НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри

Комп'ютерних наук

_____ Голуб Б.Л.

_____ " " _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему

Програмне забезпечення системи перегляду медіафайлів

Спеціальність 121 — «Інженерія програмного забезпечення»

Гарант освітньої програми

К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи

к.т.н., доцент _____ Бородкіна І.Л.

(науковий ступень та вчене звання) підпис (ПІБ)

Виконвав _____ Вакуленко Д.О.

(підпис) (ПІБ студента)

КИЇВ — 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук
_____ Голуб Б.Л.
" " _____ 2025 р.

З А В Д А Н Н Я
на виконання бакалаврської кваліфікаційної роботи студенту
Вакуленку Данилу Олеговичу

Спеціальність 121 — «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення системи перегляду медіафайлів

затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення, навчальна та методична література, державні стандарти.

Перелік питань що розглядаються:

1. Аналіз проблемної області.
2. Вибір та обґрунтування засобів для розробки системи.
3. Проектування інформаційної системи.
4. Висновки.

Дата видачі завдання " " _____ 2025 р.

Керівник бакалаврської кваліфікаційної роботи

к.т.н., доцент _____ Бородкіна І.Л.
(науковий ступень та вчене звання) підпис (ПІБ)

Завдання прийняв до виконання _____ Вакуленко Д. О.
(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- UI — User Interface, користувацький інтерфейс.
- QML — Qt Meta-object Language, мова розмітки інтерфейсу.
- Libav — набір бібліотек для обробки медіа.
- Codec — алгоритм, який стискає (кодує) і розпаковує (декодує) аудіо або відео. Наприклад: H.264, AAC.
- Stream — потік, послідовність аудіо або відеоданих в мультимедійному файлі.
- Packet — пакет, одиниця збережених даних, яка містить закодовану інформацію певної частини потоку (stream).
- Frame — фрейм/кадр, окреме зображення (для відео) або невелика порція звукових даних (для аудіо). Результат декодування пакета.
- Sample — семпл, найменша одиниця цифрового аудіо, яка містить значення амплітуди звукового сигналу.
- FPS — Frames Per Second, кількість відеокадрів за секунду.
- Sample rate — кількість семплів (одиниць звуку) за одну секунду для кожного каналу.
- PTS — мітка часу, у якій послідовності має бути відтворенний фрейм.
- DTS — мітка часу, у якій послідовності має бути декодований фрейм.

ВСТУП

У сучасному світі мультимедійні технології відіграють ключову роль у повсякденному житті людей, освіті, бізнесі, розвагах і науці. Зростає потреба в універсальних програмних засобах, здатних якісно відтворювати аудіо- та відеоконтент з підтримкою широкого спектра форматів і можливостей кастомізації. Огляд наявних програмних рішень свідчить про те, що вони або надто складні для користувача з недружнім інтерфейсом, або не забезпечують належного рівня кастомізації, продуктивності та гнучкості. Це створює потребу в розробці нового, зручного та адаптованого до сучасних вимог медіаплеєра.

Актуальність теми полягає в створенні програмного продукту, який дозволить не лише ефективно відтворювати медіафайли різних форматів, а й надасть користувачу інструменти для гнучкого налаштування параметрів відтворення, таких як швидкість, гучність, відеофільтри, еквалайзер, масштабування тощо. Крім того, враховуючи популярність кросплатформних рішень, важливою є підтримка Windows, Linux та інших операційних систем.

Мета розробки полягає у створенні кросплатформного медіаплеєра з підтримкою відтворення аудіо- та відеофайлів різних форматів, функціями зміни параметрів відтворення та застосування аудіо- й відеоефектів. Розробка такого програмного забезпечення спрямована на усунення обмежень існуючих рішень у функціональності та доступності інтерфейсу для простих користувачів.

Об'єкт дослідження — програмне забезпечення для відтворення мультимедійного контенту, що використовується для перегляду аудіо- та відеофайлів у різних форматах.

Предмет дослідження — функціональні та технічні властивості медіаплеєра: підтримка форматів, стабільність відтворення, налаштування фільтрів, зручність інтерфейсу та розширюваність архітектури.

Задачі дослідження:

- Проаналізувати існуючі медіаплеєри та виявити їхні недоліки;

- Обґрунтувати вибір інструментів і технологій розробки;
- Розробити архітектуру програмного забезпечення;
- Реалізувати функціонал відтворення медіа з можливістю налаштування параметрів;
- Провести тестування і налагодження роботи програми.

Методи та технології, що використовуються в роботі:

- Мова програмування C++. Обрана завдяки високій продуктивності, гнучкості управління пам’яттю, а також широкій підтримці мультимедійних бібліотек. Забезпечує низькорівневий контроль, необхідний для обробки відео та аудіо в реальному часі.
- Qt та QML. Фреймворк Qt забезпечує кросплатформенну підтримку, зручну побудову GUI та взаємодію з C++. QML використовується для створення гнучкого та кастомізованого графічного інтерфейсу з анімаціями. Комбінація Qt, C++ та QML дозволяє ефективно розділяти логіку і дизайн програми
- Libav. Це низькорівнева бібліотека для обробки мультимедіа (декодування, енкодингу, демультиплексування, обробки потоків). Обрана завдяки широкій підтримці медіаформатів, гнучкості й стабільності. Вона дозволяє реалізувати власну логіку обробки потоків та застосування фільтрів.

Основні результати дослідження було представлено на науковій конференції «Теоретичні та прикладні аспекти розробки комп’ютерних систем».

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметна область даної дипломної роботи охоплює розробку програмного забезпечення для відтворення мультимедійного контенту, зокрема відео та аудіофайлів різних форматів. У сучасних умовах цифрової революції та активного розвитку інформаційних технологій мультимедіа є однією з ключових складових практично всіх прикладних систем — від розважальних додатків до науково-освітніх платформ, відеоігр та професійних систем відеообробки.

Сучасні користувачі очікують від медіаплеєрів високої продуктивності, підтримки широкого спектра форматів, простоти у використанні, а також можливості налаштовувати відтворення під власні потреби. Це створює потребу у розробці більш досконалих програмних рішень, які б поєднували у собі кросплатформеність, кастомізацію, гнучкість і простоту використання інтерфейсу.

Розробка медіаплеєра передбачає використання таких основних компонентів:

- Мультимедійні бібліотеки для декодування, обробки та об'єднання відео- і аудіоданих;
- Фреймворки розробки інтерфейсу (Qt/QML) для створення інтерактивного та адаптивного UI;
- Механізми потокової обробки, синхронізації кадрів, роботи з буферами, пакетами та часовими мітками;
- Підтримка відеофільтрів та аудіообробки, які дозволяють змінювати вигляд і звучання медіа в реальному часі.

Особливу актуальність має використання бібліотеки Libav, яка є потужним інструментом для низькорівневої обробки медіа. Вона дозволяє здійснювати прямий контроль над процесом декодування, конвертації, фільтрації, що

забезпечує більшу гнучкість у розробці плеєра, порівняно з використанням готових віджетів.

Водночас Qt і QML дозволяють розробити сучасний, анімований та адаптивний графічний інтерфейс, що є особливо важливим для забезпечення зручності користування програмою. Завдяки цим технологіям медіаплеєр може бути ефективно реалізований як на настільних, так і на мобільних платформах.

Таким чином, предметна область поєднує знання з програмної інженерії, цифрової обробки, графічного дизайну, розробки GUI та системної інтеграції. Дослідження у цій галузі спрямоване на підвищення якості відтворення мультимедійного контенту, оптимізацію ресурсів системи, а також покращення взаємодії користувача з додатком.

1.2 Аналіз вимог до програмної системи

Аналіз вимог — це початковий та важливий етап життєвого циклу розробки програмного забезпечення. На цьому етапі виявляються, формалізуються та документуються очікування користувачів і замовника щодо функціоналу та якості системи. Ретельний аналіз вимог дозволяє чітко зрозуміти, що саме має робити система, як вона має себе поводити та в яких умовах працювати.

Цей етап знижує ризики непорозумінь між замовником і розробником, дозволяє уникнути помилок у пізніших фазах, мінімізує кількість змін і доопрацювань у майбутньому, що в результаті зменшує загальні витрати на розробку.

1.2.1 Функціональні вимоги

Функціональні вимоги описують, що саме повинна робити система, які дії має виконувати користувач і яку поведінку очікується від додатку у відповідь. Ці вимоги прямо випливають з мети проєкту та потреб кінцевого користувача.

Для медіаплеєра основні функціональні вимоги включають:

- Завантаження та відтворення відеофайлів різних форматів.
- Завантаження та відтворення аудіофайлів різних форматів.
- Керування відтворенням: запуск, пауза та зупинка.
- Перемотування вперед/назад.
- Зміна швидкості відвторення (збільшення/зменшення).
- Зміна гучності відтворення (збільшення/зменшення).
- Зміна масштабів відтворення, перемикання між повноекранним та звичайним режимами.
- Можливість приближення/віддалення елементів відео.
- Налаштування відеофільтрів у реальному часі.
- Налаштування аудіоеквалайзеру у реальному часі.

Ці вимоги відповідають основним очікуванням користувача, який бажає мати зручний і функціональний інструмент для перегляду медіа.

1.2.2 Нефункціональні вимоги

Нефункціональні вимоги описують якісні характеристики системи: наскільки вона зручна, швидка, безпечна тощо. Вони не описують функції напряму, але регламентують обмеження та умови виконання функцій. Визначено такі нефункціональні вимоги:

- Продуктивність — програвач має забезпечувати плавне відтворення HD та FullHD відео без затримок або фризів, з мінімальним споживанням ресурсів.
- Кросплатформеність — підтримка основних операційних систем: Windows, MacOS, Linux.
- Інтуїтивно зрозумілий інтерфейс — користувач повинен легко орієнтуватися в управлінні без потреби в інструкції.

- Модульність архітектури — можливість розширення системи в майбутньому.

1.2.3 Системні вимоги

Системні вимоги — це обмеження, що стосуються середовища розробки, виконання, сумісності та інструментів, які повинні бути використані. У межах даного проєкту було визначено наступні технічні вимоги:

- Мова програмування — C++, оскільки вона дозволяє ефективно працювати з пам’яттю та має доступ до низькорівневих мультимедійних бібліотек.
- Фреймворк — Qt 6, який підтримує розробку GUI з використанням QML для сучасного адаптивного інтерфейсу.
- Бібліотека для роботи з медіа — Libav, що забезпечує обробку відео та аудіо на низькому рівні.

1.3 Моделювання предметної області

Моделювання предметної області — це процес побудови абстрактного уявлення (моделі) про реальний об'єкт або процес, який підлягає дослідженню чи автоматизації. Основна мета цього процесу — краще зрозуміти структуру, елементи та взаємозв’язки в системі до початку її реалізації. Це дозволяє знизити ризики проєктування та уникнути логічних помилок. Найбільш поширеним інструментом моделювання є мова UML (Unified Modeling Language), яка забезпечує універсальний засіб для створення моделей системи на різних рівнях абстракції [1].

Модель предметної області є спрощеним уявленням про те, як має функціонувати система, які сутності в ній існують і як вони між собою взаємодіють.

Мета моделювання:

- виявити ключові об'єкти, їхні атрибути та дії;
- зрозуміти логіку взаємозв’язків між компонентами системи;
- створити основу для архітектурного проєктування ПЗ;
- полегшити комунікацію між замовником і розробником.

1.3.1 Діаграма прецедентів

Діаграма прецедентів використовується для візуального представлення функціональних вимог до системи. Основна мета цієї діаграми — показати, які дії може виконувати користувач (або зовнішня система) при взаємодії із системою, та як ці дії пов’язані між собою.

Основне призначення діаграми класів полягає у відображенні:

- структури системи з позиції об’єктів, що її складають;
- взаємозв’язків між елементами програмної системи;
- механізмів повторного використання коду;
- залежностей між класами для подальшої реалізації в коді.

Компоненти діаграми:

- Актор — користувач або інша система, яка взаємодіє із системою.
- Прецедент — функціональна можливість або сценарій, який система надає користувачеві.
- Зв’язки — асоціації між акторами та прецедентами.
- Межі системи — рамки, які показують, що саме входить до системи.

Розглянемо діаграму прецедентів системи перегляду медіафайлів представлену на рис. 1.

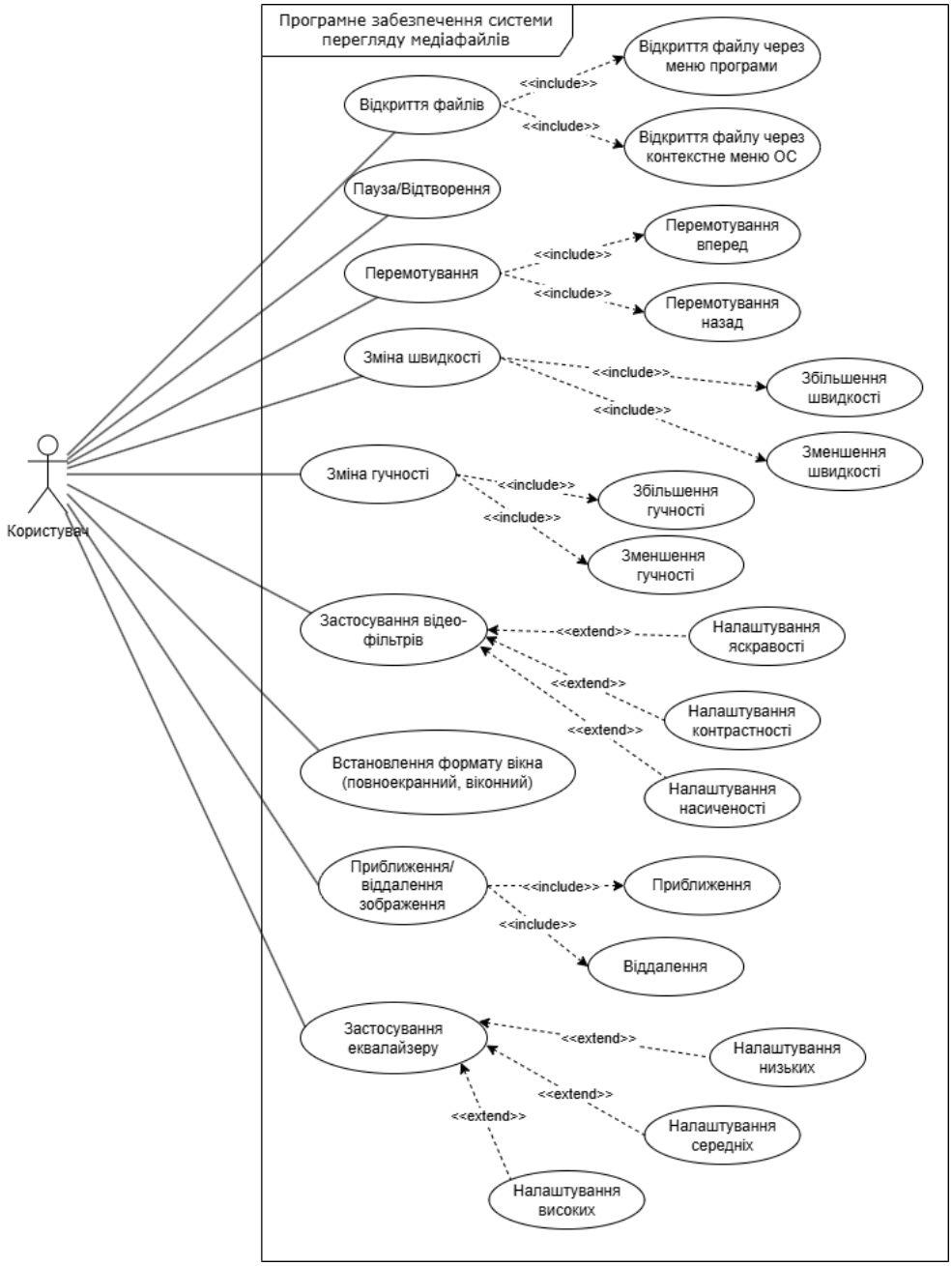


Рис. 1 – Діаграма прецедентів системи перегляду медіафайлів

На діаграмі представленний один єдиний актор – Користувач. Та перелік доступних для нього прецедентів:

- Відкриття файлів, що включає в себе відкриття файлу через меню програми або відкриття файлу через контекстне меню операційної системи.
- Пауза або відтворенням медіафайлу.

- Перемотування, що включає в себе або перемотування вперед, або перемотування назад.
- Зміна швидкості, що включає в себе збільшення або зменшення швидкості відтворення.
- Зміна гучності, що включає в себе збільшення або зменшення гучності відтворення аудіо.
- Застосування відеофільтрів, що дозволяє налаштовувати величину яскравості, контрастності та насиченості.
- Встановлення формату вікна, а саме встановлення віконного або повноекранного режиму вікна.
- Налаштування еквалайзеру, що дозволяє коригувати низькі, середні та високі частоти аудіо.

