Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка програмного забезпечення для керування медіавідтворенням на персональному комп'ютері»

Виконав: студент ___4___ курсу
групи __2ПІ-21б___
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Щерба Антон Олегович
(прізвище та ініціали)
Керівник: д-р. філос., асист каф. ПЗ О.В. Карась
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Черняк О. І.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ
д.т.н., проф. Романюк О. Н.
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Щербі Антону Олеговичу

1. Тема роботи – розробка програмного забезпечення для керування медіавідтворенням на персональному комп'ютері.

Керівник роботи: Карась Олександр Володимирович, д.ф., асист. каф. ПЗ Карась О.В., затверджені наказом вищого навчального закладу від «20» березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025р

3. Вхідні дані до роботи: Дані про поточний медіаконтент, отримані з API музичних сервісів; вихідні дані: візуальне відображення інформації про поточний медіаконтент; реакція на команди управління; інтерактивний графічний інтерфейс, який відображається поверх усіх вікон та реагує на дії користувача

4. Зміст текстової частини: вступ, аналіз розвитку систем керування медіавідтворенням та постановка задач дослідження, розробка моделі та алгоритмів програмного продукту, розробка програмних модулів системи керування медіавідтворенням, тестування програми, висновки, список використаної літератури, додатки.

5. Перелік ілюстративного матеріалу: титульний слайд, мета, об'єкт та предмет дослідження, задачі дослідження, новизна і практична цінність отриманих результатів, порівняльний аналіз аналогів, архітектура застосунку, загальний алгоритм роботи системи, алгоритм завантаження та відображення інформації про поточний трек, демонстрація інтерфейсу застосунку «Початковий екран», демонстрація інтерфейсу застосунку «Режим Spotify», демонстрація інтерфейсу застосунку «Режим Youtube», демонстрація інтерфейсу застосунку «Вибір альбому», тестування авторизації Spotify, тестування алгоритму управління аудіовідтворенням, апробація та публікація матеріалів бакалаврської дипломної роботи, фінальний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Карась О. В., д.ф., асист. каф. ПЗ	24.03.2025	01.06.2025

7. Дата видачі завдання 24 березня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи етапів роботи	Строк виконання	Примітка
1	Аналіз цифрових платформ для керування медіавідтворенням на персональному комп'ютері	25.03.2025-30.03.2025	викон.
2	Дослідження методів персоналізованих рекомендацій музичного контенту	31.03.2025-12.04.2025	викон.
3	Побудова моделей функціонування системи оренди та обміну музичними файлами	13.04.2025-30.04.2025	викон.
4	Розробка модулів авторизації, пошуку та рекомендацій з використанням Spotify API	01.05.2025-08.05.2025	викон.
5	Створення інтерфейсу користувача застосунку для медіавідтворення	09.05.2025-16.05.2025	викон.
6	Тестування програмного забезпечення та перевірка функціональності	17.05.2025-20.05.2025	викон.
7	Оформлення матеріалів до захисту БКР	21.05.2025-30.05.2025	викон.

Студент _____ **А. О. Щерба**
(підпис) (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи _____ **О. В. Карась**
(підпис) (ініціали та прізвище)

Анотація

УДК 004.04

На укр. мові. Бібліогр. : 20 назв ; рис. : 15; табл. 2.

Бакалаврська кваліфікаційна робота містить 100 сторінок формату А4, на яких наведено 11 рисунків та 2 таблиці, а список використаних джерел налічує 26 найменувань.

У межах бакалаврської кваліфікаційної роботи розроблено програмні модулі для системи керування медіавідтворенням з підтримкою популярних музичних сервісів Spotify та YouTube. Метою проєкту є створення компактного, функціонального та зручного інтерфейсу, який надає користувачеві можливість переглядати інформацію про поточне мідіавідтворення (назву, виконавця, обкладинку) та здійснювати базове керування відтворенням у реальному часі — зокрема, ставити на паузу, перемикати треки та регулювати гучність.

Особливістю реалізованого застосунку є відображення інтерфейсу поверх усіх відкритих вікон у правому верхньому куті екрана, що забезпечує швидкий доступ до елементів керування без потреби перемикання між програмами. Для отримання інформації про медіаконтент реалізовано інтеграцію з API сервісів Spotify і YouTube.

