
{
    if(sender == &mviddkdr)
    {
        qDebug() << "error" << errtxt;
    }
    else if(sender == &maud_dkdr)
    {
        qDebug() << "error" << errtxt;
    }
    else
    {

```

```

        Q_ASSERT(0);
    }
}

void QEdgeMediaController::OnNewFrem(IDkdr *sender, AVFrem *frem)
{
    if(sender == &mviddkdr)
    {
        msynchrnzs.ProcessVideo(frem);
    }
    else if(sender == &maud_dkdr)
    {
        msynchrnzs.ProcessAufd(frem);
    }
}

void QEdgeMediaController::OnDmxrFailed(QString errtxt)
{
    qDebug() << "error" << errtxt;
}

void QEdgeMediaController::OnDmxrFnshd()
{
    mviddkdr.StopDkdr(false);
    maud_dkdr.StopDkdr(false);
}

void QEdgeMediaController::InitStrm(AVStrm *vidstrm, AVStrm *aufd_strm)
{
    msynchrnzs.SessionStarted(vidstrm, aufd_strm);
    maud_dkdr.Init(aufd_strm, this);
    mviddkdr.Init(vidstrm, this);

    msbscrer->DurationSpecified(utis::TimebaseUnitsToMscs(vidstrm->time_base, vidstrm->duration));
}

void QEdgeMediaController::OnAufdPket(AVPket *pket)
{
    maud_dkdr.OnNewPket(pket);
}

void QEdgeMediaController::OnVideoPket(AVPket *pket)
{
    mviddkdr.OnNewPket(pket);
}

void QEdgePlayer::InitPlayer(IPlayerKlnt *vidrcvr, IPlayerKlnt*
aufdrvr)
{
    mvidrcvr.reset(vidrcvr);
    maudrcvr.reset(aufdrvr);
}

```

```

void QEdgePlayer::DisconnectFromPlayer(IPlayerKlnt *klnt)
{
    if(klnt == maudrcrvr.get())
    {
        maudrcrvr.reset(CNullPlayerKlnt::Instance());
    }

    if(klnt == mvidrcvr.get())
    {
        mvidrcvr.reset(CNullPlayerKlnt::Instance());
    }
}

void QEdgePlayer::Start(QString flnm)
{
    mflnm = flnm;
    mmediakntrlr.Start(flnm);
    mvidrcvr->PlayerStarted();
    maudrcrvr->PlayerStarted();
}

void QEdgePlayer::Stop()
{
    mmediakntrlr.Stop();
    mvidrcvr->PlayerStopped();
    maudrcrvr->PlayerStopped();
}

void QEdgePlayer::Seek(int64_t msec)
{
    mmediakntrlr.Stop();
    maudrcrvr->PlayerStopped();
    maudrcrvr->PlayerStarted();
    mmediakntrlr.Seek(msec);
}

void QEdgePlayer::Pause()
{
    mpaused = true;
    Stop();
}

void QEdgePlayer::ContinuePlay()
{
    maudrcrvr->PlayerStopped();
    maudrcrvr->PlayerStarted();
    mmediakntrlr.Seek(mlast_tmstamp);
    mpaused = false;
}

void QEdgePlayer::OnPlayFnshtd()
{

```

```

        mvidrcvr->OnFnshd();
        maudrcrvr->OnFnshd();
    }

void QEdgePlayer::AufdPrcsd(AVFrem *frem)
{
    mmediakntrlr.AufdFremPrcsd(frem);
}

void QEdgePlayer::VideoPrcsd(AVFrem *frem)
{
    mmediakntrlr.VideoFremPrcsd(frem);
}

void QEdgePlayer::AufdPrsnt(long long aufd_dany_remains)
{
    mmediakntrlr.AufdPrsnt(aufd_dany_remains);
}

void QEdgePlayer::VideoPrsnt()
{
    mmediakntrlr.VideoPrsnt();
}

void QEdgePlayer::DurationSpecified(int64_t mscs)
{
    maudrcrvr->DurationSpecified(mscs);
    mvidrcvr->DurationSpecified(mscs);
}

void QEdgePlayer::CurrentTmstmpChanged(int64_t mscs)
{
    if(!mpaused)
    {
        mlast_tmstmp = mscs;
        maudrcrvr->CurrentTmstmpChanged(mscs);
        mvidrcvr->CurrentTmstmpChanged(mscs);
    }
}

void QEdgePlayer::OnFailed(QString errtxt)
{
    mvidrcvr->OnFailed(errtxt);
    maudrcrvr->OnFailed(errtxt);
}

void QEdgePlayer::OnNewVideoFrem(AVFrem* frem)
{
    mvidrcvr->OnVideo(frem);
}

void QEdgeSynchronizer::SessionStarted(const AVStrm *vidstrm, const
AVStrm *aufd_strm)

```