Діаграма прецедентів дозволяє ідентифікувати основні функціональні можливості системи, надаючи зрозумілу і узагальнену картину її призначення. Вона є особливо корисною на початкових етапах проєктування для узгодження вимог з кінцевими користувачами або замовниками.

1.3.2 Діаграма класів

Діаграма класів відображає структурну модель системи, засновану на об’єктно-орієнтованому підході. Вона дозволяє визначити основні класи системи, їхні атрибути, методи та взаємозв’язки між ними. Це одна з найважливіших діаграм для проєктування архітектури програмного забезпечення, оскільки вона прямо впливає на структуру коду.

До основних елементів діаграми належать:

- Клас — прямокутник, що містить ім’я класу, список атрибутів та методів.
- Атрибути — властивості, що характеризують стан об'єкта класу.

- Методи — функціональність, яку реалізує клас.
- Зв’язки між класами. На даному етапі проєктування лише зв’язки асоціацій (один до одного, один до багатьох).

Розглянемо діаграму класів системи перегляду медіафайлів представлену на рис. 2.

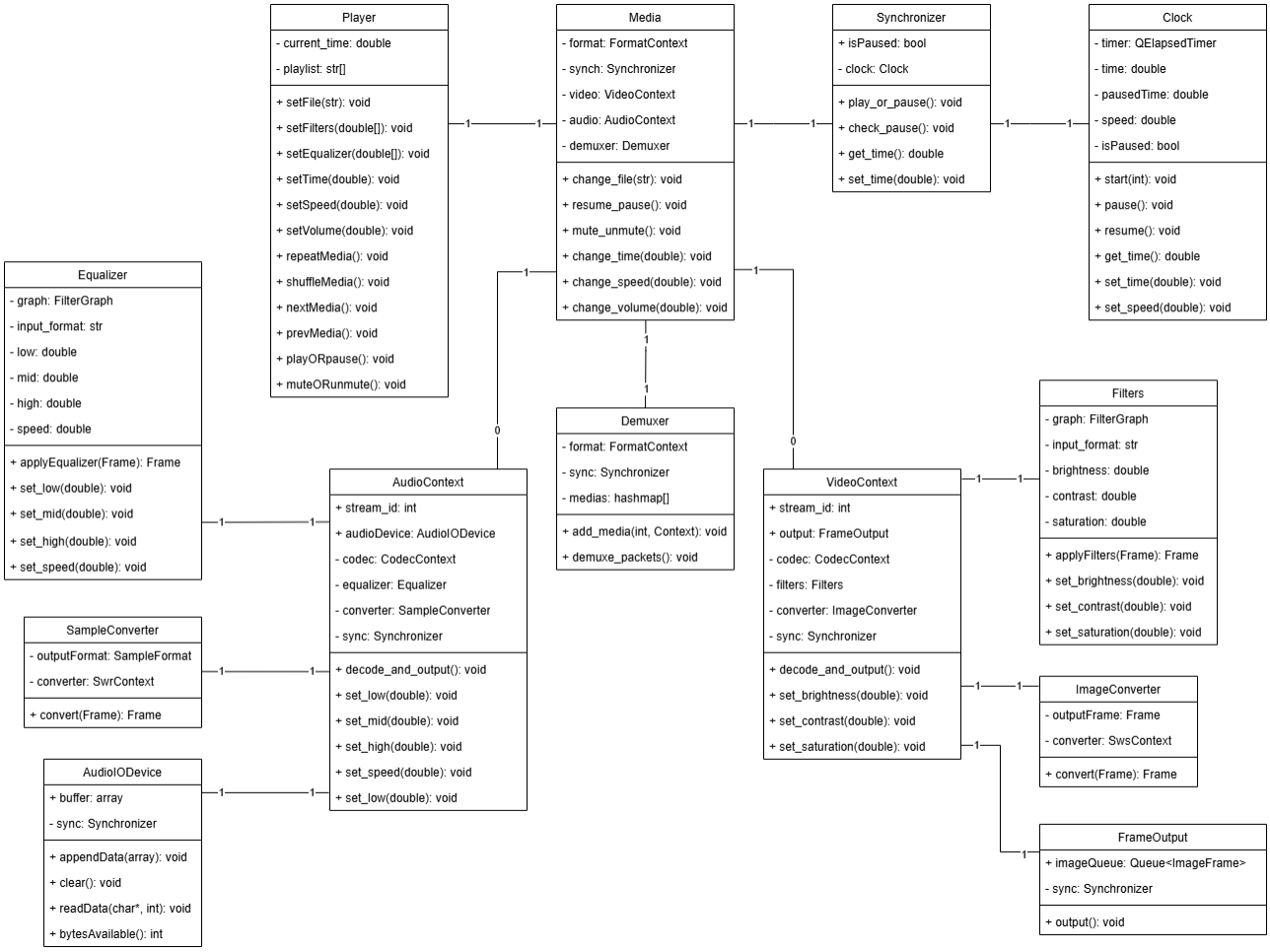


Рис. 2 – Діаграма класів системи перегляду медіафайлів

На діаграмі представлено 13 класів:

- Player – клас взаємодії інтерфейсної та логічної частини програми. Клас приймає сигнали від UI-інтерфейсу та передає їх Media класу.
- Media – головний клас логічної частини програми. Керує відтворенням, створенням та налаштуванням відео та аудіо класів.

- Synchronizer – клас, відповідальний за синхронізацію відтворення відео та аудіо.
- Clock – клас, що зберігає поточний стан та час відтворення.
- Demuxer – клас, відповідальний за розбиття медіафайлу на відео- та аудіо-пакети.
- VideoContext – клас, відповідальний за декодування, накладання фільтрів та конвертацію у вихідний формат зображень.
- Filters – клас, відповідальний за накладання фільтрів на декодовані зображення.
- ImageConvertor – клас, відповідальний за конвертацію зображень у вихідний формат.
- FrameOutput – клас, відповідальний за вивід зображень у UI-інтерфейс.
- AudioContext – клас, відповідальний за декодування, накладання аудіофільтрів та конвертацію у вихідний формат аудіо семплів.
- Equalizer – клас, відповідальний за налаштування аудіо частот семплів.
- SampleConvertor – клас, відповідальний за конвертацію аудіо у вихідний формат.
- AudioIODevice – клас, відповідальний за вивід аудіо у пристрій аудіо виводу.

Діаграма класів забезпечує цілісне розуміння структури програмної системи та дозволяє ефективно реалізувати її функціональність за допомогою модульного, масштабованого та повторно використовуваного коду.

1.4 Огляд інформаційних джерел та існуючих рішень

Перед повноцінним створенням системи необхідно виконати аналіз наукових, технічних та практичних джерел, які стосуються предметної області розробки медіаплеєра. Також слід розглянути існуючі програмні рішення, що вже реалізують подібний функціонал. Такий огляд дозволяє виявити актуальні підходи, технології, алгоритми, а також визначити недоліки, які може вирішити запропонована система.

1.4.1 Огляд інформаційних джерел

Під час вивчення предметної області були використані такі основні категорії джерел:

- Наукові статті та публікації, що описують методи декодування мультимедійного контенту, алгоритми обробки аудіо та відео, а також сучасні підходи до реалізації медіасистем.
- Офіційна документація бібліотек та фреймворків, таких як Ffmpeg(libav), Qt, QML, які використовуються для реалізації функціональності відтворення, фільтрації та обробки медіа.
- Досвід спільноти (форумні обговорення, open-source репозиторії GitHub, Stack Overflow), що дозволив глибше зрозуміти практичні аспекти реалізації мультимедійних додатків.

Ці джерела стали основою для формування підходу до реалізації проєкту, вибору технологій та прийняття архітектурних рішень.

1.4.2 Огляд існуючих програмних рішень

На ринку вже існує велика кількість медіаплеєрів з різними рівнями функціональності. Огляд інформаційних джерел дозволить сформувати чітке уявлення про предметну область, сучасні технології та підходи до реалізації мультимедійних систем. Аналіз існуючих рішень виявить як їхні сильні

сторони, так і обмеження, які обґрунтовують необхідність створення власного програмного рішення.

Серед найбільш популярних рішень можна виділити наступні:

VLC Media Player. Відомий кросплатформенний медіаплеєр з відкритим вихідним кодом. Використовує бібліотеку libVLC.

Переваги:

- Відкритий вихідний код.
- Підтримка великої кількості форматів.
- Потужний функціонал з можливістю кастомизації субтитрів, фільтрів, еквалайзеру, тощо.
- Кросплатформеність (Windows, Linux, macOS, Android).
- Активна спільнота розробників.

Недоліки:

- Складна структура коду, важко модифікувати або інтегрувати.
- Обмежені можливості кастомізації GUI для власних потреб.
- Застарілий інтерфейс.

KMPlayer. Пропонує розширені функції, проте є закритим і не дає можливості модифікації або інтеграції з іншими системами.

Переваги:

- Підтримує велику кількість форматів та кодеків.
- Має вбудовані функції для відеофільтрації, корекції кольору, субтитрів.
- Зручний інтерфейс з широкими можливостями налаштування.
- Наявність розширених функцій відтворення (збільшення, уповільнення, скріншоти тощо).

Недоліки:

- Закритий вихідний код — неможливість модифікації або інтеграції в інші системи.
- Обмежена підтримка нестандартних або експериментальних фільтрів.

MPV Player. Потужний, але орієнтований на командну строку та конфігураційні файли, з обмеженим графічним інтерфейсом. Базується на FFmpeg і libmpv.

Переваги:

- Потужне ядро на базі FFmpeg і libmpv.
- Великий функціонал, гарна альтернатива VLC.
- Мінімалістичний інтерфейс.
- Можливість автоматизації через скрипти.

Недоліки:

- Відсутність графічного інтерфейсу для кінцевого користувача за замовчуванням.
- Високий поріг входу для налаштувань (через конфігураційні файли).
- Не призначений для візуальної кастомізації або гнучкого GUI.

Windows Media Player. Простий в користуванні, але має закритий код, обмежену кросплатформеність та застарілий інтерфейс.

Переваги:

- Інтеграція з Windows, не потребує додаткового встановлення.
- Мінімалістичний і зрозумілий інтерфейс.
- Оптимізована робота з медіа в середовищі Windows.

Недоліки:

- Обмежена кількість підтримуваних форматів без сторонніх кодеків.
- Закритий код і неможливість розширення функціоналу.
- Відсутність сучасних функцій для кастомізації, фільтрації відео чи обробки звуку.

- Не оновлюється з появою нових форматів — функціонально застарілий.

Попри широкий вибір готових медіаплеєрів, жоден із них не надає повної свободи кастомізації, гнучкого GUI, інтеграції змінних відео- та аудіофільтрів у режимі реального часу, а також зручної архітектури для розширення. Саме ці фактори обґрунтовують необхідність розробки власного рішення з урахуванням виявлених недоліків і зосередженістю на сучасних технологіях (C++, QML, libav/FFmpeg).

1.5 Постановка завдання

На основі проведеного системного аналізу предметної області та огляду існуючих програмних рішень сформульовано задачу розробки сучасного медіаплеєра, який би забезпечував зручне, стабільне та функціонально насичене відтворення мультимедійного контенту.

Мета проєкту. Розробити медіаплеєр на основі C++, Qt/QML і Libav, який дозволяє користувачу відтворювати відео й аудіо різних форматів, керувати відтворенням, застосовувати відеофільтри, налаштовувати аудіоеквалайзер і масштабувати зображення.

Об'єкт дослідження. Інформаційна система для програвання мультимедійних файлів, яка включає засоби декодування, обробки та відображення відео- й аудіоданих.

Предмет дослідження. Програмні методи, алгоритми та архітектурні рішення для побудови ефективного та гнучкого медіаплеєра, орієнтованого на взаємодію з користувачем і якісне відтворення медіаконтенту.\

Типи користувачів та їхні потреби:

- Звичайні користувачі — потребують інтуїтивно зрозумілого інтерфейсу, базових функцій відтворення (пауза, перемотування, гучність, повноекранний режим), стабільної роботи без збоїв.
- Просунуті користувачі / ентузіасти — очікують додаткових функцій: зміна швидкості, налаштування формату відтворення, доступ до параметрів аудіо/відео.
- Розробники/дослідники — зацікавлені в наявності відкритої архітектури або можливості розширення функціоналу.

Вимоги користувачів:

- Підтримка найпопулярніших відео- та аудіоформатів.
- Можливість регулювання гучності, яскравості, контрасту, швидкості.
- Застосування відеофільтрів та еквалайзера для налаштування якості медіа.
- Простий та адаптивний інтерфейс.
- Надійність і висока продуктивність при відтворенні.

Функціональні можливості системи:

- Відкриття й відтворення локальних медіафайлів.
- Управління відтворенням: пауза, перемотування, зміна гучності, швидкості.
- Візуальні налаштування: зміна розміру вікна, масштабування відео.
- Накладення відеофільтрів у реальному часі.
- Налаштування аудіоеквалайзера.

Очікуваний результат:

- Готовий програмний продукт — медіаплеєр із функціоналом, що відповідає поставленим вимогам.

- Підвищення зручності перегляду мультимедійного контенту користувачами.
- Гнучка архітектура системи, яка дозволяє легко адаптувати програму під нові вимоги.

1.6 Висновки

У першому розділі було здійснено всебічний аналіз предметної області, що стосується розробки медіаплеєра на основі сучасних технологій програмування. В результаті виконано наступне:

У підпункті 1.1 було розкрито сутність предметної області — програмного забезпечення для відтворення мультимедійних файлів. Було окреслено, що така система повинна забезпечувати зручне та якісне програвання аудіо- та відеоконтенту, відповідати вимогам сучасного користувача щодо функціональності, інтерфейсу та продуктивності.

У підпункті 1.2 проведено детальний аналіз вимог до програмної системи, включно з функціональними, нефункціональними та системними вимогами. Це дало змогу чітко окреслити очікування від розроблюваної системи та сформувати основу для її архітектури.

У підпункті 1.3 виконано моделювання предметної області за допомогою UML-діаграм. Зокрема, побудовано діаграми прецедентів для опису взаємодії користувача із системою та діаграму класів для відображення структури системи. Це дозволило візуалізувати функціональність і логіку системи, а також визначити основні компоненти й взаємозв’язки між ними.

У підпункті 1.4 здійснено аналіз існуючих програмних рішень (KMPlayer, VLC, Windows Media Player), визначено їхні переваги та недоліки. На основі цього сформовано бачення того, які функції доцільно включити у власну розробку, а яких — уникати або реалізовувати інакше.

У підпункті 1.5 була сформульована постановка завдання: окреслено мету створення системи, визначено основні функціональні можливості, описано типи

користувачів і сформульовано очікувані результати від реалізації проєкту. Це є ключовим етапом, який визначає подальший напрям дослідження та розробки.

Загалом, перший розділ заклав теоретичне та методологічне підґрунтя для реалізації програмної системи, окреслив вимоги, визначив проблематику, цілі та задачі, а також провів порівняння аналогів, що забезпечує обґрунтованість і актуальність подальших етапів розробки.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі здійснюється проєктування архітектури програмного забезпечення, що є наступним етапом після системного аналізу предметної області. Основною метою даного етапу є формалізація структури майбутньої програмної системи, визначення її логіки, компонентів, взаємозв'язків та інформаційної моделі.

Проєктування охоплює як інформаційне забезпечення (структура і взаємозв'язки даних), так і програмне забезпечення (архітектурна побудова системи, компоненти, класи, пакети, логіка взаємодії між об'єктами).

Проєктна представлення у цьому розділі, є основою для подальшої реалізації програмного забезпечення, забезпечує узгодженість компонентів і дає змогу ефективно керувати розробкою, тестуванням і супроводом системи.

2.1 Логічна модель даних у вигляді ER-діаграми

ER-діаграма — це графічне представлення логічної структури даних у системі. Вона моделює сутності, атрибути та зв'язки між ними. Така модель дозволяє формалізувати уявлення про інформаційні об'єкти предметної області, їхні властивості та взаємодії до моменту реалізації конкретної бази даних.

Зазвичай ER-діаграма застосовується при проєктуванні систем, де важливо чітко структурувати зберігання, обробку та зв'язки великих обсягів даних, наприклад у базах даних підприємств, облікових системах, CRM тощо.