Розробка здійснювалася із використанням мов програмування Python у середовищі PyCharm. Створені програмні компоненти можуть бути використані як частини настільних мультимедійних застосунків або вбудовані у системи контролю аудіовідтворення.

Ключові слова: програмні модулі, система керування медіавідтворенням, API, інтеграція медіаплатформ, відтворення медіа.

Abstract

UDC 004.04

Shcherba A. O. Software Development for Media Playback Control on PC: Bachelor's Qualification Thesis in Specialty 121 Software Engineering, Educational Program – Software Engineering. Vinnytsia: VNTU, 2025. 100 pages.

In Ukrainian. Bibliography: 26 sources; illustrations: 11; tables: 2.

As part of the bachelor's qualification thesis, software modules were developed for a media playback control system supporting popular music services such as Spotify and YouTube. The project's goal is to create a compact, functional, and user-friendly interface that allows users to view information about the current media playback (title, artist, cover art) and perform basic playback control in real time – including pausing, switching tracks, and adjusting volume.

A key feature of the developed application is its ability to display the interface above all open windows in the top-right corner of the screen, ensuring quick access to control elements without the need to switch between programs. To retrieve media content information, integration with the Spotify and YouTube APIs was implemented.

The development was carried out using the Python programming language in the PyCharm environment. The created software components can be used as parts of desktop multimedia applications or embedded into audio playback control systems.

Keywords: software modules, media playback management system, API, media platform integration, media playback.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасних операційних системах спостерігається тенденція до впровадження більш інтуїтивно зрозумілих засобів керування медіаконтентом, що забезпечують користувачу доступ до інформації про трек та елементи керування без необхідності переходу до основного вікна відтворення. Серед сучасних рішень, переважають інструменти, орієнтовані на певну платформу або системну інтеграцію, але вони здебільшого не підтримують одночасну роботу з кількома сервісами та не дозволяють налаштувати поведінку елементів інтерфейсу під потреби користувача [1, 2].

Станом на сьогодні відсутні універсальні програмні рішення, які б поєднували:

- асинхронне виявлення змін у медіапараметрах у реальному часі;
- багатоплатформену підтримку;
- реактивне оновлення інтерфейсу без втручання користувача.

Що створює певний вакуум у сегменті програм для зручного керування медіа в багатозадачному режимі. У той час, як більшість існуючих рішень орієнтовані на ручну взаємодію або обмежені контекстом певного програвача, потреба у системі, що забезпечує динамічну інтеграцію з різними платформами та оперативну реакцію на зміну контенту, залишається актуальною.

Проблематика побудови реактивних медіаінтерфейсів [3,4] із підтримкою кількох джерел наразі перебуває на стадії первинних реалізацій або дослідних прототипів, а їхнє масштабування й оптимізація для реального використання залишаються відкритими задачами. Це дозволяє стверджувати, що тема дослідження недостатньо розроблена у частині комплексного вирішення задачі багатопотокового відстеження змін медіа та асинхронного керування в інтерфейсі користувача.

Мета та завдання дослідження. Підвищення зручності керування багатоплатформеним медіавідтворенням у середовищі операційної системи шляхом розробки програмного засобу з реактивним графічним інтерфейсом, що

здійснює асинхронне виявлення змін у стані медіа та забезпечує інтеграцію з різними сервісами (Spotify, YouTube, SoundCloud).

Завдання дослідження:

– провести аналіз сучасних інтерфейсних рішень для керування медіавідтворенням;

– сформувати функціональні та нефункціональні вимоги до системи асинхронного контролю медіа;

– розробити архітектуру програмного рішення з підтримкою багатопотокового моніторингу змін параметрів медіа;

– реалізувати подієву модель реактивного оновлення інтерфейсу користувача;

– забезпечити інтеграцію з API Spotify, YouTube та SoundCloud для отримання актуального медіаконтенту;

– реалізувати графічний інтерфейс з автоматичним відображенням інформації про трек та елементами керування;

– провести тестування працездатності системи в умовах реального часу;

– розробити інструкцію користувача.

Об’єкт дослідження. Об’єктом дослідження є процес керування відтворенням мультимедійного контенту з урахуванням його динамічних змін у середовищі операційної системи.

Предмет дослідження. Предметом дослідження є методи асинхронного моніторингу змін у стані медіавідтворення та засоби реактивного оновлення інтерфейсу користувача для керування контентом з кількох платформ.