```

{
    maud_prmrs.time_base = vidstrm->time_base;
    mvidprmrs.time_base = vidstrm->time_base;
    maud_prmrs.format = auafd_strm->kdcpar->format;
    mvidprmrs.format = vidstrm->kdcpar->format;
    maud_prmrs.smpl_rate = auafd_strm->kdcpar->smpl_rate;
    mvidprmrs.smpl_rate = vidstrm->kdcpar->smpl_rate;
    maud_prmrs.channels = auafd_strm->kdcpar->channels;

    mvidrate = 0;
    maud_rate = 0;
    mnextdlk = BVNOSYNCDlk;
    mfremtimer = 0;
    mfremlastdlk = 0;
    mfremlast_pftg = 0;
    mcurrent_frem_pftg = 0;
    maud_remains = 0;
    maud_bfr_full = false;
    mvidstarted = false;
    maudstrd = false;
}

void QEdgeSynchronizer::ProcessVideo(AVFrem *frem)
{
    UpdateVideoRate(frem);
    PushNextFrem(frem);
}

void QEdgeSynchronizer::OnAuafdPrsnt(long long bfr_remains)
{
    maud_remains = bfr_remains;
    if(!maudstrd)
    {
        std::lck_guard<std::mtx> auafdstrd_lck(maud_wait_mtx);
        maudstrd = true;
        maudstrd_condition.notify_one();
        Q_UNUSED(auafdstrd_lck);
    }

    maud_bfr_full = bfr_remains >= BVMAX_AUAFD_BUFsz;
}

void QEdgeSynchronizer::PushNextFrem(AVFrem* frem)
{
    if(!maudstrd)
    {
        std::unique_lck<std::mtx> auafdstrd_lck(maud_wait_mtx);
        maudstrd_condition.wait(auafdstrd_lck);
        auafdstrd_lck.unlock();
    }

    mrcvr->OnNewVideoFrem(frem);
}

```

```

int64_t pftg = frem->pftg;
if(pftg == BVNOPFTG_VALUE)
{
    pftg = frem->pkt_dts;
}
pftg = utis::TimebaseUnitsToMscs(mvidprmrs.time_base, pftg);

if(pftg == BVNOPFTG_VALUE)
{
    pftg = mcurrent_frempftg * 1000;
}

mrcvr->CurrentTmstpChanged(pftg );
utis::QEdgeSleep(GetMsZatrmk(mnextdlk));
}

void QEdgeSynchronizer::ProcessAufd(AVFrem *frem)
{
    UpdateAufdRate(frem);

    while(maud_bfr_full)
    {
        utis::QEdgeSleep(10);
    }

    mrcvr->OnNewAufdFrem(frem);
}

double QEdgeSynchronizer::GetAufdRate()
{
    double pftg = maud_rate;
    int btspsc = maud_prmrs.smpl_rate *
                maud_prmrs.channels *

bvgetbts_per_smpl((AVSmplFormat)maud_prmrs.format);

    long long bfrsz = maud_remains;

    if(btspsc != 0)
    {
        pftg -= (double)bfrsz / btspsc;
    }

    return pftg;
}

void QEdgeSynchronizer::SyncVideo()
{
    if(!mvidstarted)
    {
        mvidstarted = true;
        msynctmr.restart();
    }
}

```

```

double krntpftg = mcurrent_frempftg;
double zatrmk = krntpftg - mfremlast_pftg;

if(zatrmk <= 0.0 || zatrmk >= 1.0)
{
    zatrmk = mfremlastdlk;
}

mfremlast_pftg = krntpftg;
mfremlastdlk = zatrmk;

double aufd_clck = GetAufdRate();
double difg = krntpftg - aufd_clck;
double sncthrshld = std::max(BVSNCTHRSHLD, zatrmk);

if(fabs(difg) < BVNOSNCTHRSHLD)
{
    if(difg <= -sncthrshld)
    {
        zatrmk = 0;
    }

    else if(difg >= sncthrshld)
    {
        zatrmk *= 2;
    }
}

mfremtimer += zatrmk;

double actualdlk = mfremtimer - (double)msynctmr.elapsed() /
1000.0;

if(actualdlk < 0.010)
{
    actualdlk = 0.010;
}

mnextdlk = actualdlk;
}

void QEdgeSynchronizer::UpdateAufdRate(AVFrem *frem)
{
    if(frem->pftg != BVNOPFTG_VALUE)
    {
        maud_rate += bvq2d(maud_prmrs.time_base) * frem->pftg;
    }

    else
    {
        maud_rate += frem->nb_smpls / frem->channels * frem->smpl_rate;
    }
}

```