У контексті даного програмного забезпечення, яке є медіаплеєром, відсутня класична база даних у традиційному розумінні. Програма не використовує постійне сховище складної структурованої інформації з великою кількістю сутностей і зв'язків між ними. Вся інформація (наприклад, шляхи до файлів, налаштування користувача тощо) обробляється або в оперативній пам'яті, або зберігається у простих конфігураційних файлах, без застосування реляційної

СУБД. Через це побудова повноцінної ER-діаграми є недоцільною та неінформативною, оскільки в системі:

- відсутні сутності, що зберігаються в базі даних;
- зв’язки між даними є тимчасовими та реалізуються на рівні об'єктної логіки в програмі;
- дані не мають складної багатозв'язної структури, притаманної класичним предметним областям із СУБД.

Детальний опис та візуалізація інформаційної бази системи дивіться у розділі 3.

Замість ER-діаграми доцільніше використати діаграму класів, яка показує логічну структуру об’єктів програми та взаємозв’язки між ними. Вона краще відображає архітектуру об’єктно-орієнтованої системи, де замість таблиць і зв’язків між ними використовуються класи, атрибути, методи та відносини типу «успадкування», «агрегація», «композиція».

2.2 Діаграма класів та кооперацій

2.2.1 Діаграма класів

Діаграма класів є одним з базових інструментів моделювання у нотації UML і використовується для графічного зображення структури об'єктно-орієнтованої програмної системи. Вона дозволяє візуалізувати основні класи, їхні атрибути, методи та логічні зв’язки між класами, такі як наслідування, асоціації, агрегації та композиції [8].

Основне призначення діаграми класів полягає у відображенні:

- структури системи з позиції об’єктів, що її складають;
- взаємозв’язків між елементами програмної системи;
- механізмів повторного використання коду;
- залежностей між класами для подальшої реалізації в коді.

Цей тип діаграми використовується на етапі проектування системи, коли формується її архітектура. Вона допомагає розробникам:

- краще зрозуміти логіку взаємодії компонентів;
- визначити, які класи необхідні;
- проаналізувати, як ці класи будуть взаємодіяти один з одним у процесі виконання програми.

Елементи діаграми:

- Клас — прямокутник, що містить ім’я класу, список атрибутів (змінних) та методів (функцій).
- Атрибути — властивості, що характеризують стан об'єкта класу.
- Методи — функціональність, яку реалізує клас.
- Зв’язки між класами:
  - асоціації,
  - успадкування,
  - агрегації/композиції.

Розглянемо діаграму класів системи перегляду медіафайлів представлену на рис. 3.

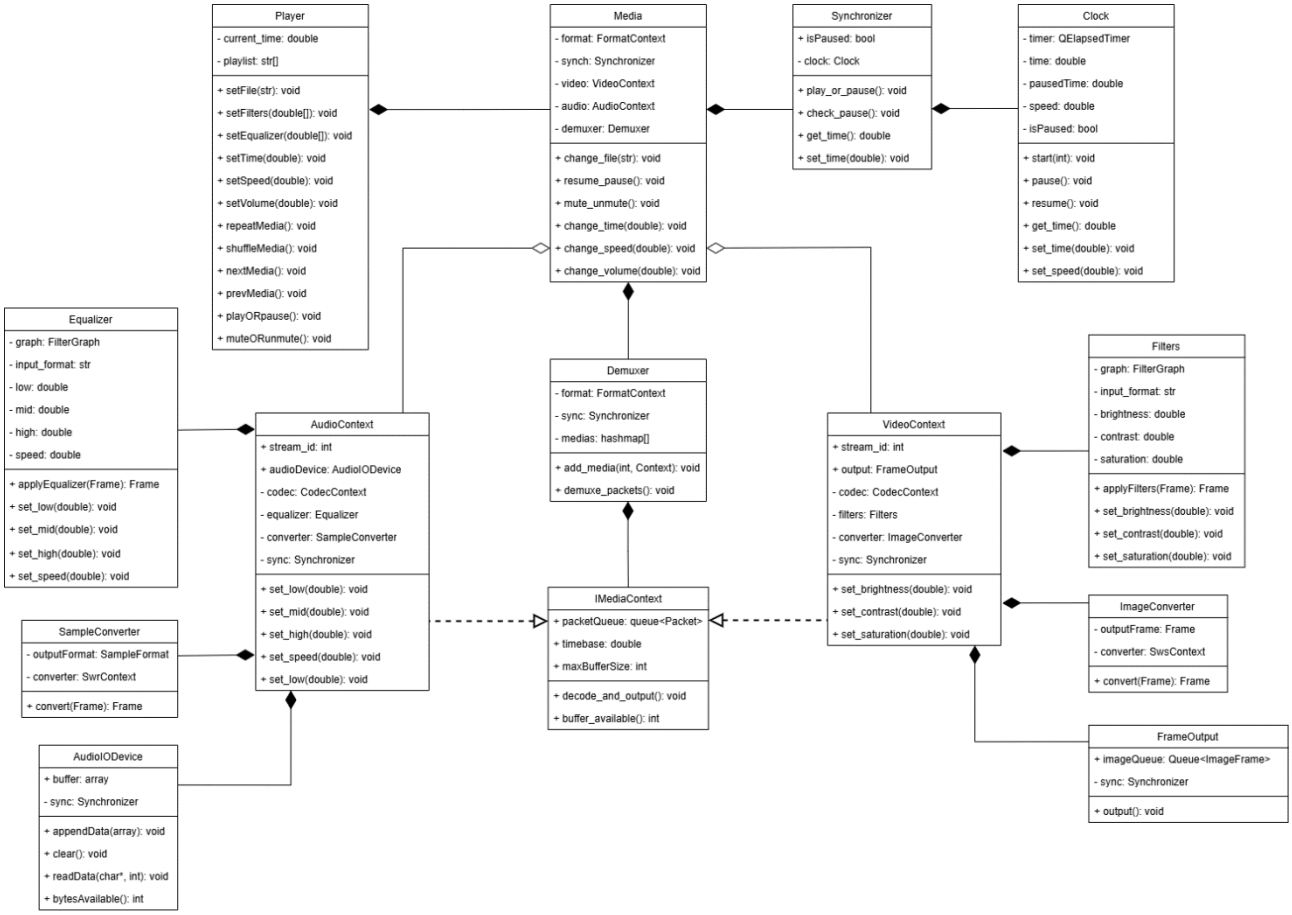


Рис. 3 – Діаграма класів системи перегляду медіафайлів

На діаграмі представлено 14 класів:

- Player – клас взаємодії інтерфейсної та логічної частини програми. Клас приймає сигнали від UI-інтерфейсу та передає їх Media класу. Містить клас Media.
- Media – головний клас логічної частини програми. Керує відтворенням, створенням та налаштуванням відео та аудіо класів. Містить клас-синхронізатор(Synchronizer), клас-демультиплесувальник(Demuxer) та за наявності відео та аудіо класи.
- Synchronizer – клас, відповідальний за синхронізацію відтворення відео та аудіо, містить клас Clock.
- Clock – клас, що зберігає поточний стан та час відтворення.

- Demuxer – клас, відповідальний за розбиття медіафайлу на відео- та аудіо-пакети. Містить перелік медіа-класів успадкованих від IMediaContext.
- IMediaContext – інтерфейс, що містить основу для декодування медіапотоків, а саме чергу закодованих пакетів та метод їх декодування.
- VideoContext – клас, відповідальний за декодування, накладання фільтрів та конвертацію у вихідний формат зображень. Містить інформацію про відео-кодек, клас накладання фільтрів(Filters), конвертор(ImageConvertor) та клас FrameOutput.
- Filters – клас, відповідальний за накладання фільтрів на декодовані зображення. Містить інформацію про параметри налаштування фільтрів.
- ImageConvertor – клас, відповідальний за конвертацію зображень у вихідний формат. Містить інформацію про вихідний відео формат.
- FrameOutput – клас, відповідальний за вивід зображень у UI-інтерфейс. Містить чергу зображень на вивід.
- AudioContext – клас, відповідальний за декодування, накладання аудіофільтрів та конвертацію у вихідний формат аудіо семплів. Містить інформацію про аудіо-кодек, клас еквалайзер(Equalizer), конвертор(SampleConvertor) та клас AudioIODevice.
- Equalizer – клас, відповідальний за налаштування аудіо частот семплів. Містить інформацію про параметри налаштування частот еквалайзеру.
- SampleConvertor – клас, відповідальний за конвертацію аудіо у вихідний формат. Містить інформацію про вихідний аудіо формат.
- AudioIODevice – клас, відповідальний за вивід аудіо у пристрій аудіо виводу. Містить аудіо буфер семплів.

Завдяки діаграмі класів забезпечується зручна візуалізація логіки системи, що полегшує процес розробки, налагодження та підтримки коду. Вона також є основою для розробки інших типів діаграм, таких як діаграми кооперацій, послідовностей, станів тощо.

2.2.2 Діаграма кооперацій

Діаграма кооперацій є різновидом діаграм взаємодії в UML, основна мета яких — показати, як об’єкти взаємодіють між собою під час виконання певного сценарію або прецеденту. Вона фокусується не лише на об'єктах, а й на послідовності обміну повідомленнями між ними, що дозволяє зрозуміти динаміку системи [8].

Основне призначення діаграми кооперацій:

- Відображення взаємодії між об’єктами в рамках конкретного процесу.
- Демонстрація послідовності повідомлень між об'єктами.
- Аналіз поведінки системи з точки зору розподілу відповідальностей.

Цей тип діаграми використовується для:

- моделювання сценаріїв взаємодії (наприклад, запуск функціоналу, обробка дій користувача тощо);
- верифікації правильності зв’язків, які були закладені в діаграмі класів;
- покращення розуміння потоків управління у системі;
- тестування логіки поведінки об’єктів до реалізації коду.

Розглянемо діаграму кооперацій відтворення відео на рис. 4.

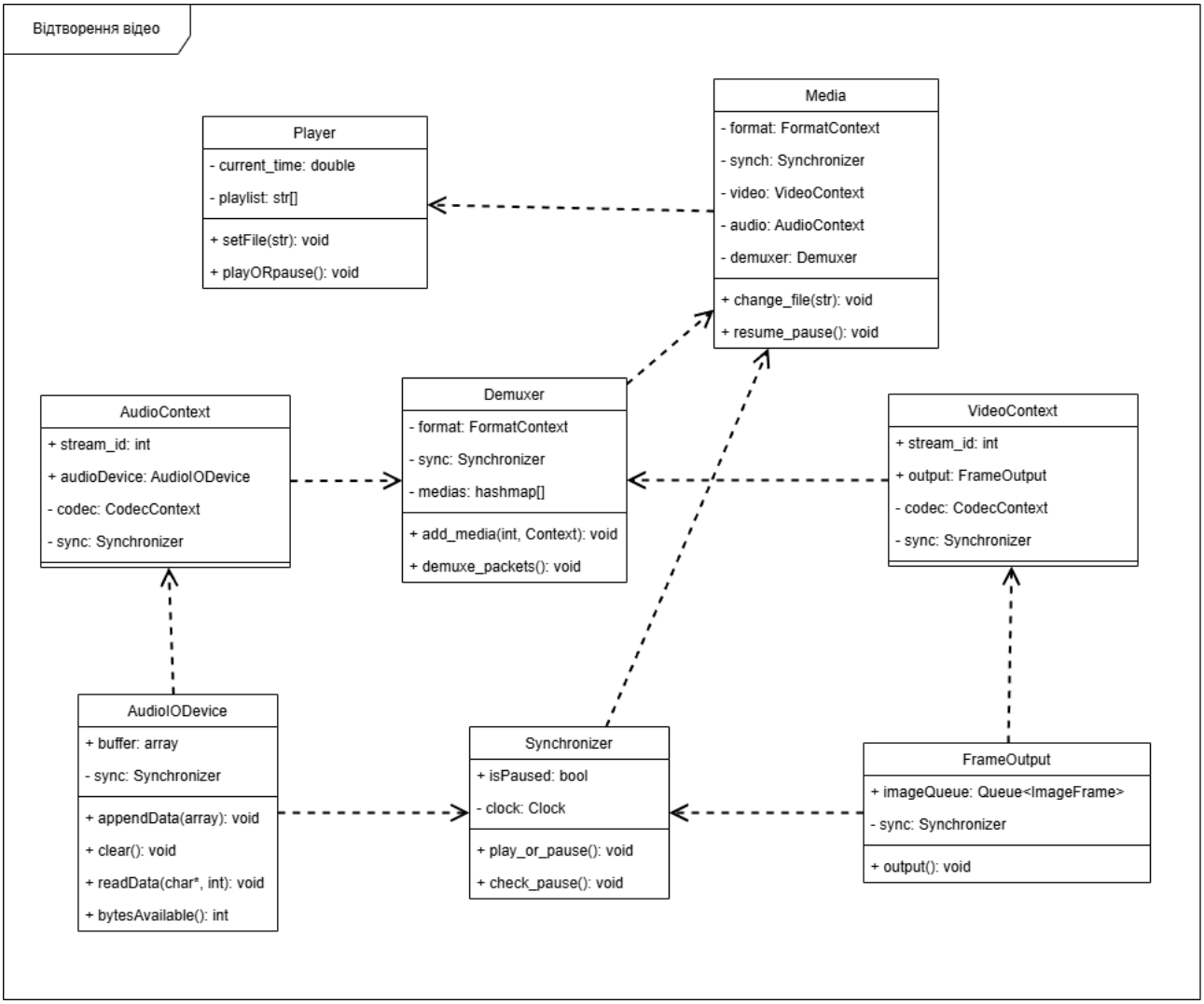


Рис. 4 – Діаграма кооперацій відтворення відео

На діаграму зображено 8 об'єктів:

- Player – керує паузою/відтворенням.
- Media – змінює стан об’єкта Synchronizer, який відповідає за паузу/відтворення вивода медіа. Додає медіакласи до черги Demuxer.
- Synchronizer – зупиняє/відновлює вивід у медіа класах (FrameOutput та AudioIODevice).
- Demuxer – розбиває медіа потоки на відповідні пакети (відео- та аудіо-пакети).

- VideoContext – декодує пакети у зображення та надсилає їх на вивід у чергу FrameOutput.
- FrameOutput – виводить зображення у UI-інтерфейс у відповідний момент часу.
- AudioContext – декодує пакети у аудіо семпли та надсилає їх на вивід у буфер AudioIODevice.
- AudioIODevice – надсилає аудіо семпли у буфер аудіо пристроя.

Розглянемо діаграму кооперацій застосування відеофільтрів та еквалайзеру на рис. 5.

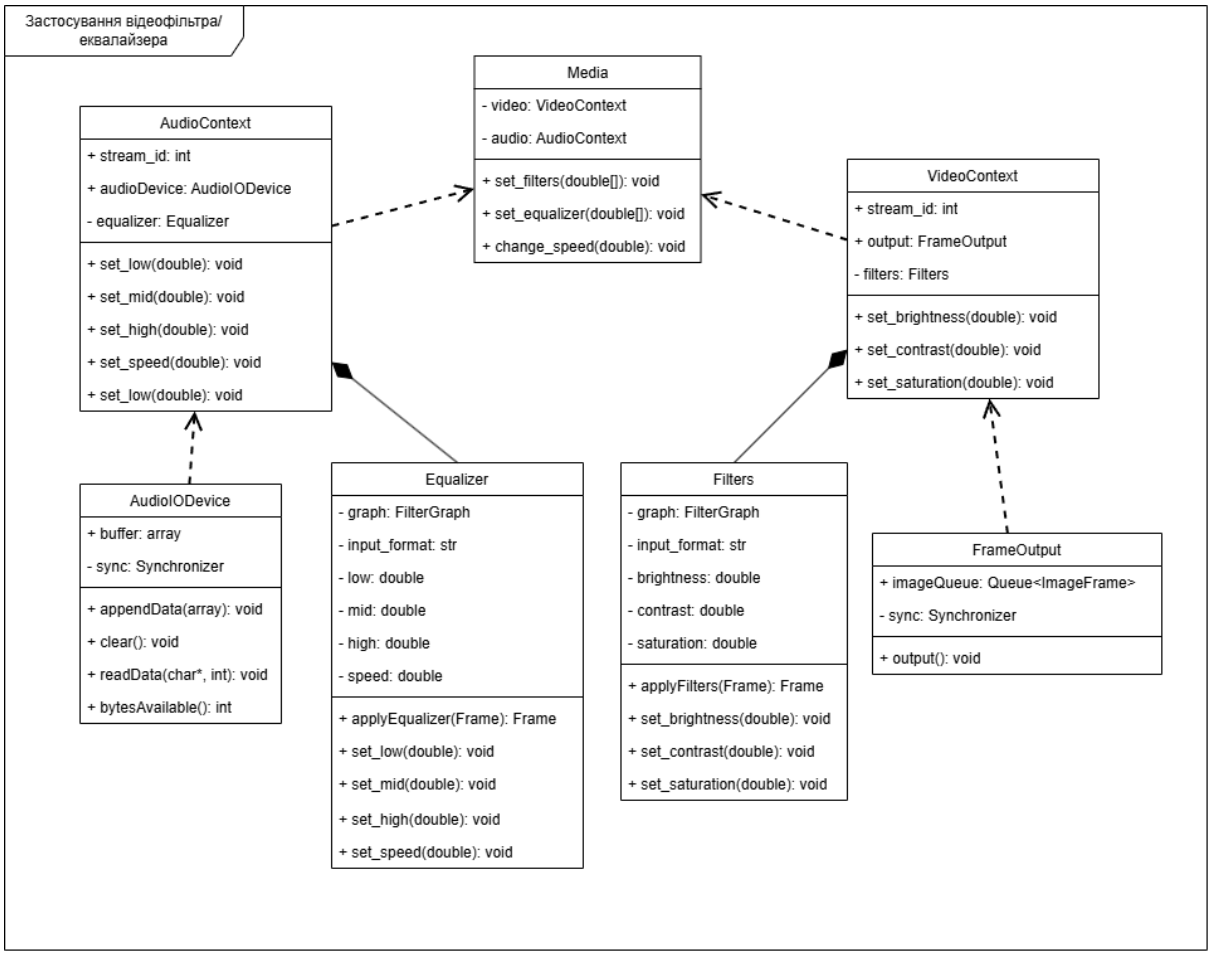


Рис. 5 – Діаграма кооперацій кооперацій застосування відеофільтрів та еквалайзеру

На діаграмі зображено 7 об’єктів:

- Media – надсилає відповідні нові параметри на застосування до VideoContext або AudioContext.
- VideoContext – приймає та оновлює нові фільтри у Filters. Надсилає відфільтроване зображення на вивід у FrameOutput.
- Filters – накладає фільтри на зображення відповідно параметрам.
- FrameOutput – виводить зображення у UI-інтерфейс.
- AudioContext – приймає нові параметри частот та відповідно оновлює Equalizer. Надсилає відфільтровані семпли на вивід у AudioIODevice.
- Equalizer – змінює частоти семплів відповідно параметрам.
- AudioIODevice – виводить аудіо через пристрій аудіо виводу.