Методи дослідження. У роботі використано методи системного аналізу для формування вимог; багатопотокове програмування на Python для моніторингу стану медіа; об’єктно-орієнтоване проєктування для побудови архітектури застосунку; подієве моделювання для реалізації реактивної логіки; інтерфейсне програмування з використанням бібліотеки customtkinter для створення GUI.

Наукова новизна отриманих результатів. Запропоновано асинхронну архітектуру виявлення та обробки змін медіа-стану в реальному часі, що поєднує багатопотоковий моніторинг параметрів системи (гучність, трек) із подієвою чергою для реактивного оновлення інтерфейсу без ручної взаємодії користувача. Такий підхід забезпечує миттєвий зворотний зв’язок та покращує інтерактивність медіаінтерфейсу [5].

Практичне значення одержаних результатів. Результати дослідження дозволяють підвищити зручність керування медіавідтворенням у багатозадачному середовищі, забезпечивши безперервний та уніфікований доступ до інформації про трек та інструментів управління незалежно від джерела контенту. Розроблене рішення може бути використане як у персональних десктопних середовищах, так і в корпоративних медіасистемах [6].

Особистий внесок здобувача. Усі результати дослідження, включно з формулюванням архітектури, реалізацією функціоналу та тестуванням програмного модуля, отримані автором особисто.

Апробаці матеріалів бакалаврської кваліфікаційної роботи. Матеріали доповідались на конфіренції «Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025)», Вінниця 2025.

Публікації. Результати дослідження були опубліковані в матеріалах конфреренції Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025) [7].

1 АНАЛІЗ РОЗВИТКУ СИСТЕМ КЕРУВАННЯ МЕДІАВІДТВОРЕННЯМ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану технологій керування медіавідтворенням

У сучасну цифрову епоху мультимедійний контент перетворився з простої форми розваг на постійного супутника користувача – на роботі, у транспорті, під час навчання чи дозвілля. Зростання популярності музичних сервісів, таких як Spotify, YouTube Music, Apple Music та SoundCloud, призвело до підвищеного попиту на зручні, зрозумілі та водночас багатофункціональні засоби керування медіавідтворенням. У цьому контексті актуальним стає створення застосунків, які поєднують простоту використання з широкими можливостями налаштування та сумісністю з різними платформами.

Базові функції керування мультимедіа зазвичай реалізуються на рівні операційних систем. Наприклад, у Windows використовується MediaSession API, який дозволяє працювати з активною медіасесією: керувати відтворенням, перемикати треки чи змінювати гучність. Аналогічні функції доступні і в macOS, і в Linux, проте вони рідко дають змогу змінювати зовнішній вигляд інтерфейсу або працювати з кількома джерелами аудіо одночасно. У більшості випадків користувач змушений задовольнятися стандартними елементами керування, які не відповідають сучасним вимогам персоналізації.

Сторонні медіаплеєри, такі як VLC, foobar2000 або AIMP, пропонують ширший набір функцій – підтримку локальних аудіофайлів, створення плейлистів, наявність еквалайзерів тощо. Водночас їм бракує повноцінної інтеграції з хмарними сервісами або потоковим відтворенням, а для взаємодії зі Spotify чи YouTube потрібно використовувати додаткові плагіни або API [8], що ускладнює інтеграцію та знижує зручність для звичайного користувача.

Короткочасні сповіщення про зміну треку чи рівня гучності, які зазвичай з’являються у верхній частині екрана, є типовим інтерфейсним рішенням у багатьох операційних системах. Хоча вони інформують про зміни, однак не дають

змоги безпосередньо керувати відтворенням – для цього користувачеві потрібно відкрити відповідний застосунок. У багатозадачному середовищі це створює певні незручності.

Сучасний підхід до керування відтворенням також включає використання відкритих API музичних платформ. Spotify та YouTube надають розробникам доступ до інформації про поточний трек, обкладинку, назву, виконавця, а також дозволяють змінювати чергу або стан відтворення. Ці можливості відкривають перспективи для створення індивідуальних застосунків із розширеним функціоналом. Водночас, при розробці таких рішень необхідно враховувати вимоги до авторизації, захисту персональних даних і стабільності мережевого з'єднання.