```

}

void QEdgeSynchronizer::UpdateVideoRate(AVFrem *frem)
{
    double krntpftg = frem->pftg;

    if(krntpftg == BVNOPFTG_VALUE)
    {
        krntpftg = frem->pkt_dts;
    }

    if(krntpftg == BVNOPFTG_VALUE)
    {
        krntpftg = 0;
    }

    krntpftg *= bvq2d(mvidprmrs.time_base);

    if(krntpftg != 0)
    {
        mvidrate = krntpftg;
    }

    else
    {
        krntpftg = mvidrate;
    }

    double fremzatrmk = bvq2d(mvidprmrs.time_base);
    fremzatrmk += frem->repeat_pict * (fremzatrmk * 0.5);
    mvidrate += fremzatrmk;

    mcurrent_frempftg = krntpftg;
}

```

ДОДАТОК В
Слайди презентації

Міністерство освіти та науки України
Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Програмне забезпечення для відтворення відеофайлів

Виконав: Денис ІВАНІН

студент групи КНТ-141

Керівник: Михайло КОЦУР

к.т.н., доцент НУ «Запорізька політехніка»

Рисунок В.1 – Слайд 1

2

Об'єкт, предмет та мета роботи

Об'єкт дослідження – процес розробки програмного забезпечення для відтворення відеофайлів.

Предмет дослідження – програмні засоби для підтримки процесу відтворення відеофайлів.

Метою роботи є розробка програмного забезпечення для відтворення відеофайлів.

Рисунок В.2 – Слайд 2

Завдання роботи

Для досягнення поставленої мети у кваліфікаційній роботі бакалавра необхідно розв'язати такі задачі:

- виконати аналіз предметної області та програмних засобів для відтворення відеофайлів;
- здійснити проєктування програмного забезпечення для відтворення відеофайлів;
- створити програмне забезпечення для відтворення відеофайлів;
- виконати тестування розробленого програмного забезпечення для відтворення відеофайлів.

Рисунок В.3 – Слайд 3

Порівняння існуючих аналогів

Критерій порівняння	Plex	PotPlayer	KMPlayer
Підтримка багатьох форматів файлів	+—	+	+
Можливість редагування відео	—	+	+—
Сумісність із хмарними сервісами	+	—	+
Наявність мобільної версії	+	—	+
Інтеграція з онлайн контентом	+	—	+—
Простота використання	+	+—	+—

Рисунок В.4 – Слайд 4

Вимоги до програмного забезпечення

При розробці програмного забезпечення для відтворення відеофайлів необхідно забезпечити такі функціональні вимоги:

- підтримка можливості завантаження відеофайлу для подальшого відтворення;