Діаграма кооперацій є логічним доповненням до діаграми класів. Якщо діаграма класів показує що існує в системі (структура), то діаграма кооперацій демонструє як саме ці об’єкти взаємодіють під час виконання функціональності.

2.3 Діаграма пакетів

Діаграма пакетів (Package Diagram) є одним з елементів нотації UML і використовується для структурного групування елементів моделі в логічні блоки, які називаються пакетами. Така діаграма дозволяє відобразити архітектуру програмної системи на вищому рівні абстракції, що сприяє кращому розумінню організації проєкту [8].

Основною метою використання діаграми пакетів є:

- логічне групування класів, інтерфейсів та інших елементів системи в окремі пакети (модулі);
- визначення залежностей між цими пакетами;
- структуризація великого проєкту, що підвищує читабельність та керованість кодової бази;
- спрощення рефакторингу та подальшої підтримки системи.

Пакет у UML — це контейнер, який може містити інші пакети, класи, інтерфейси тощо. Діаграма пакетів показує, які частини системи взаємозалежні, та допомагає уникати циклічних або надмірних залежностей між модулями. На практиці кожен пакет може відповідати:

- окремому модулю або бібліотеці;
- підсистемі з власною функціональністю;
- фізичній директорії у файловій структурі проєкту.

Залежності між пакетами позначаються стрілками, які вказують на те, що один пакет використовує (імпортує) інший.

У контексті даної системи (медіаплеєра), діаграма пакетів дозволяє:

- розділити функціональні частини проєкту: інтерфейс користувача, обробка медіа, відтворення відео/аудіо, застосування фільтрів, налаштування, логіка керування тощо;
- визначити чіткі межі відповідальності для кожного компонента;
- забезпечити модульність та полегшити повторне використання частин системи;
- оптимізувати розгортання та обслуговування системи у майбутньому.

Розглянемо діаграму пакетів системи перегляду медіафайлів на рис. 6:

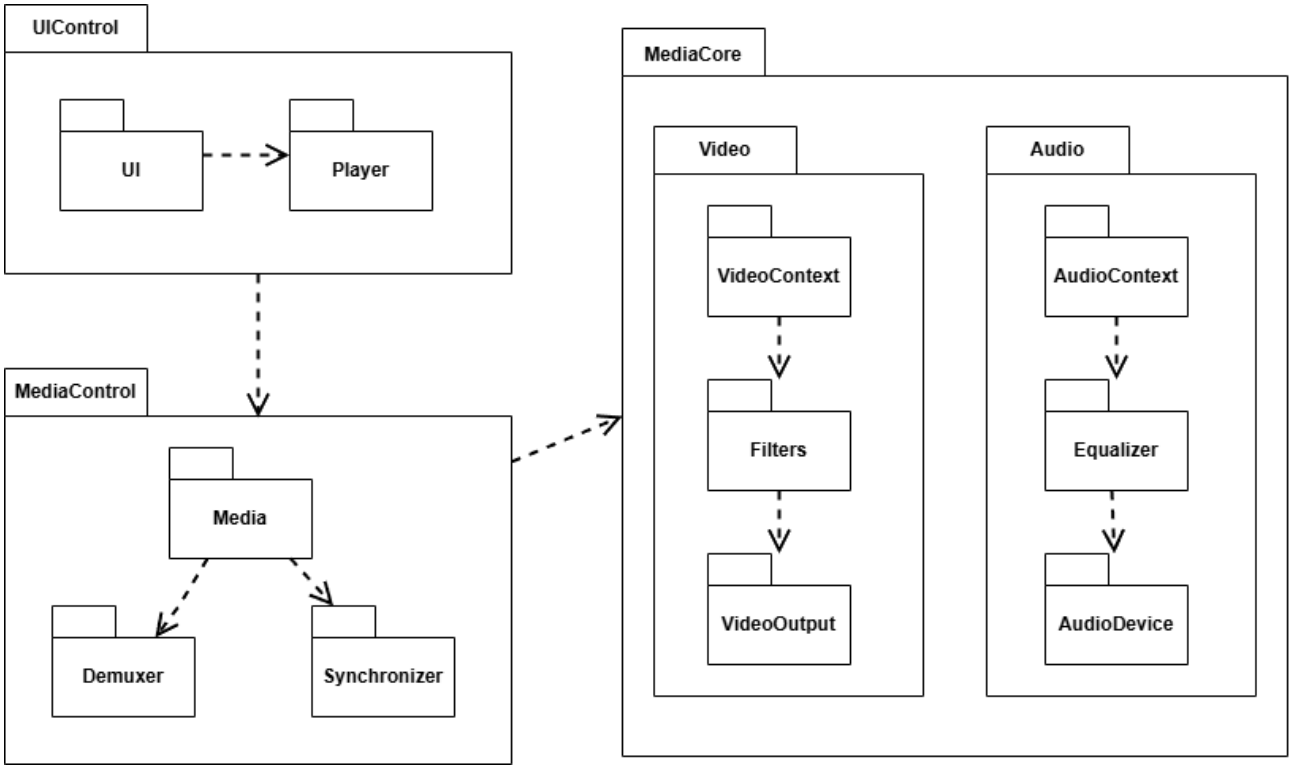


Рис. 6 – Діаграма пакетів системи перегляду медіафайлів

На діаграмі розташовано 3 головних пакета:

- UIControl – пакет складається з двох підпакетів, а саме пакету UI, який надсилає сигнали до пакета Player.
- MediaControl – пакет складається з трьох підпакетів, а саме пакету Media, який керує паузою/відтворенням через пакет Synchronizer, та сигналізує Demuxer про необхідність демультиплексування.
- MediaCore – пакет складається з двох підпакетів Video та Audio, які у свою чергу складаються ще з трьох підпакетів, а саме пакету AudioContext/VideoContext, що надсилає декодований фрейм на фільтрацію у Filters/Equalizer, який у свою чергу надсилай відфільтрований фрейм на вивід у VideoOutput/AudioDevice.

Діаграма пакетів — це зручний інструмент для побудови високорівневої архітектури програмного забезпечення. Вона дозволяє створити організовану, зрозумілу і масштабовану структуру, яка зменшує складність розробки та спрощує командну роботу над проєктом.

2.4 Діаграма компонентів

Діаграма компонентів (Component Diagram) — це структурна діаграма в нотації UML, яка використовується для відображення фізичних частин програмної системи, зокрема її компонентів (модулів, бібліотек, файлів) та взаємозв'язків між ними [8].

Призначення діаграми компонентів:

- моделювати фізичну архітектуру системи;
- відображати компоненти програмного забезпечення, які можуть бути повторно використані або замінені;
- ілюструвати взаємозалежність між модулями;
- підтримувати управління версіями та залежностями;
- покращити розгортання та інтеграцію системи.

Компонентами можуть бути:

- виконувані файли (.exe, .dll),
- динамічно підключувані бібліотеки,
- окремі модулі або плагіни,
- частини коду, реалізовані як окремі проєкти.

На діаграмі компонентів відображаються:

- самі компоненти (у вигляді прямокутників зі спеціальним позначенням);
- інтерфейси, які вони реалізують або використовують;

- залежності між компонентами — хто кого викликає, використовує або реалізує.

Це дозволяє визначити:

- які частини системи є взаємозалежними;
- які компоненти можуть бути відокремлені для тестування або заміни;
- як здійснюється взаємодія між логічно відокремленими частинами проєкту.

Розглянемо діаграму компонентів системи перегляду медіафайлів на рис. 7:

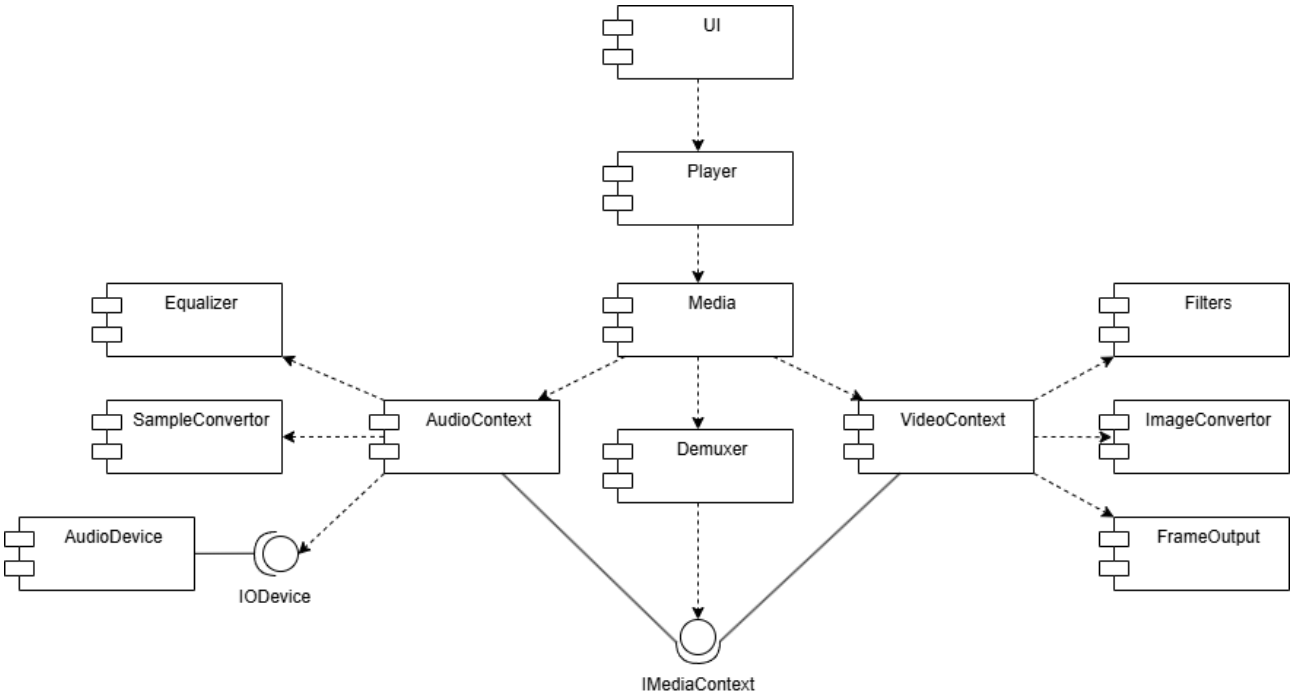


Рис. 7 – Діаграма компонентів системи перегляду медіафайлів

На діаграмі зображено 12 компонентів та 2 інтерфейсі.

- UI надсилає сигнали до компоненту Player.
- Player передає сигнали до компоненту Media.
- Media створює відео та аудіо класи (AudioContext та VideoContext), та додає їх у компонент демультиплексування Demuxer.

- Demuxer зберігає перелік медіа як наслідників інтерфейсу IMediaContext та надсилає їм демультиплексовані пакети.
- VideoContext отримує пакети, декодує їх, накладає фільтри на декодовані зображення через Filters, конвертує зображення у вихідний формат через ImageConvertor та надсилає готові зображення на вивід у компонент FrameOutput.
- AudioContext отримує пакети, декодує їх, накладає фільтри на декодовані семпли через Equalizer, конвертує семпли у вихідний формат через SampleConvertor та надсилає готові семпли на вивід у компонент AudioDevice через інтерфейс IODevice.

Діаграма компонентів відіграє ключову роль у структуризації складних програмних систем, дозволяючи візуалізувати, як реалізовано компоненти, як вони взаємодіють і які залежності існують між ними. У випадку розробки медіаплеєра ця діаграма допомагає забезпечити надійність, модульність та гнучкість архітектури, що є важливим для подальшої підтримки й розвитку ПЗ.

2.5 Висновки

У другому розділі було здійснено проєктування інформаційного та програмного забезпечення, яке є основою для реалізації функціональної, надійної та масштабованої системи — медіаплеєра.

Розпочато з логічного моделювання даних, де розглянуто принципи побудови ER-діаграм як інструменту для представлення сутностей і зв’язків у системі. З огляду на специфіку застосунку, було обґрунтовано відмову від класичної ER-діаграми, у зв’язку з формою збереження даних у наявній системі.

Детально розглянуто діаграму класів, яка дозволила структурувати об’єкти предметної області, визначити їх атрибути, методи та взаємозв’язки. Діаграма

кооперацій доповнила це представлення, показавши взаємодії між об'єктами в межах конкретних сценаріїв використання.

Діаграма пакетів дозволила логічно згрупувати компоненти системи у функціональні блоки, покращивши розуміння загальної архітектури програмного продукту. Вона стала основою для підтримки принципів інкапсуляції, низької зв’язаності та високої модульності.

Діаграма компонентів надала уявлення про фізичну структуру проєкту, описала модулі, бібліотеки та зв’язки між ними. Це критично важливо для ефективної реалізації, підтримки та масштабування програмного забезпечення, а також для забезпечення незалежного розвитку окремих частин системи.

Усі побудовані діаграми в сукупності створили цілісне уявлення про структуру, функціональність і взаємодію елементів системи. Проєктування, здійснене на цьому етапі, забезпечує чітке бачення архітектури застосунку та слугує міцним фундаментом для його реалізації.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Цей розділ присвячено безпосередньо розробці інформаційної та програмної складових системи — медіаплеєра. У ньому детально розглядаються питання побудови інформаційної бази, вибору інструментальних засобів та створення програмної логіки, яка реалізує поставлені функціональні вимоги.

Спочатку аналізується система управління інформаційною базою, яка забезпечує зберігання та обробку необхідних даних. Далі розглядається процес проєктування та реалізації самої інформаційної бази.

Окрему увагу приділено обґрунтованому вибору інструментарію, необхідного для реалізації прикладного програмного забезпечення. Інструменти було підібрано з урахуванням сумісності, продуктивності, зручності використання, а також відповідності технічним і функціональним вимогам системи.

Завершується розділ описом процесу алгоритмізації та програмування основних модулів системи. На цьому етапі формалізовано логіку роботи застосунку, реалізовано ключові функції та забезпечено цілісну взаємодію між компонентами, створюючи основу для практичного використання розробленого програмного забезпечення.

3.1 Система управління інформаційною базою

Система управління інформаційною базою (СУІБ) — це програмне забезпечення або вбудований модуль, що відповідає за створення, збереження, обробку та доступ до структурованих даних, необхідних для функціонування програмної системи. Зазвичай СУІБ реалізується у вигляді повноцінної системи керування базами даних (СУБД), яка забезпечує централізоване зберігання та обробку великих обсягів даних.