Незважаючи на наявність великої кількості інструментів, більшість із них мають певні обмеження. Одним із головних недоліків є відсутність компактного, постійно доступного інтерфейсу, що працював би поверх усіх вікон. Крім того, лише незначна частина програм здатна забезпечити одночасну інтеграцію з кількома сервісами, наприклад, Spotify і YouTube у межах єдиного інтерфейсу [9].

З урахуванням цих обставин, виникає необхідність створення застосунку, який дозволятиме керувати відтворенням без необхідності перемикання уваги з основного завдання. Такий інтерфейс має не лише відображати інформацію про трек, а й надавати можливість зручно регулювати гучність, перемикати композиції чи ставити їх на паузу. Оптимальним рішенням видається компактний модуль, розміщений у верхньому куті екрана, який постійно знаходиться поверх інших вікон і при цьому не заважає роботі користувача.

У підсумку можна стверджувати, що, незважаючи на значний поступ у сфері медіавідтворення, ще залишається відчутна потреба у спеціалізованих рішеннях, які б поєднували компактність, багатоплатформну інтеграцію, зручність та широкий функціонал.

1.2 Порівняльний аналіз аналогів

У процесі створення програмного забезпечення для керування медіавідтворенням важливим етапом виступає порівняльний аналіз наявних рішень на ринку. Це дає змогу оцінити рівень реалізації аналогічного функціоналу в інших застосунках і виявити недоліки, які можна усунути при розробці власного продукту. Основні критерії, за якими доцільно проводити оцінювання таких програм, включають зручність інтерфейсу, ступінь візуальної інтеграції з операційною системою, функціональність (наявність елементів управління, підтримка кількох платформ або сервісів), гнучкість налаштувань, а також рівень споживання системних ресурсів. Крім того, важливо враховувати частоту оновлень проєкту, активність спільноти чи розробника, відкритість програмного коду та стабільність роботи застосунку.

Одним із відомих прикладів подібного ПЗ є застосунок ModernFlyouts, зображений на рисунку 1.1 [10]. Цей застосунок створено як сучасна альтернатива стандартним спливаючим повідомленням у Windows, що з’являються при зміні гучності, яскравості, перемиканні треків або активації клавіш Caps Lock, Num Lock і Scroll Lock. Стандартні системні елементи Windows вважаються застарілими як у плані дизайну, так і за функціональними можливостями, тому ModernFlyouts було розроблено у відповідь на потребу користувачів у більш естетичному, гнучкому й зручному рішенні [11].

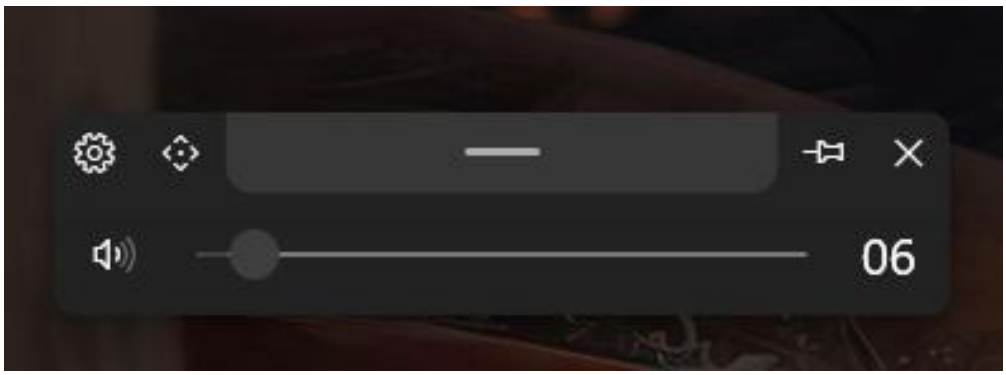

Рисунок 1.1 – Функціональний інтерфейс ModernFlyouts

ModernFlyouts вирізняється інтуїтивно зрозумілим і сучасним інтерфейсом, створеним на основі принципів Fluent Design System. Завдяки цьому програма органічно поєднується з візуальним оформленням Windows 10 і 11. Користувач має змогу не лише змінювати зовнішній вигляд системних повідомлень, а й налаштовувати їх розташування, прозорість, тривалість відображення та анімаційні ефекти. Така гнучкість дає змогу персоналізувати взаємодію з операційною системою відповідно до власних уподобань.