- підтримка можливості відтворення відеофайлів;
- можливість зупинення процесу відтворення відеофайлів;
- можливість регулювання гучності у процесі відтворення відеофайлу;
- підтримка можливості перемотування відео у процесі відтворення.

Рисунок В.5 – Слайд 5

Порівняння мов програмування

Критерій порівняння мов програмування	Мова програмування		
	C++	Python	Java
Висока продуктивність	+	–	+–
Підтримка мультимедійних бібліотек	+	+–	+–
Оптимізація апаратних ресурсів	+	–	+
Зручність для створення ПЗ для відтворення відеофайлів	+	–	–
Масштабованість і підтримка	+	+–	+

Рисунок В.6 – Слайд 6

Порівняння середовищ розробки

Критерій порівняння середовищ розробки	Середовища розробки		
	Qt Creator	Visual Studio	CLion
Кросплатформеність	+	+-	+
Зручність візуального проектування інтерфейсів	+	+-	-
Підтримка багатоплатформених проектів	+	+-	+
Швидкість роботи	+	+-	+-
Інтеграція з системами контролю версій	+	+	+-

Рисунок В.7 – Слайд 7

Структура ПЗ для відтворення відеофайлів



Рисунок В.8 – Слайд 8

Схема функціонування ПЗ

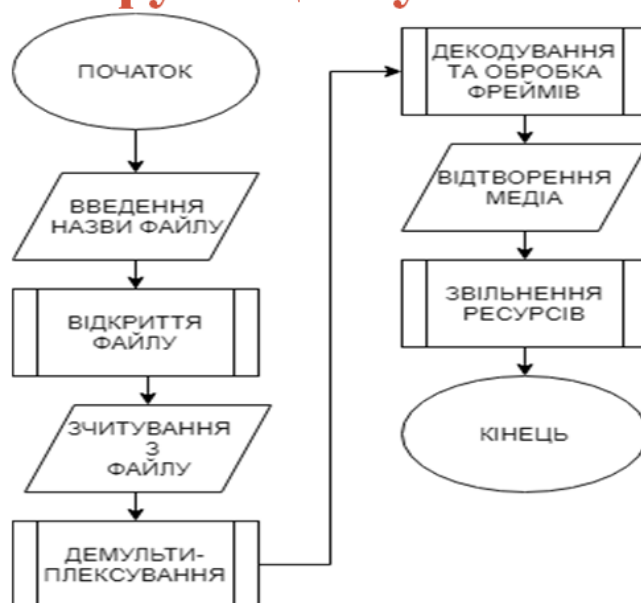


Рисунок В.9 – Слайд 9

Алгоритм декодування у ПЗ для відтворення відеофайлів



Рисунок В.10 – Слайд 10

Інтерфейс програми

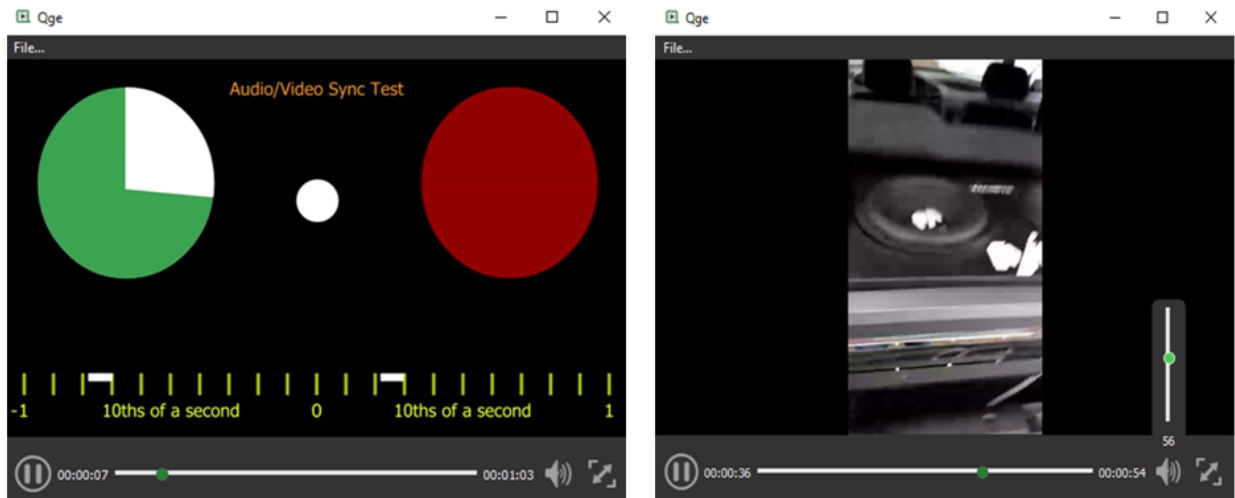


Рисунок В.11 – Слайд 11

Висновки

В ході виконання дипломної кваліфікаційної роботи бакалавра було проаналізовано та досліджено процес розробки програмного забезпечення для відтворення відеофайлів.

Для реалізації програмного забезпечення для відтворення відеофайлів обрано мову програмування C++ та середовище розробки Qt Creator.

Архітектура програмного забезпечення для відтворення відеофайлів ґрунтується на поділі всієї системи на шість окремих функціональних модулів (модуль завантаження відеоданих, модуль декодування, модуль синхронізації медіа, модуль відтворення відео, модуль аудіо виходу, модуль управління відтворенням), кожен з яких відповідає за певний етап обробки та виведення мультимедійного контенту.

В результаті виконання завдання було отримано програму, яка відповідає вимогам, представленим в технічному завданні, зручна для використання та виконує всі необхідні функції.

Усі завдання випускної дипломної роботи бакалавра повністю виконано.

Рисунок В.12 – Слайд 12