Однак характер розроблюваної системи — медіаплеєра, який є настільним прикладним програмним забезпеченням — не потребує реалізації окремої інформаційної бази чи повноцінної СУБД. Це зумовлено наступними факторами:

- Відсутність складної структури даних. Система не працює з об’ємними наборами інформації або багатозв’язними таблицями, які потребують реляційного подання.
- Орієнтація на локальне використання. Всі дії користувача відбуваються на локальному рівні, без обміну даними через мережу чи необхідності централізованого зберігання.
- Тимчасовий або неструктурований характер даних. Список відтворення, положення у файлі, параметри відображення чи звуку — це дані, які не вимагають складної моделі збереження.

У даній системі замість повноцінної інформаційної бази використовується файлова модель збереження. Основні параметри та конфігураційні дані (наприклад, останній відкритий файл, позиція відтворення, рівень гучності, вибрані фільтри тощо) зберігаються у вигляді локального конфігураційного файлу. Такий підхід є цілком достатнім для функціонування медіаплеєра, а також дозволяє уникнути зайвих ускладнень, пов’язаних з проєктуванням та підтримкою СУБД.

Таким чином, у розробленій системі відсутня окрема інформаційна база в класичному розумінні, оскільки:

- Вся необхідна інформація або зчитується безпосередньо з файлів користувача, або формується під час сесії;
- Тимчасові дані не потребують довготривалого зберігання;
- Застосування повноцінної СУБД суперечило б принципу легкості та простоти застосунку.

Такий підхід дозволяє підтримувати гнучкість системи, зменшити обсяг ресурсів, спростити інсталяцію, а також забезпечити швидкий доступ до конфігураційних параметрів без необхідності звертання до зовнішніх сервісів або СУБД.

3.2 Представлення інформаційної бази

У контексті медіаплеєра поняття інформаційної бази набуває дещо іншого значення. Замість класичної структурованої бази даних система оперує медіафайлами — основним джерелом вхідної інформації, яка підлягає обробці, декодуванню та виведенню.

3.2.1 Формат збереження

Медіафайл — це цифровий контейнер, який зберігає одне або кілька потоків мультимедійних даних. Медіафайли зберігаються безпосередньо на файловій системі операційної системи користувача. Вся структура медіафайлів організована через файлові каталоги, що дозволяє зберігати файли різних форматів (MP4, AVI, MKV, MP3 та ін.) у відповідних папках. Це спрощує доступ до файлів, оскільки не потрібно додатково налаштовувати або підтримувати складні бази даних. Формат медіафайлу визначає структуру контейнера, спосіб індексації потоків, підтримку додаткових треків, тощо.

Зазвичай медіафайл складається з:

- Відеопотоку (наприклад, закодованого за допомогою H.264, VP9, AV1).
- Аудіопотоку (AAC, MP3, Opus).
- Субтитрів (у форматах SRT, ASS, внутрішніх чи зовнішніх).
- Метаданих (назва, автор, тривалість).

Контейнер — це спеціальний формат файлу, який об’єднує кілька потоків медіаданих (відео, аудіо, субтитри) в один файл і забезпечує:

- синхронізацію між відео та аудіо (наприклад, щоб звук не відставав від картинки),
- гнучкість — можливість містити декілька треків (кілька аудіодоріжок, субтитрів),
- метадані — інформацію про кодеки, тривалість, розділи, обкладинки,
- швидкий доступ — через індексацію кадрів підтримує перемотування та пошук по таймінгу.

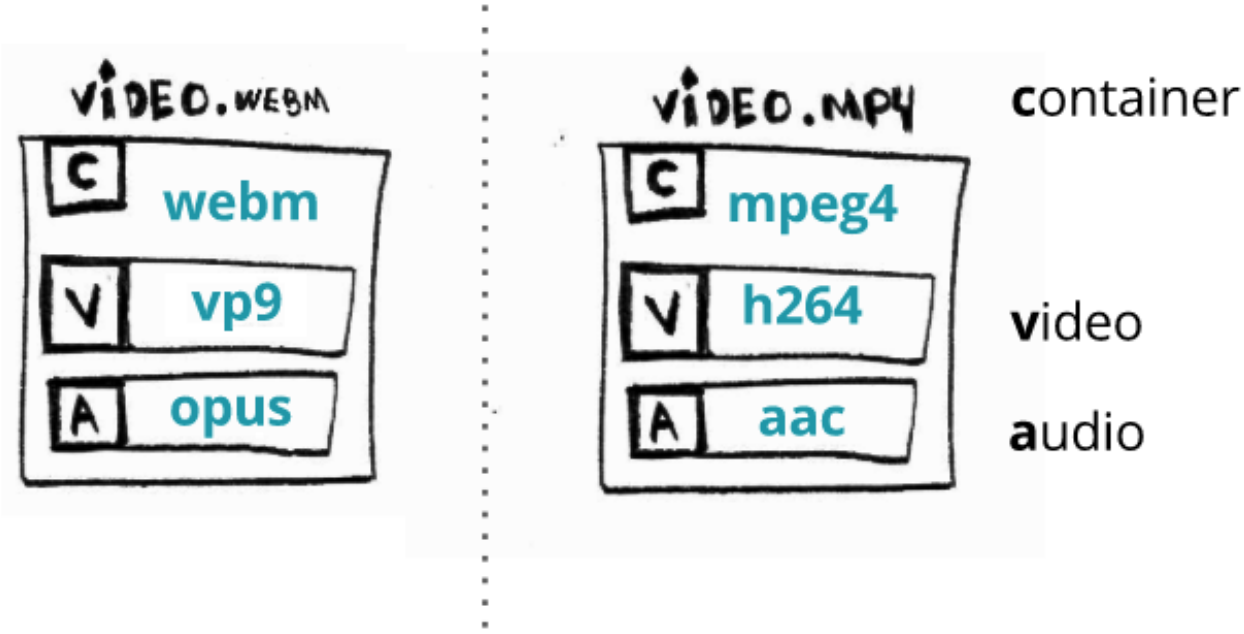


Рис. 8 – Приклад структури медіафайлів

Кожен контейнер (MP4, MKV, AVI) має власну структуру зберігання, але в загальному випадку складається з заголовка, таблиці потоків і послідовності мультимедійних пакетів [2]. Приклад послідовності розташування даних у медіафайлів:

1. Заголовок, тип формату (MP4).
2. Потік 1, тип потоку – відео, кодек – H.264, метадані.
3. Потік 2, тип потоку – аудіо, кодек – AAC, метадані.

4. Сирі дані у вигляді чергування стисненних пакетів різних потоків.

Без контейнера кожен потік зберігався б окремо — відео в одному файлі, аудіо в іншому, субтитри в третьому — і плеєр повинен був би вручну синхронізувати їх під час відтворення [9].

Приклади різних контейнерів:

- MP4. Дуже популярний, підтримує відео, аудіо, субтитри, метадані. Добре працює у вебі та на мобільних пристроях.
- MKV. Гнучкий відкритий формат, підтримує будь-які кодеки, кілька треків відео/аудіо/субтитрів, глави, обкладинки.
- AVI. Старіший формат, менш гнучкий, часто має проблеми з новими кодеками та субтитрами.
- MOV. Пропрієтарний формат Apple, схожий на MP4.
- WEBM. Веб-орієнтований відкритий формат, оптимізований для стрімінгу (використовує VP9/AV1 + Opus).

3.2.2 Кодеки та стиснення

Кодек — це програма або алгоритм, який:

- Кодує (стискає) відео або аудіо під час створення/збереження медіафайлу.
- Декодує (розпаковує) стиснуті дані під час відтворення.

Кодек не є форматом файлу, а є методом стиснення інформації, яку потім обгортає контейнер.

Приклади відеокодеків:

- H.264. Найпоширеніший. Висока якість при малому розмірі.

- H.265. Ще краща компресія, особливо для 4K.
- VP9. Відкритий аналог H.265, використовується в YouTube.
- AV1. Новий відкритий стандарт, ще ефективніший, але ресурсозатратний.

Приклади аудіокодеків:

- MP3. Найпопулярніший аудіокодек зі стисканням з втратами.
- AAC. Краща якість ніж MP3 за того ж бітрейту.
- FLAC. Стискання без втрат.
- Opus. Для VoIP та стрімінгу, адаптивний бітрейт.

Стиснення — це процес зменшення обсягу даних без або з частковою втратою інформації, щоб:

- зменшити розмір файлів;
- пришвидшити передачу через мережу;
- знизити навантаження на пам’ять і диск.

Для розуміння чому потрібне стиснення навіть статичних на локальному диску медіафайлів, проведемо порівняння:

- Зображення без стиснення:
  - Формат: Розмір екрану у пікселях 1920×1080, 24 біти (3 байти на піксель RGB).
  - Розмір: 1920 × 1080 × 3 байти = 6 220 800 байтів ≈ 6 МБ (на 1 кадр!)
  - 6 МБ × 30 = 180 МБ. => 1 хвилина ≈ 10,8 ГБ!
- Зображення зі стисненням (JPEG або H.264 кадр)
  - Розмір того ж кадру у форматі JPEG: ~300 КБ (із втратами)
  - Отже, 1 секунда відео (30 кадрів) = ~300 КБ × 30 = 9 МБ
  - 1 хвилина ≈ 540 МБ

Є два типи стиснення:

- Стискання без втрат (FLAC, HuffYUV, Apple ProRes):
  - Зберігає усі бітові дані.
  - При розпаковці дані ідентичні оригіналу.
  - Застосовується для зберігання майстр-версій (архіви, професійне аудіо/відео).
- Стискання з втратами (H.264, MP3, AAC):
  - Видаляє інформацію, малопомітну для людини, щоб зменшити розмір.
  - Дані не можна відновити в оригінальному вигляді.
  - Найчастіше застосовується в потоковому відео, фільмах, YouTube тощо.

Усі медіафайли — це не прості потоки даних, а результат складного багатоступеневого стиснення за допомогою кодеків, об’єднаного у зручний контейнер. Така архітектура:

- забезпечує сумісність між пристроями;
- дозволяє швидко завантажувати і переглядати відео;
- оптимізує обсяг сховища;
- дає можливість ефективно декодувати відео на льоту під час перегляду.

У контексті розробленого медіаплеєра — кодеки і стиснення є центральним елементом, який визначає якість, продуктивність та гнучкість у роботі з мультимедіа.

3.2.3 Структура та послідовність розпакування медіафайлів

Медіафайл — це контейнер потоків стисненних даних. Тому для відтворення медіафайлу необхідно здійснити розпакування (декодування) цих потоків у послідовності, що забезпечує коректне відображення інформації.

Основні етапи розпакування:

- Ініціалізація контейнера. Файл відкривається та аналізується його структура, визначаються всі доступні потоки (відео, аудіо тощо). Контейнер надає інформацію про формат і розташування даних.
- Читання пакетів з контейнера. Дані у файлі зберігаються у вигляді пакетів — блоків стиснених даних, які належать конкретному потоку. Пакети послідовно витягуються із контейнера.
- Визначення типу потоку та передачі пакета до декодера. Кожен пакет направляється у відповідний декодер (відео- чи аудіо-кодек) залежно від типу потоку.
- Декодування пакетів у кадри або семпли. Кодек розпаковує стиснені дані пакета, перетворюючи їх у формат, придатний для відтворення — відеокадри або аудіосемпли.
- Синхронізація та обробка часу відтворення. Для коректного відтворення кадри або семпли маркуються часовими позначками (PTS, DTS), щоб забезпечити правильний порядок та синхронність аудіо- та відеопотоків.
- Відтворення розпакованих даних. Розкодовані кадри та аудіо відправляються на відповідні пристрої відтворення — відеоекран, аудіопристрій.

Невідповідність або порушення порядку пакетів призводить до збоїв, затримок або артефактів у відтворенні. Синхронізація аудіо і відео забезпечує

комфортний перегляд без розсинхронізації. Ефективне розпакування мінімізує затримки та навантаження на систему.

Таким чином, процес розпакування — це складний, багатоступеневий механізм, який трансформує стиснені потоки з контейнера у відтворюваний контент, забезпечуючи якість і стабільність роботи системи. У даній системі перегляду медіафайлів процес розпакування забезпечується за допомогою бібліотеки FFmpeg та її C++ API — libav.

Таким чином, представлення інформаційної бази у медіаплеєрі полягає в ефективній обробці вхідних мультимедійних потоків, без застосування класичної БД, але з реалізацією всіх етапів роботи з файлами — від відкриття до відтворення у візуальній формі.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Процес розробки прикладного програмного забезпечення вимагає ретельного підбору інструментарію, який дозволяє ефективно реалізувати функціональність, відповідати вимогам до продуктивності, стабільності та гнучкості системи. Нижче наведено опис обраних інструментів, бібліотек та мов програмування, які використовуються для реалізації програмного забезпечення — медіаплеєра.

Мова програмування C++ — це високопродуктивна мова програмування загального призначення, яка поєднує низькорівневий контроль за ресурсами з підтримкою об'єктно-орієнтованого програмування.

C++ використовується для реалізації ядра медіаплеєра: логіки обробки відео та аудіо, взаємодії з бібліотеками для декодування (Libav), керування пам’яттю, потоками, буферами та іншими низькорівневими механізмами.

Причини вибору C++:

- Надзвичайно швидка обробка даних — критично важливо при роботі з відео в реальному часі.
- Дає повний контроль над пам’яттю та ресурсами.
- Підтримується всіма ключовими мультимедійними бібліотеками (наприклад, Libav/FFmpeg, Qt).
- Підходить для кросплатформенного ПЗ.

Фреймворк Qt — це кросплатформенний фреймворк для створення графічних інтерфейсів та розробки прикладного ПЗ з підтримкою GUI, OpenGL, потоків, файлових систем, мультимедіа та мереж.

Функції у проєкті:

- Організовує роботу інтерфейсу користувача.
- Інтегрується з QML для розділення логіки (C++) та UI.
- Використовується для обробки подій, створення вікон, елементів керування, взаємодії з OpenGL.

Причини вибору:

- Підтримує сучасні технології UI.
- Вбудована підтримка мультимедіа, OpenGL.
- Потужна екосистема та документація.
- Підтримка платформ: Windows, Linux, macOS, Android.

QML (Qt Modeling Language) — декларативна мова опису інтерфейсу користувача, розроблена для Qt. Схожа на JSON або CSS/JavaScript і дозволяє швидко створювати UI з реактивною поведінкою.

Функції у проєкті:

- Визначає зовнішній вигляд інтерфейсу медіаплеєра (кнопки, слайдери, меню).

- Підтримує анімації, стилі та адаптивний дизайн.
- Спрощує розділення логіки (C++) та UI.

Причини вибору:

- Дуже швидка розробка UI.
- Ідеально інтегрується з C++.
- Дозволяє будувати UI, що гнучко реагує на зміну станів (наприклад, під час перемотування або зміни фільтра).
- Менше коду, швидше розроблення, краще розділення обов'язків.

Libav — це набір бібліотек для обробки відео та аудіо. Є частиною або аналогом FFmpeg. Серед основних модулів:

- libavformat — робота з контейнерами (MP4, MKV, AVI),
- libavcodec — декодування та кодування (H.264, AAC тощо),
- libavutil — утиліти, які підтримують решту модулів,
- libswscale — масштабування та перетворення відео,
- libswresample — обробка аудіо (частота, канали тощо).

Функіонал у проєкті:

- Відкриває та зчитує медіафайли.
- Розпізнає потоки (аудіо, відео, субтитри).
- Декодує кадри відео та аудіо.
- Масштабує, перетворює формати, передає на рендеринг.

Причини вибору:

- Підтримує практично всі формати медіа.
- Надійність та стабільність у промислових застосуваннях.
- Висока продуктивність при роботі з HD/4K відео.
- Повна гнучкість і контроль над усім медіапотоком.

Cmake — це система автоматизованої побудови проєктів, яка дозволяє описувати структуру програмного коду у спеціальному конфігураційному файлі (CMakeLists.txt) і на основі цього генерує скрипти для складання (Makefile, Visual Studio Project тощо). Вона особливо корисна для великих або кросплатформених проєктів, де є багато залежностей або бібліотек, адже автоматизує процес компіляції, зв’язування та налаштування середовища розробки.

Функції у проєкті:

- Автоматично конфігурує проєкт залежно від ОС та середовища.
- Підключає зовнішні бібліотеки (Qt, Libav).
- Збирає весь код у виконуваний файл.
- Дозволяє масштабувати та підтримувати проєкт.

Причини вибору:

- Основний стандарт у проєктах з використанням C++ і Qt.
- Дозволяє створювати кросплатформенне ПЗ.
- Гнучко керує залежностями.

Qt Creator — офіційне IDE від Qt. Орієнтоване на створення проєктів із використанням C++, QML і CMake.