З технічного погляду, ModernFlyouts — це легкий застосунок із відкритим вихідним кодом, який активно підтримується спільнотою на платформі GitHub. Завдяки постійним оновленням програма отримує нові функції, оптимізації та виправлення виявлених помилок. Водночас вона характеризується низьким рівнем споживання системних ресурсів і працює у фоновому режимі, не заважаючи основним процесам користувача.

ModernFlyouts також надає можливість змінювати стиль і тему оформлення спливаючого вікна, а також переміщувати його на екрані. Однак, попри широкий спектр естетичних налаштувань, функціональність програми залишається обмеженою: вона не дозволяє безпосередньо керувати медіавідтворенням, не підтримує інтеграцію з потоковими сервісами й орієнтується переважно на взаємодію з локальними аудіопристроями. Таким чином, незважаючи на високу стабільність роботи та загальну зручність, застосунок не задовольняє потреби користувачів, які шукають розширені мультимедійні можливості.

Ще одним подібним рішенням є FluentFlyout [12] (рис. 1.2), який позиціонується як компактна альтернатива стандартному системному індикатору зміни гучності у Windows. Основна мета цього застосунку полягає в наданні користувачу мінімалістичного та швидкодіючого інтерфейсу, що не перевантажує систему зайвими візуальними елементами. Дизайн програми відповідає стилістиці Fluent Design і забезпечує візуальну узгодженість із сучасним середовищем Windows [13].

Завдяки простій архітектурі, FluentFlyout практично не навантажує систему й залишається ефективним навіть на малопотужних або застарілих пристроях. Програма працює у фоновому режимі, миттєво активуючи спливаюче вікно при зміні гучності. Вона не використовує складні графічні ефекти, що дозволяє пришвидшити взаємодію та не відволікати користувача.

Разом з тим, функціональність FluentFlyout є обмеженою — він не підтримує керування медіавідтворенням і не забезпечує інтеграції зі сторонніми сервісами, такими як Spotify чи YouTube. Через це застосунок слід розглядати як рішення для користувачів, які потребують лише базового відображення змін гучності без додаткових можливостей.

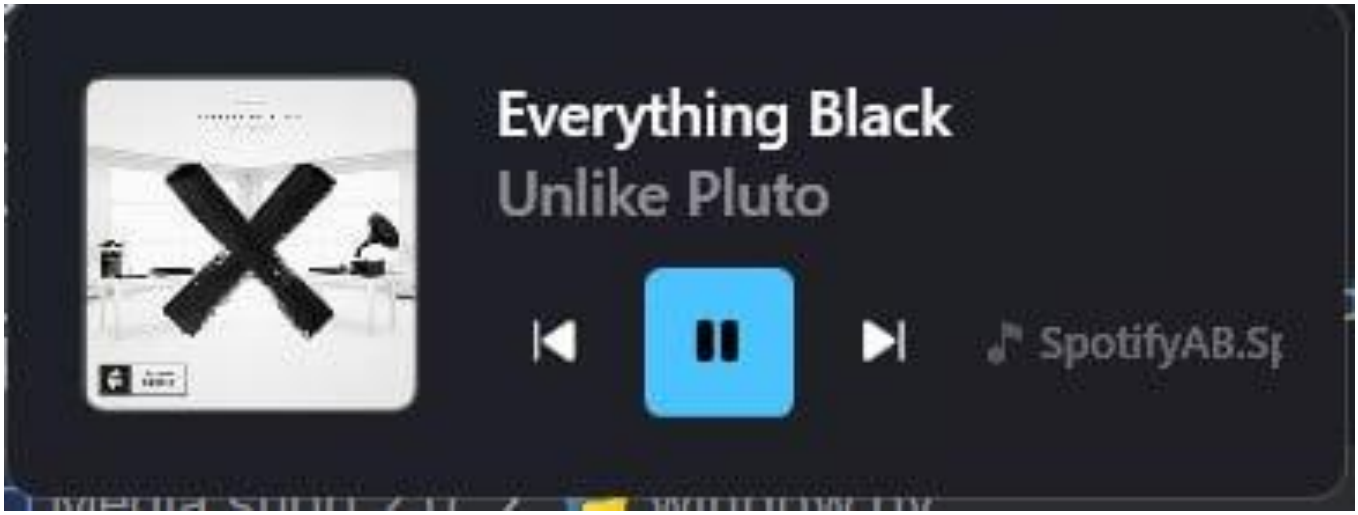

Рисунок 1.2 – Функціональний інтерфейс FluentFlyout

AudioFlyout [14] (рис. 1.3) є однією з найпростіших та водночас найефективніших альтернатив для базового керування гучністю в середовищі Windows. Основна ідея цього застосунку полягає в мінімалізмі та раціональному використанні системних ресурсів. Його інтерфейс зосереджений виключно на забезпеченні швидкого доступу до регулювання рівня звуку, без додаткових функцій чи складних меню.

Програма не перевантажена графічними елементами або анімаціями, що дозволяє їй працювати надзвичайно швидко та стабільно навіть на системах із

невисокими технічними характеристиками. Такий підхід робить AudioFlyout особливо привабливим для користувачів, які шукають простий та надійний інструмент без зайвого функціонального навантаження.

На відміну від інших рішень, таких як ModernFlyouts чи FluentFlyout, цей застосунок майже не пропонує можливостей кастомізації інтерфейсу або інтеграції з іншими службами. Відсутні опції керування медіавідтворенням, зміни тем оформлення чи підключення до потокових сервісів. Таким чином, AudioFlyout є доцільним вибором для користувачів, які потребують виключно базових функцій – зокрема регулювання гучності – і не мають потреби в розширеній мультимедійній інтеграції.

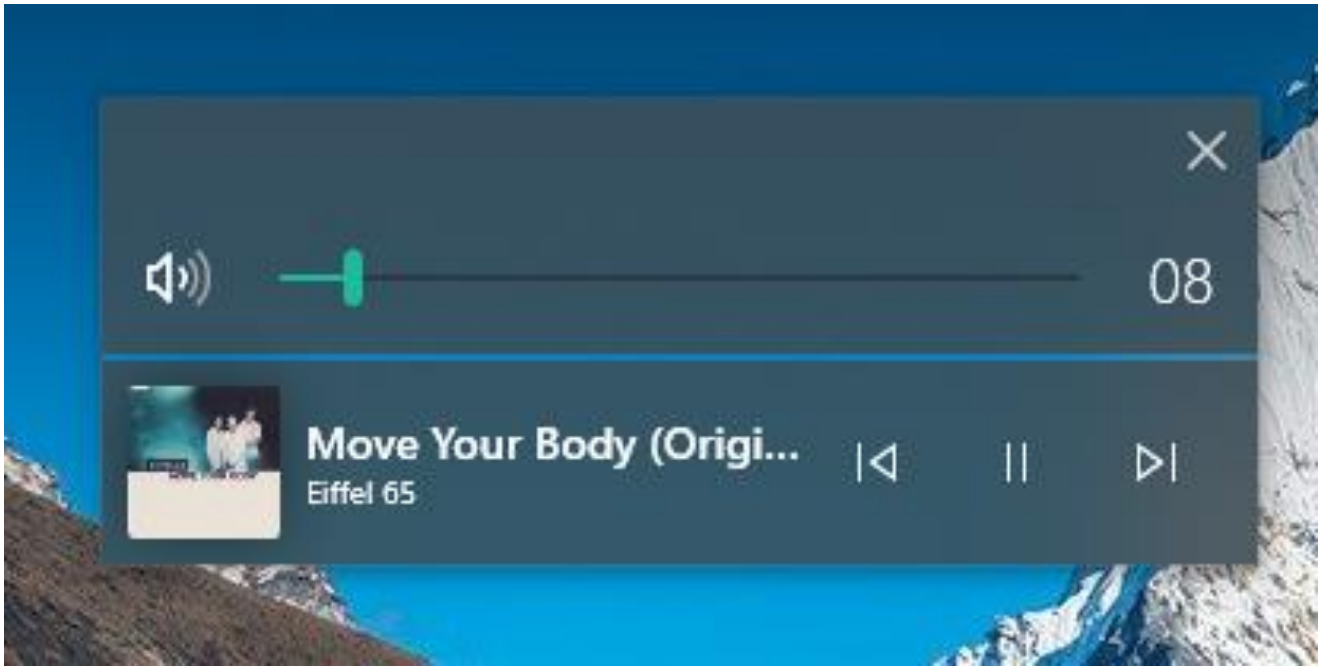

Рисунок 1.3 – Функціональний інтерфейс AudioFlyout

Однією з головних переваг AudioFlyout є його надзвичайно мала вага та мінімальне споживання системних ресурсів. Програма функціонує у фоновому режимі без створення додаткового навантаження на центральний процесор чи оперативну пам’ять, що робить її ідеальним рішенням для використання на

малопотужних або застарілих комп’ютерах. Такий підхід забезпечує стабільну та швидку роботу, навіть у системах з обмеженими технічними характеристиками.