Функції у проєкті:

- Служить основним середовищем розробки.
- Надає зручне автодоповнення, інтеграцію з QML, UI-редактор.
- Дозволяє відлагоджувати програму в реальному часі.

Причини вибору:

- Ідеальна інтеграція з Qt та CMake.
- Можливість одночасної роботи з C++ і QML.

- Зручно для швидкого прототипування та рефакторингу.

Git — це система керування версіями, яка зберігає історію змін у проєкті та дозволяє ефективно керувати кодом, працювати з різними версіями, створювати гілки для нових функцій та об’єднувати зміни.

Функції у проєкті:

- Зберігає історію змін.
- Дозволяє відкатитися до попередніх версій.
- Працювати в гілках (наприклад, окремо розробляти аудіофільтр).

Причини вибору:

- Стандарт де-факто в індустрії.
- Сумісний з GitHub/GitLab для зберігання та співпраці.
- Дозволяє ефективно управляти великим проєктом.

Вибраний набір інструментів є оптимальним для створення кросплатформенного медіаплеєра з гнучким графічним інтерфейсом, високою продуктивністю при обробці медіа та можливістю масштабування. Використання C++ та Libav забезпечує продуктивність і контроль, Qt та QML — сучасний інтерфейс, а CMake та Git — надійність у розробці.

3.4 Алгоритмізація та програмування програмних модулів

На цьому етапі відбувається безпосередня реалізація програмної системи згідно з попередньо розробленими моделями та архітектурними рішеннями. Програмні модулі створюються як окремі логічні компоненти, кожен із яких реалізує конкретний набір функцій. Особливу увагу приділено алгоритмізації — формальному опису логіки роботи системи, що дозволяє ефективно реалізувати функціональність, забезпечити розширюваність і підтримуваність коду. У цій

частині описується, які саме модулі були реалізовані, як вони взаємодіють між собою, і які алгоритми покладено в основу їхньої роботи.

Загальна структура коду програми:

1. QML файли інтерфейсу програми: Main.qml, та інші складові елементи — Menubar.qml, ToolsMenu.qml, EqualizerWindow.qml, FiltersWindow.qml.
2. Файли, що керують взаємодією інтерфейсної та логічної частини програми: Player.cpp та Media.cpp.
3. Файли розпакування, розподілення та декодування медіа: Media.cpp, Demuxer.cpp, VideoContext.cpp та AudioContext.cpp.
4. Файли обробки декодованих даних: VideoContext.cpp, Filters.cpp, ImageConvertor.cpp, AudioContext.cpp та Equalizer.cpp.
5. Файли синхронізації та виводу медіа у відповідні пристрої: Synchronizer.cpp, FrameOutput.cpp, AudioIODevice.cpp.

Весь інтерфейс

3.4.1 Розробка інтерфейсу програми

У процесі розробки було створено графічний інтерфейс користувача за допомогою мови опису інтерфейсів QML. Вибір QML зумовлений її зручністю для створення сучасного, адаптивного та візуально привабливого інтерфейсу, що легко інтегрується з логікою на C++ через фреймворк Qt.

Попередньо інтерфейс було змодельовано у Figma, що дало змогу опрацювати структуру, компонування елементів та зовнішній вигляд ще до реалізації, а також забезпечити зручність взаємодії користувача з програмою.

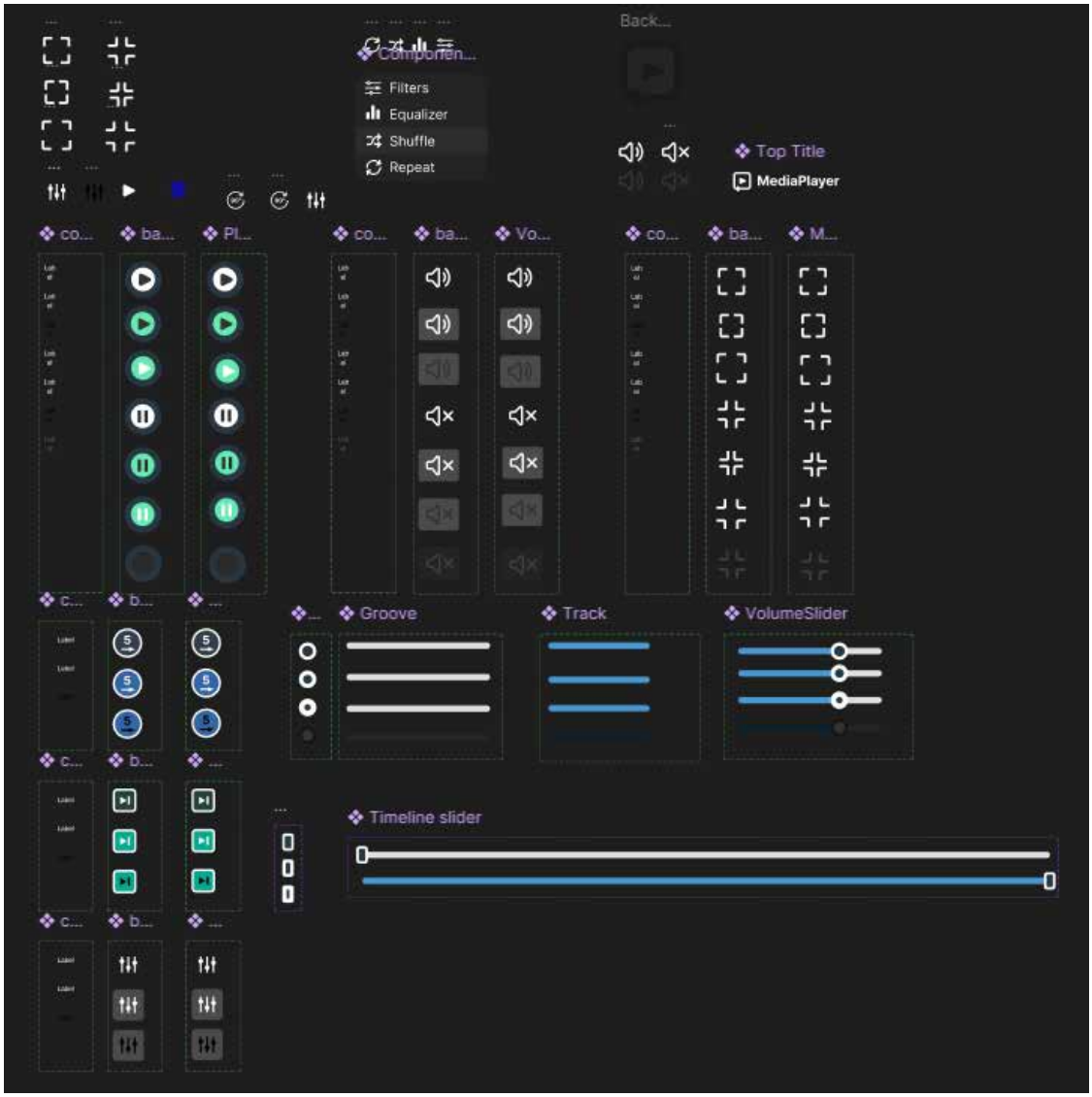


Рис. 9 – Попереднє моделювання елементів інтерфейсу у Figma

Більш детальний опис інтерфейсу дивіться у Додатку А.

3.4.2 Система сигналів і слотів Qt

Однією з ключових особливостей фреймворку Qt є система сигналів і слотів, яка відіграє критичну роль у реалізації моєї системи. Це потужна система комунікації між об'єктами, яка дозволяє реалізовувати реактивну поведінку інтерфейсу користувача та внутрішніх модулів програми.

Сигнал — це повідомлення, яке об'єкт може надіслати, коли в ньому відбувається певна подія (наприклад, натискання кнопки, завершення

відтворення відео, зміна гучності тощо). Слот — це функція, яка реагує на відповідний сигнал. Коли сигнал з'єднаний зі слотом, виклик сигналу автоматично призводить до виклику пов’язаного слота.

Переваги сигналів і слотів:

- Слабке зв'язування: об'єкти не знають нічого один про одного, що покращує модульність та розширюваність програми.
- Асинхронна передача повідомлень: дозволяє реалізовувати складні сценарії взаємодії між компонентами, не блокуючи основний потік виконання.
- Читабельність і зручність: код стає логічно зрозумілим і зручним для супроводу.

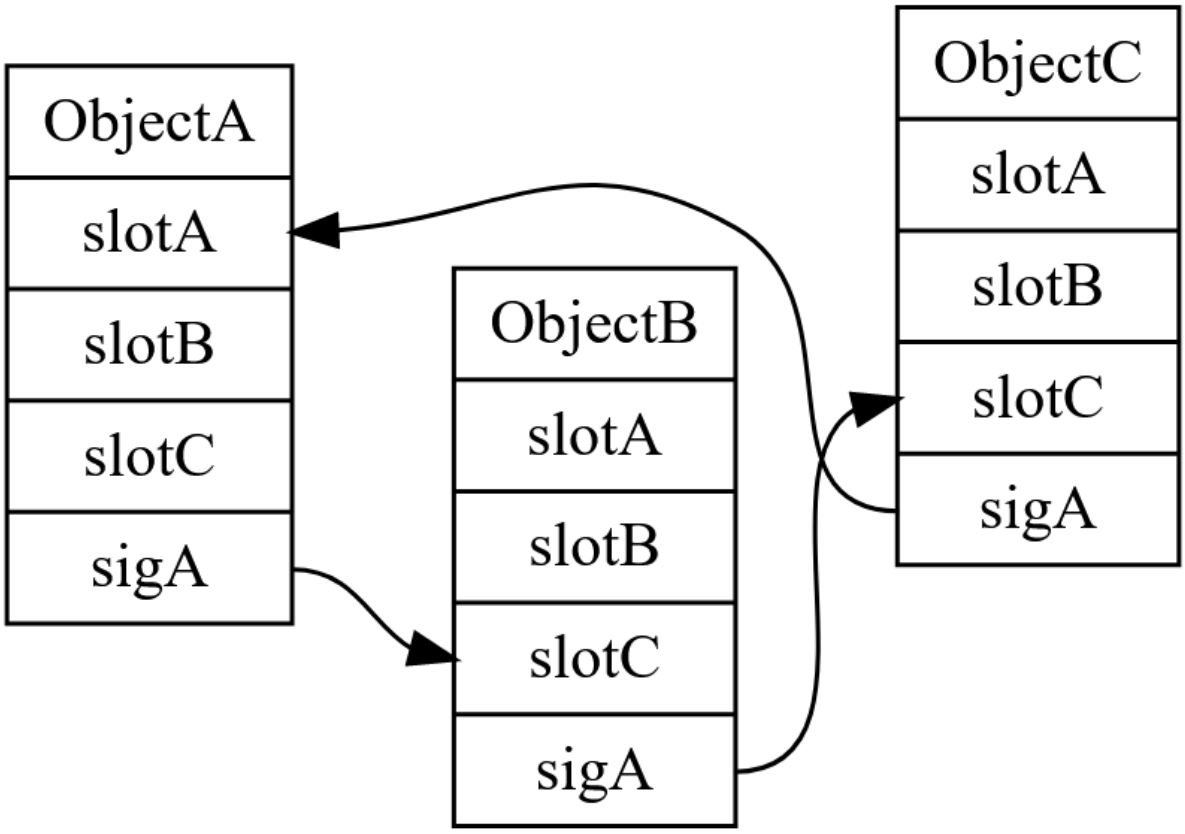


Рис. 10 – Взаємодія сигналів та слотів у об’єктах-наслідниках QObject

Система сигналів і слотів реалізується через метаоб’єктну систему Qt (Meta-Object System) та механізм механізму наслідуваних класів QObject, які забезпечують динамічне викликання методів та обробку подій. Після компіляції спеціальний препроцесор Qt (moc) генерує додатковий код для обробки сигналів і слотів [7].

Система сигналів і слотів у системі перегляду медіафайлів допомогає реалізувати наступні алгоритми:

- Взаємодія інтерфейсної і логічної частин програми.
- Механізм сповіщення о необхідності та результату демультиплексування.
- Механізм сповіщення о надходженні та виводу декодованих зображень/семплів.

3.4.3 Алгоритм демультиплексування

Демультиплексування (demultiplexing) — це процес розділення медіафайлу, який містить кілька типів даних (наприклад, відео, аудіо, субтитри), на окремі потоки (streams), кожен з яких відповідає за свій тип даних.

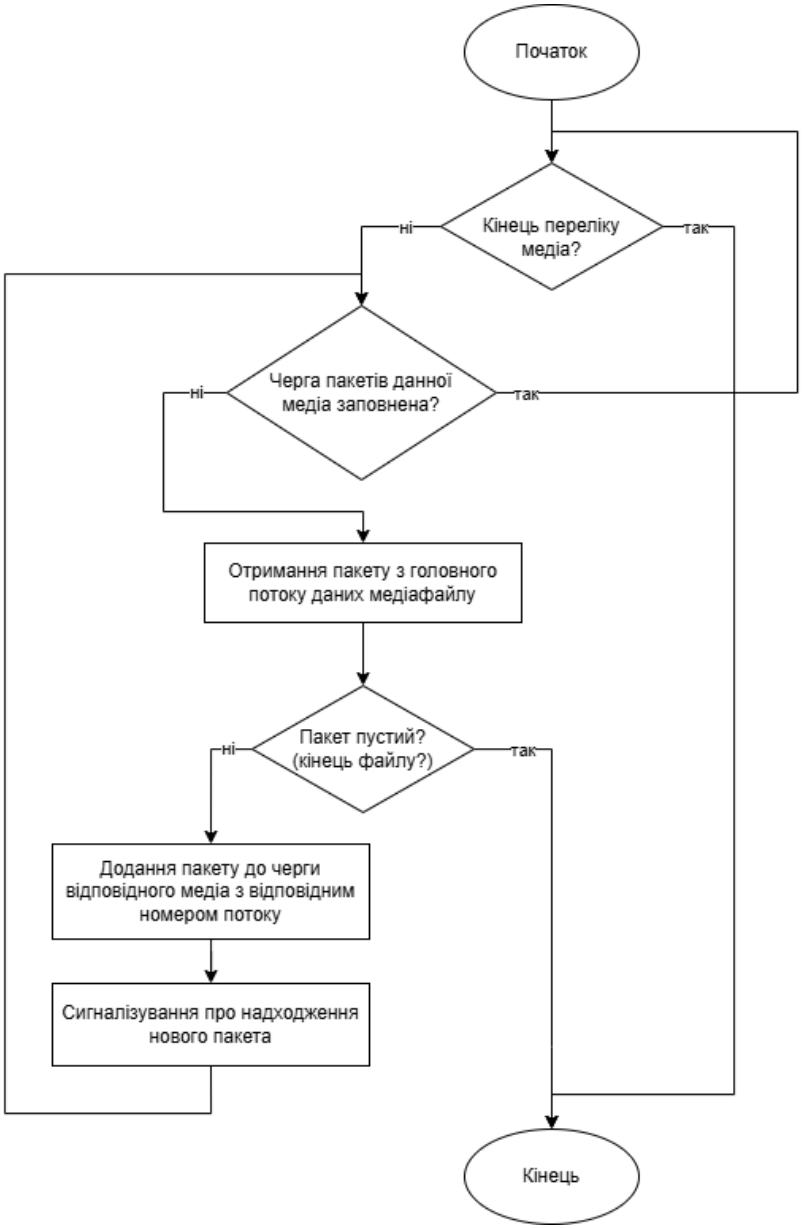


Рис. 11 – Алгоритм демультиплексування

На початку процесу демультиплексування запускається цикл проходу по списку наявних медіа, де перевіряється, чи не заповнена їх черга пакетів. У випадку неповної черги пакетів, викликається метод демультиплексування медіафайлу. Дістається наступний пакет з поточної позиції медіафайлу, і додається до черги пакетів медіа з відповідним номером потоку що і у отриманого пакета. Метод демультиплексування викликається до моменту, поки не будуть заповнені всі медіа черги пакетів.

3.4.4 Алгоритм декодування, накладання фільтрів та конвертації

Декодування — це процес перетворення закодованого (стисненого) медіапотоку у сиру (нестиснену) форму, яку може обробляти або виводити пристрій.

Конвертація — це процес зміни формату, структури або властивостей медіаданих після декодування.