Інтерфейс застосунку виконаний у стилі плоского (flat) дизайну з акцентом на функціональність та зручність. Всі елементи керування гучністю розташовані в компактному вікні, яке можна швидко викликати за допомогою гарячих клавіш. Візуальне оформлення є лаконічним – відсутні надлишкові анімації чи графічні ефекти, що дозволяє користувачеві зосередитися безпосередньо на основній задачі – регулюванні звуку. Чітко відображений слайдер гучності дозволяє точно налаштовувати рівень звуку відповідно до поточних потреб.

Незважаючи на ці переваги, функціональність AudioFlyout залишається обмеженою. Програма не підтримує інтеграцію зі сторонніми потоковими медіасервісами, такими як Spotify чи YouTube, не надає можливостей керування відтворенням (пауза, перемикання треків) та не відображає інформації про відтворюваний контент. Через це вона орієнтована насамперед на користувачів, які шукають простий та швидкий інструмент для зміни гучності без необхідності взаємодії з більш складними мультимедійними функціями.

Для більш вимогливих користувачів, які прагнуть комплексного керування мультимедійним контентом, AudioFlyout може здатися недостатньо функціональним. Водночас для базового сценарію використання – зручного регулювання гучності – цей застосунок залишається одним з найефективніших рішень.

У контексті розробки нових застосунків для керування медіавідтворенням важливо враховувати не лише ефективність використання ресурсів, а й можливість інтеграції з популярними потоковими сервісами. Створення таких програм передбачає глибоке розуміння принципів роботи з аудіо- та відеопотоками, використання відкритих API (наприклад, Spotify Web API, YouTube Data API), а також розробку інтуїтивно зрозумілих інтерфейсів, які забезпечують швидку взаємодію користувача з медіа. Важливо також передбачити підтримку багатоплатформенності, що дозволить охопити ширше коло користувачів.

Результати порівняння аналогів систем керування медіавідтворенням зведено в таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз аналогів

Критерії	Modern Flyouts	Fluent Flyout	Audio Flyout	Власний модуль
Підтримка кількох платформ	-	-	-	+
Підтримка кількох сервісів	+	-	-	+
Легкість налаштування	+	+	+	+
Компактність інтерфейсу	+	+	-	+
Наявність інтеграції з системою сповіщень	+	-	+	+
Загальна оцінка	80%	60%	60%	100%

Відповідно до проведеного порівняльного аналізу існуючих рішень, розробка власних програмних модулів для керування медіавідтворенням є обґрунтованою та доцільною. Виявлені обмеження у функціональності програм ModernFlyouts, FluentFlyout та AudioFlyout свідчать про наявність незадоволених потреб користувачів, зокрема в контексті багатосервісної інтеграції, управління відтворенням та персоналізації інтерфейсу.

Запропонований до розробки програмний продукт має на меті усунути недоліки аналізованих рішень, поєднуючи їх найкращі риси з новими можливостями. Зокрема, йдеться про створення інтерфейсу, що поєднує компактність і зручність, а також модулів, які забезпечують підтримку відтворення з кількох популярних медіасервісів (Spotify, YouTube, локальні плеєри тощо).

Отже, розробка власної системи керування медіавідтворенням відкриває широкі перспективи для підвищення комфорту взаємодії користувача з мультимедійним контентом. Завдяки інтеграції з різними сервісами, гнучкому налаштуванню та інтуїтивно зрозумілому інтерфейсу, майбутній продукт зможе забезпечити персоналізований, продуктивний і стабільний досвід роботи з аудіо- та відеовмістом, задовольняючи потреби як базових, так і просунутих користувачів.

1.3 Визначення оптимального функціоналу, обґрунтування та аналіз методів реалізації

Розробка програмного забезпечення для керування медіавідтворенням потребує комплексного підходу до вирішення завдань, пов’язаних зі збором інформації про медіаконтент, її обробкою, виведенням у компактному інтерфейсі та реалізацією інтуїтивного управління. Існує кілька підходів до створення таких систем, кожен з яких має свої переваги та обмеження.