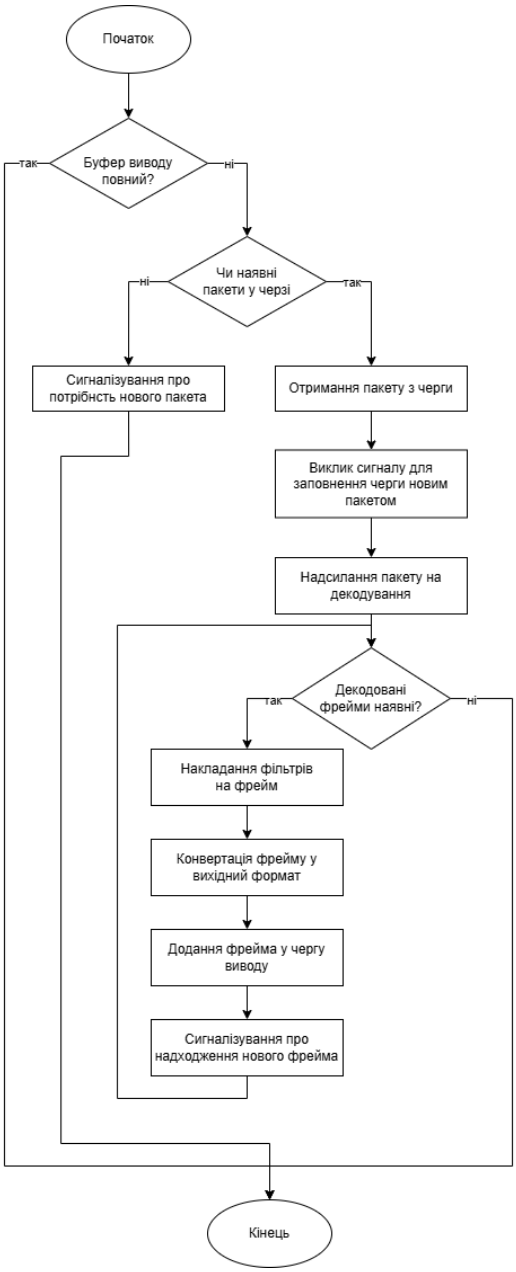


Рис. 12 – Декодування, накладання фільтрів та конвертація

На початку алгоритму перевіряється, чи повний буфер виводу. Якщо буфер повний, то алгоритм закінчується. Інакше далі перевіряється, чи наявні пакети у черзі для декодування. Якщо черга пуста, то сигналізується про необхідність нових пакетів, і алгоритм закінчується. Якщо черга не пуста, то з неї дістається пакет, викликається сигнал для заповнення черги новим пакетом. Пакет надсилається на декодування. Після декодування ми отримуємо послідовність декодованих фреймів, для кожного з яких ми: накладаємо фільтри, конвертуємо у вихідний формат та додаємо у буфер виводу. Після кожного додання фрейму у буфер, ми сигналізуємо про це. Коли декодовані фрейми закінчуються, алгоритм завершається.

3.4.5 Алгоритм синхронізації виводу відеокадрів

Синхронізація — це процес узгодження часових відміток (PTS — Presentation TimeStamp) аудіо та відео кадрів. Головна мета — забезпечити одночасне відтворення відповідних відеокадрів та звуку.

Як це працює у моїй системі:

- Після декодування ми отримуємо фрейм із часовою міткою (AVFrame->pts).
- Аудіопотік зазвичай працює непреривно — оскільки відтворення аудіо в реальному часі чутливе до затримок.
- Відеопотік підлаштовується до аудіо: відображення кадрів відбувається у відповідності до аудіо-часу.

- Система синхронізації за допомогою класу Synchronizer визначає, коли саме відображати черговий кадр на основі поточного часу.

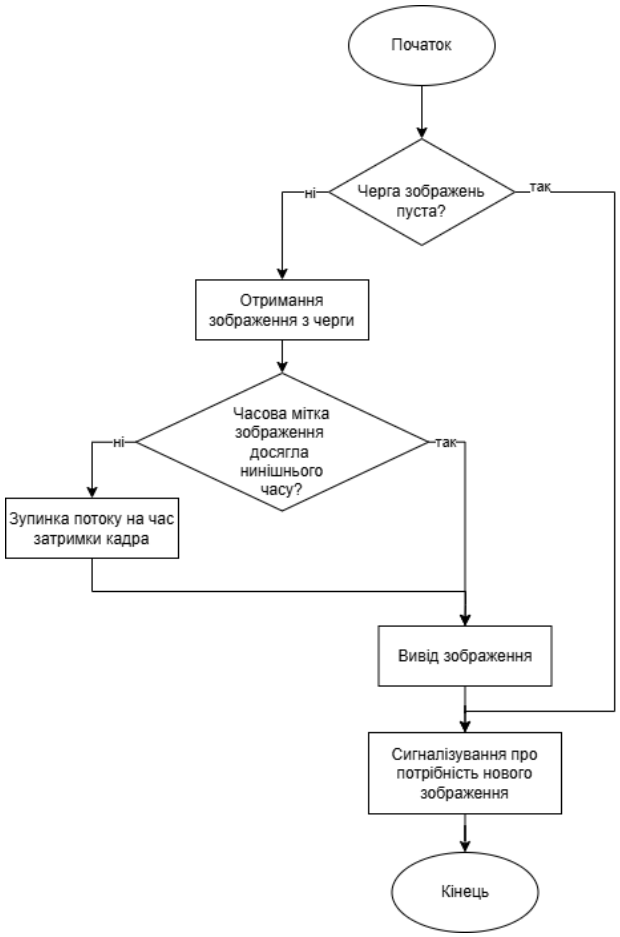


Рис. 13 – Синхронізація виводу відеокадра

На початку алгоритмму синхронізації виводу відеокадрів перевіряється, чи наявні зображення у черзі виводу. Якщо черга пуста, то викликається сигнал про необхідність нових зображень і алгоритм завершається. Якщо зображення наявні, то відеокадр дістається з черги, перевіряється його часова мітка (PTS). Якщо часова мітка кадра досягла нинішнього часу, то кадр виводиться, інакше потік призупиняється на час затримки, після чого кадр так само виводиться. Перед закінченням алгоритму сигналізується про необхідність нового кадра.

Детальний код всіх алгоритмів дивіть у Додатку Б.

3.5 Висновки

У третьому розділі було здійснено комплексну розробку інформаційного та програмного забезпечення медіаплеєра. Розглянуто архітектурні особливості побудови системи управління інформаційною базою, в тому числі обґрунтовано її специфічну реалізацію без традиційного сховища даних. Детально проаналізовано формати збереження медіафайлів, особливості контейнерів, принципи роботи кодеків та важливість стиснення інформації для зменшення обсягів передавання та зберігання.

Важливе місце у розділі відведено опису внутрішньої структури медіафайлів, процесам демультиплексування, декодування та синхронізації потоків, які реалізуються у системі. Це дозволяє забезпечити плавне та синхронізоване відтворення мультимедійного контенту. Окремо було обґрунтовано вибір інструментарію для розробки, включаючи мову C++, бібліотеку Qt та бібліотеки libav для роботи з медіа.

Завершальною частиною розділу стало алгоритмічне описання ключових модулів, зокрема сигналів і слотів Qt, розділення потоків медіа, накладання фільтрів, конвертації та відображення кадрів.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇСИСТЕМИ

У цьому розділі наведено основні аспекти, що стосуються впровадження, тестування та подальшої експлуатації розробленої системи. Описано процес перевірки її працездатності, визначено апаратні та програмні вимоги для забезпечення коректної роботи, а також представлено склад інсталяційного пакету. Ці рекомендації є важливими для забезпечення стабільної та ефективної роботи програмного забезпечення у реальних умовах експлуатації.

4.1 Тестування системи

Тестування програмного забезпечення є критично важливим етапом життєвого циклу розробки, що дозволяє виявити та усунути помилки до впровадження системи в експлуатацію. Метою тестування є перевірка правильності, надійності, продуктивності та зручності використання програмного продукту відповідно до вимог, визначених на етапі проєктування.

У процесі тестування медіаплеєра було проведено наступні види тестування:

- Функціональне тестування — перевірялася робота основних функцій: відтворення відео- та аудіофайлів, пауза/відновлення, перемотування, зміна гучності, перемикання фільтрів, зміна розміру вікна, зум, зміна швидкості відтворення, синхронізація звуку і відео. Функції працювали відповідно до очікувань у більшості сценаріїв.
- Тестування стабільності — перевірявся тривалий запуск програми з відтворенням великої кількості різних медіафайлів. Система показала стабільність роботи без витоків пам’яті чи збоїв.

- Тестування з некоректними файлами — було перевірено поведінку системи при відкритті пошкоджених або не підтримуваних файлів. У більшості випадків плеєр коректно виводив повідомлення про помилку.
- Перевірка продуктивності — оцінювався FPS, навантаження на процесор та оперативну пам’ять під час декодування медіа. Показники залишались у допустимих межах для сучасних ПК.
- Тестування інтерфейсу користувача — оцінювалась зручність користування елементами керування. Була перевірена адаптивність інтерфейсу до різних розмірів вікна та різних DPI екранів.

Для ручного функціонального тестування використовувались реальні сценарії використання. Для перевірки стабільності запускались серії автоматичних скриптів на основі QtTest та простого логування внутрішніх помилок.

Було проведено окреме тестування функціональних модулів. Тестування здійснювалось вручну, із застосуванням послідовних сценаріїв, орієнтованих на реальне використання медіаплеєра:

Тестування функції відкриття медіафайлів. На цьому етапі перевірявся механізм завантаження локальних відео- та аудіофайлів через графічний інтерфейс. Було протестовано файли з різними розширеннями та різними кодеками. Також перевірялася реакція системи на пошкоджені або неіснуючі файли — у результаті плеєр виводив повідомлення про помилку без аварійного завершення роботи.

Тестування декодування та відтворення. Особливу увагу було приділено перевірці модуля декодування відео- та аудіопотоків. Було протестовано відтворення відео у форматах H.264, H.265 та аудіо у форматах AAC, MP3. Система демонструвала плавне відтворення та синхронізацію аудіо і відео.

Тестування управління відтворенням. Тестувались такі функції, як пауза, відновлення, перемотування вперед/назад, перемикання швидкості. Кожна дія

активувалась як за допомогою клавіатурних комбінацій, так і через елементи інтерфейсу. Було перевірено коректну зміну станів програвача та виведення відповідного кадру після кожної дії.

Тестування фільтрів. Модуль фільтрації зображення дозволяє накладати базові ефекти (яскравість, контрастність, насиченість). Під час тестування фільтри вмикались/вимикались у реальному часі, і перевірялась їх миттєва дія на поточний відеопотік без затримок.

Тестування інтерфейсу користувача. Було перевірено коректність відображення кнопок, елементів керування, шкали прогресу. Інтерфейс був протестований на кількох розмірах вікна, щоб впевнитись у його адаптивності. Також проводилась перевірка реакції інтерфейсу на події користувача (наведення, натискання, перетягування).

Тестування на різних типах файлів. З метою перевірки універсальності, система тестувалась на файлах з різними параметрами: частотою кадрів, розміром кадру, бітрейтом. В усіх випадках було забезпечено правильне декодування без візуальних або звукових артефактів.

Результати тестування свідчать про достатню готовність системи до експлуатації. Усі критичні помилки були усунені, а виявлені незначні недоліки не впливають на основну функціональність продукту.

4.2 Вимоги до апаратного та програмного забезпечення

Для забезпечення стабільної та ефективної роботи розробленої системи перегляду медіафайлів необхідно дотримуватись певних мінімальних та рекомендованих вимог як до апаратного, так і програмного забезпечення.

Мінімальні апаратні вимоги:

- Процесор: 2-ядерний CPU з частотою від 1.6 ГГц.
- Оперативна пам’ять: 1 ГБ.
- Графічний адаптер: підтримка OpenGL.

- Місце на диску: щонайменше 200 МБ для самої програми + місце для медіафайлів.
- Роздільна здатність екрана: від 1280×720 пікселів.

Рекомендовані апаратні вимоги:

- Процесор: 4-ядерний CPU з частотою від 2.5 ГГц.
- Оперативна пам’ять: 4 ГБ або більше.
- Графічний адаптер: Дискретна відеокарта апаратним декодуванням відео.
- Місце на диску: SSD з мінімум 500 МБ вільного простору.
- Роздільна здатність екрана: Full HD (1920×1080) і вище.

Програмні вимоги:

- Операційна система: Windows 10/11, Linux (Ubuntu 20.04+), macOS 11+
- Qt Framework: Qt 6.6 або новіший (з підтримкою QML та мультимедійних модулів).
- FFmpeg / Libav: для декодування та обробки медіа.
- CMake: для збірки проєкту.
- Git: для керування версіями коду.

Додаткові вимоги:

- Підтримка доступу до відеофайлів із локального диска
- Права на читання/запис у робочу директорію програми
- Встановлені бібліотеки залежностей.

Ці вимоги забезпечують належну якість відтворення, обробки та керування медіаконтентом, дозволяючи користувачеві комфортно працювати з розробленим програмним забезпеченням навіть на середньостатистичному ПК.

4.3 Склад інсталяційного пакету

Інсталяційний пакет розробленого програмного забезпечення містить усі необхідні компоненти, що забезпечують повну працездатність системи на кінцевому пристрої користувача без потреби додаткового налаштування або встановлення сторонніх бібліотек вручну.

Основні компоненти інсталяційного пакету:

- Виконуваний файл програми. Основний .exe (для Windows) або виконуваний файл (для Linux/macOS), який є зібраним результатом проєкту.
- Бібліотеки Qt. Динамічні бібліотеки (Qt6Core.dll, Qt6Gui.dll, Qt6Qml.dll, Qt6Multimedia.dll), необхідні для роботи графічного інтерфейсу, мультимедіа та логіки програми.
- QML-модулі та компоненти (qml/ директорія з елементами інтерфейсу).
- Бібліотеки для роботи з медіа. Зовнішня бібліотека libav, яка відповідає за декодування відео та аудіо, демультиплексування та конвертацію потоків. У вигляді файлів динамічних бібліотек avcodec.dll, avformat.dll, avutil.dll.
- Файли ресурсів. Зображення та іконки, необхідні для повноцінного функціонування інтерфейсу. Включаються у вигляді окремої папки resources/ або інтегруються в .qrc файл Qt.
- Ліцензії. Текстові файли з інформацією про ліцензії використаних бібліотек (LGPL для Qt, GPL та FFmpeg).
- Інсталяційний скрипт або інсталятор. Готовий інсталятор, який автоматично встановлює програму на комп’ютер користувача, створює ярлики.

Структура каталогу після встановлення:

- resources/
- MediaPlayer.exe
- Qt6Core.dll
- Qt6Gui.dll
- Qt6Qml.dll
- avcodec.dll
- avformat.dll
- LICENSE.txt

Такий склад пакету забезпечує повну автономність програми, дозволяючи користувачу почати роботу одразу після інсталяції, без необхідності встановлення додаткового ПЗ.

4.4 Висновки

У даному розділі було розглянуто ключові аспекти впровадження та експлуатації розробленої системи перегляду медіафайлів. Проведено тестування функціональних модулів, що підтвердило стабільну роботу програми та правильність реалованих алгоритмів обробки медіаконтенту. Окремо проаналізовано вимоги до апаратного та програмного забезпечення, які дозволяють ефективно запускати програму навіть на комп’ютерах із середніми технічними характеристиками. Також сформовано інсталяційний пакет, що включає всі необхідні бібліотеки, ресурси та виконуваний файл, забезпечуючи простоту розгортання системи кінцевим користувачем. Це створює передумови для безперебійного впровадження програмного забезпечення та його подальшої експлуатації.

ВИСНОВКИ

У результаті виконання дипломної роботи було успішно розроблено програмне забезпечення для перегляду медіафайлів, що відповідає сучасним вимогам до зручності, функціональності та продуктивності. У першому розділі здійснено системний аналіз предметної області, визначено ключові функціональні, нефункціональні та системні вимоги, а також сформульовано чітке завдання для реалізації. На основі цього проведено моделювання предметної області, що дозволило сформувати логічну структуру системи.

У другому розділі реалізовано проєктування інформаційного та програмного забезпечення із використанням UML-діаграм, що відображають логіку взаємодії між компонентами системи. Особливу увагу приділено модульній структурі та зв’язкам між компонентами, що створює основу для надійного функціонування системи.

У третьому розділі виконано безпосередню розробку програмного продукту з використанням мови програмування C++, фреймворку Qt та графічного інтерфейсу на QML. Розглянуто алгоритми обробки медіафайлів — від демультиплексування до декодування, обробки фільтрами й синхронізації виводу, що забезпечує якісне відтворення відео. Розробка інтерфейсу була попередньо змодельована в Figma, що дозволило досягти інтуїтивної взаємодії з користувачем.

У четвертому розділі подано рекомендації щодо впровадження та експлуатації системи: описано процес тестування, зазначено вимоги до апаратного й програмного забезпечення, а також сформовано склад інсталяційного пакету для зручного розгортання.

Таким чином, усі поставлені в роботі цілі та завдання були досягнуті, що свідчить про ефективність обраного підходу до розробки, архітектурних рішень та інструментів. Розроблена система може бути використана як завершений програмний продукт або як основа для подальшого розширення функціональності.

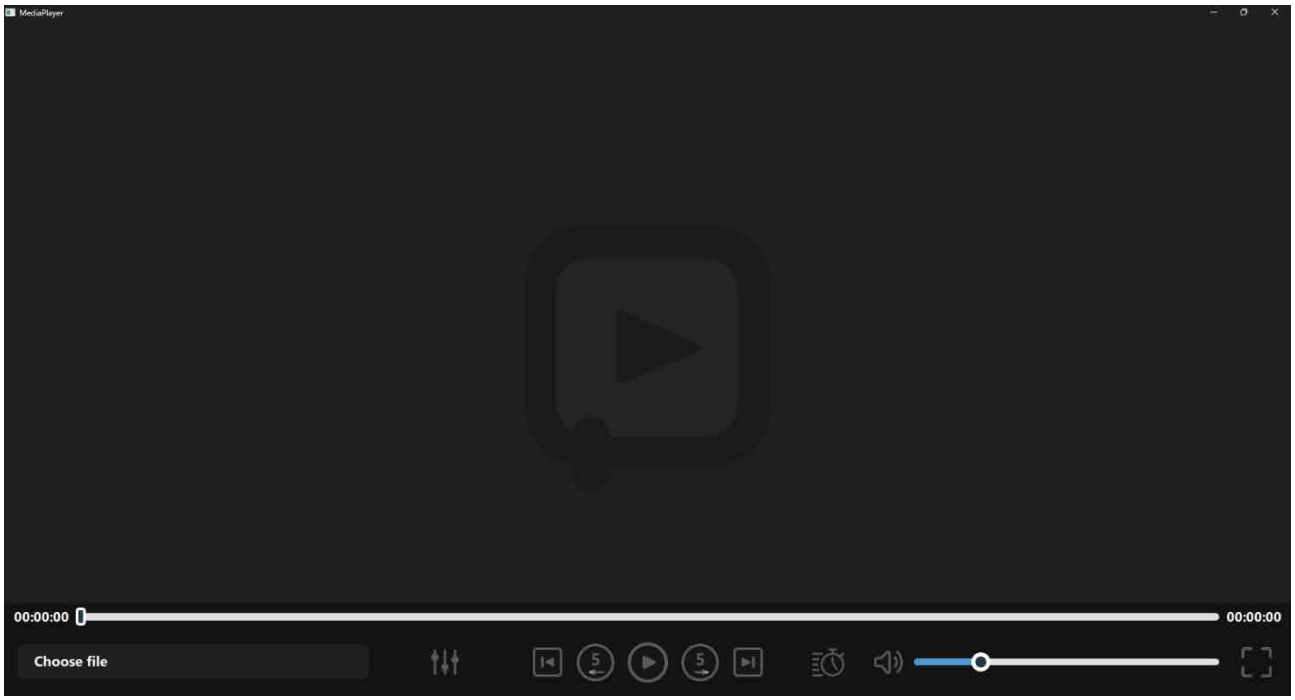
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Посібник із побудови схем UML і моделювання баз даних. URL: https://www.microsoft.com/uk-ua/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling (Дата звернення: 24.04.2025).
2. Media Introduction. FFmpeg Libav Tutorial. URL: https://github.com/leandromoreira/ffmpeg-libav-tutorial (Дата звернення: 15.03.2025).
3. FFmpeg Documentation. Офіційна документація FFmpeg. URL: https://ffmpeg.org/documentation.html (Дата звернення: 01.04.2025).
4. Qt Documentation. Офіційна документація Qt. URL: https://doc.qt.io/qt-6 (Дата звернення: 15.04.2025).
5. QML Documentation. Declarative UI design with QML. URL: https://doc.qt.io/qt-6/qmlapplications.html (Дата звернення: 07.04.2025).
6. Голуб Б.Л. Методичний посібник до вивчення дисципліни «Програмування та алгоритмічні мови». Навчальне видання Голуб Б.Л., Щукайло Є.М.-К-, Видавничий центр НАУ. 2003. – 64 с
7. Signals and Slots in Qt. Qt Documentation. URL: https://doc.qt.io/qt-6/signalsandslots.html (Дата звернення: 01.04.2025).
8. Моралес Дж. What is UML Class Diagram including UML Class Diagram Maker. MindOnMap. URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/ (Дата звернення: 24.04.2025).
9. ISO/IEC 14496-12:2022. Information technology – Coding of audio-visual objects – Part 12: ISO base media file format. ISO, 2022.
10. Blanchette J., Summerfield M. C++ GUI Programming with Qt 5. Prentice Hall, 2016. – 624 с.
11. Stroustrup B. The C++ Programming Language. Addison-Wesley, 2013. – 1376 с.

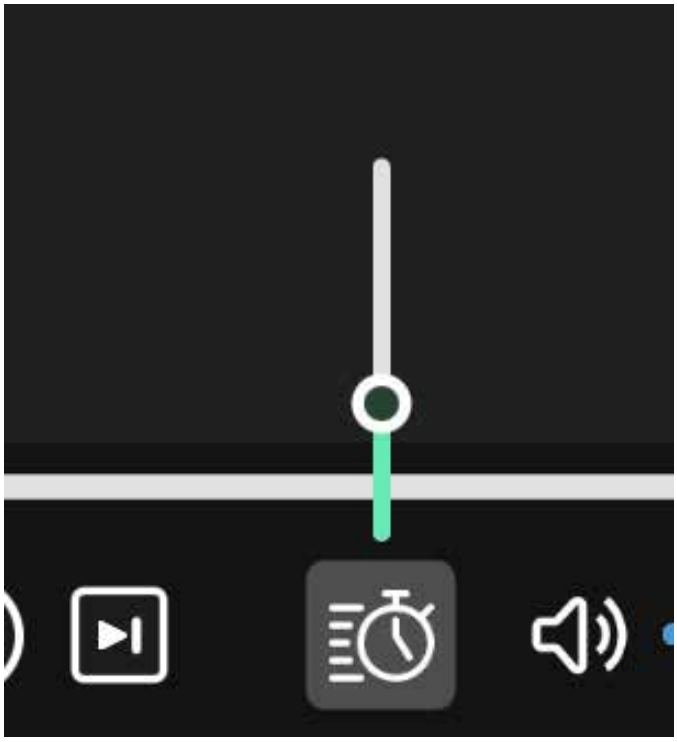
ДОДАТКИ

Додаток А

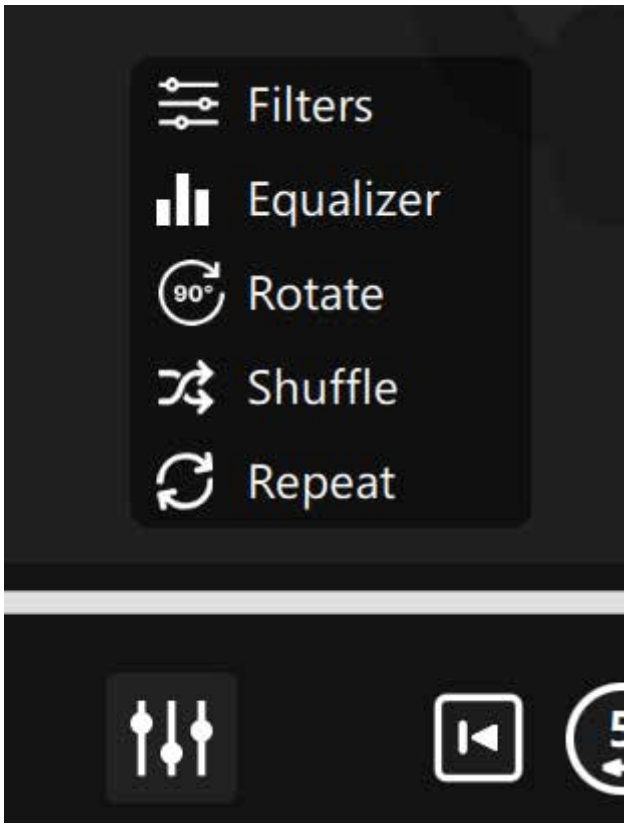
Головний екран інтерфейсу медіаплеєра



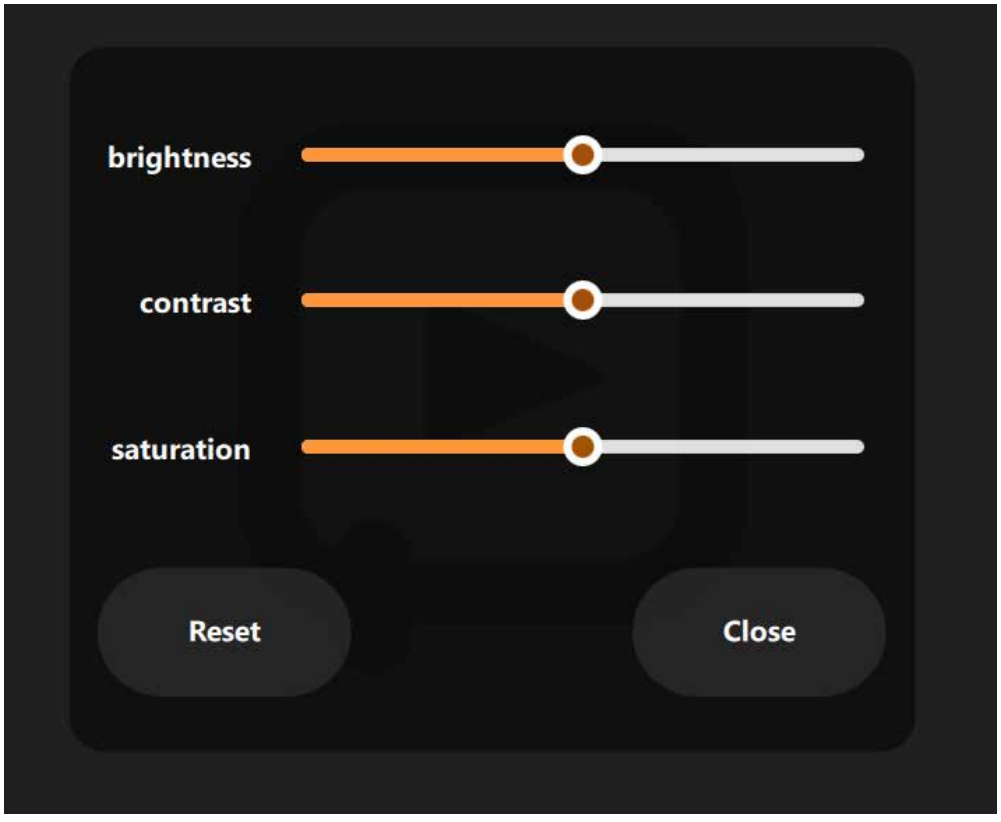
Меню налаштування швидкості



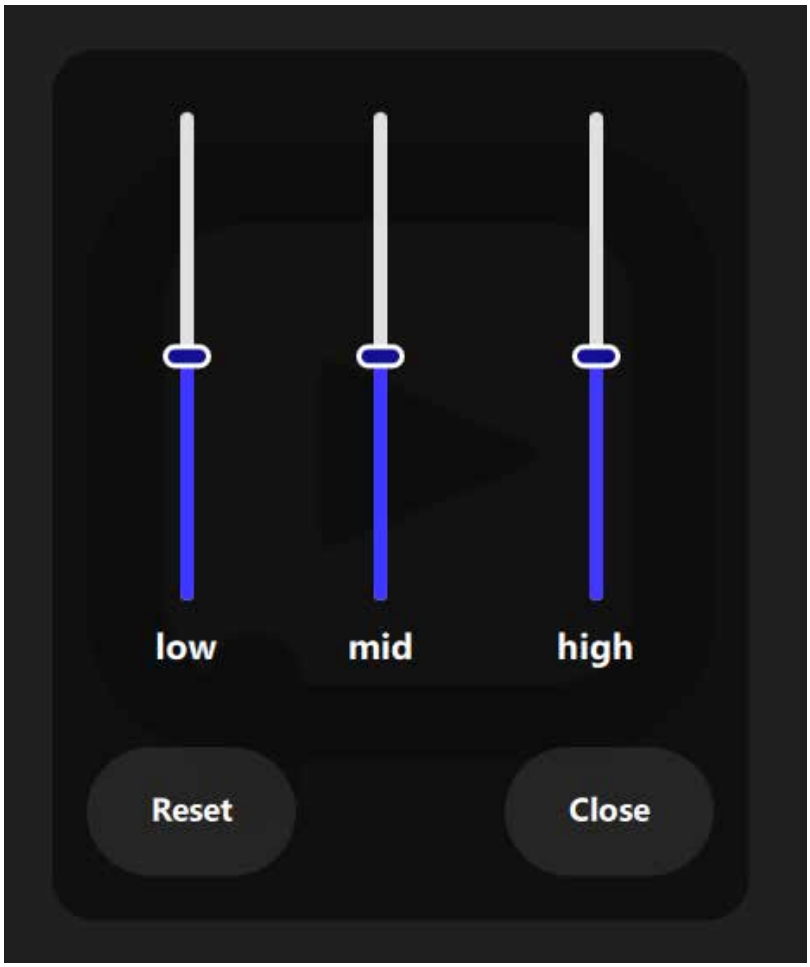
Меню додаткових функцій



Меню налаштування відео-фільтрів



Меню налаштування еквалайзеру



Додаток Б

Код функцій демультиплексування

```
void Demuxer::push_packets_to_queues()
{
    QMutexLocker _(&formatMutex);
    while(!is_queues_full())
    {
        Packet packet = make_shared_packet();
        int res = av_read_frame(format_context, packet.get());
        if (res<0){
            if (res == AVERROR_EOF)
                emit endReached();
            return;
        }

        if (medias.contains(packet->stream_index)){
            IMediaContext* media = medias[packet->stream_index];
            media->packetQueue.push(std::move(packet));
            emit media->newPacketArrived();
        }
    }
}

bool Demuxer::is_queues_full()
{
    for (auto& [_stream, context] : medias)
        if (!context->packetQueue.is_full())
```

```
            return false;
    return true;
}
```

Код функій декодування, фільтрації та конвертації

```
void AudioContext::decode_and_output()
{
    sync->check_pause();
    QMutexLocker _(&audioMutex);

    if (buffer_available() == 0)
        return;

    if (packetQueue.size() == 0){
        emit requestPacket();
        return;
    }

    Packet packet = packetQueue.pop();
    emit requestPacket();

    int result = avcodec_send_packet(codec_context, packet.get());;
    if (result < 0) {
        qDebug()<<"Error decoding audio packet: "<<result;
        return;
    }
```

```
    equalizer_and_output();
}

void AudioContext::equalizer_and_output()
{
    Frame frame = make_shared_frame();
    while (avcodec_receive_frame(codec_context, frame.get()) == 0)
    {
        Frame equalized_frame = equalizer->applyEqualizer(frame);
        if (!equalized_frame)
            continue;
        Frame converted_frame = converter->convert(equalized_frame);
        if (!converted_frame)
            continue;
        int size = converted_frame->nb_samples * converted_frame-
>ch_layout.nb_channels * format.bytesPerSample();
        audioDevice->appendData(QByteArray((const char*)converted_frame->data[0],
size));
        av_frame_unref(frame.get());
    }
}
```

Код функції синхронізації виводу відеокадрів

```
void AudioContext::decode_and_output()
{
    sync->check_pause();
```

```
    QMutexLocker _(&audioMutex);

    if (buffer_available() == 0)
        return;

    if (packetQueue.size() == 0){
        emit requestPacket();
        return;
    }

    Packet packet = packetQueue.pop();
    emit requestPacket();

    int result = avcodec_send_packet(codec_context, packet.get());;
    if (result < 0) {
        qDebug()<<"Error decoding audio packet: "<<result;
        return;
    }

    equalizer_and_output();
}

void AudioContext::equalizer_and_output()
{
    Frame frame = make_shared_frame();
    while (avcodec_receive_frame(codec_context, frame.get()) == 0)
    {
```

```
        Frame equalized_frame = equalizer->applyEqualizer(frame);
        if (!equalized_frame)
            continue;
        Frame converted_frame = converter->convert(equalized_frame);
        if (!converted_frame)
            continue;
        int size = converted_frame->nb_samples * converted_frame-
>ch_layout.nb_channels * format.bytesPerSample();
        audioDevice->appendData(QByteArray((const char*)converted_frame->data[0],
size));
        av_frame_unref(frame.get());
    }
}
```