Одним із варіантів є використання вбудованих засобів Windows, таких як стандартне flyout-вікно, яке з’являється при зміні гучності або керуванні відтворенням. Цей підхід має суттєві обмеження: обмежену підтримку медіасервісів, мінімальні можливості кастомізації та застарілий користувацький досвід. Він не дозволяє реалізувати сучасний інтерфейс або гнучке управління.

Іншим варіантом є адаптація відкритих проєктів на кшталт ModernFlyouts чи FluentFlyout. Зокрема, ModernFlyouts пропонує зручний інтерфейс на основі Fluent Design та підтримку стандартних мультимедійних подій. Водночас, його функціональність обмежена – додаток не підтримує глибоку інтеграцію зі сторонніми сервісами, такими як Spotify або YouTube. FluentFlyout є ще більш мінімалістичним і фактично не допускає розширення функцій поза межі відображення зміни гучності.

Найбільш гнучким та перспективним є створення власного програмного рішення з нуля із використанням сучасних фреймворків, таких як WPF, Electron, AvaloniaUI або MAUI. Такий підхід дозволяє:

– реалізувати підтримку популярних сервісів через їх API (Spotify, YouTube тощо) [16];

– виводити в інтерфейсі актуальні дані про відтворювану композицію: обкладинку, назву треку, виконавця;

– інтегрувати елементи керування відтворенням (пауза, наступний/попередній трек);

– створити адаптивний, кастомізований інтерфейс, який працює поверх усіх вікон і підтримує приховування;

На основі аналізу можливих підходів було прийнято рішення розробити власний модуль керування медіавідтворенням, що забезпечить максимальну гнучкість, масштабованість та сучасний користувацький досвід.

Основні завдання проєкту:

– Розробка модуля збору даних про поточний стан медіавідтворення з підтримкою платформ Spotify та YouTube;

– Розробка модуля обробки подій мультимедійного керування: попередній трек, наступний трек, пауза/відтворення;

– Створення компактного інтерфейсу, який відображає обкладинку треку, назву та виконавця;

– Реалізація керування відтворенням без необхідності відкривати відповідний застосунок;

– Забезпечення роботи інтерфейсу поверх усіх вікон із можливістю швидкого приховування/відновлення;

– Проведення тестування працездатності з різними сервісами у типових сценаріях;

– Оптимізація продуктивності для комфортної роботи на слабких ПК.

Запропонована концепція дозволяє створити програмний продукт, який перевершуватиме існуючі аналоги за функціональністю, ергономікою та адаптивністю. Такий підхід забезпечить сучасний, ефективний та приємний користувацький досвід незалежно від рівня технічної підготовки користувача або апаратних ресурсів системи.

1.4 Висновки

У першому розділі було проаналізовано сучасний стан технологій керування медіавідтворенням у цифрову епоху, коли мультимедійний контент став невід'ємною частиною повсякденного життя користувачів. Популярність музичних

сервісів, таких як Spotify, YouTube Music, Apple Music та SoundCloud, зумовила зростання потреби у зручних, функціональних та адаптивних інтерфейсах для керування відтворенням аудіо- та відеоконтенту.

Базові мультимедійні засоби, що надаються операційними системами, хоч і забезпечують виконання основних функцій, часто виявляються недостатніми у плані кастомізації, гнучкості та зручності користування. У той же час сторонні медіаплеєри (наприклад, VLC, foobar2000) пропонують розширені можливості, однак не завжди інтегруються з хмарними сервісами, що ускладнює їх повсякденне використання.

Наявні рішення для керування медіавідтворенням, зокрема сповіщення про зміну треків або гучності, не дають змоги здійснювати повноцінне управління контентом без відкриття відповідного застосунку. Це створює потребу у нових програмних рішеннях, які б об'єднували кілька сервісів в єдиний інтерфейс, забезпечуючи при цьому зручність, мінімалізм і ефективність.

Проведений аналіз існуючих аналогів вказує, що на ринку присутні окремі рішення (наприклад, ModernFlyouts, FluentFlyout), які пропонують компактні інтерфейси для мультимедійного управління. Проте ці рішення мають обмежену функціональність та не підтримують інтеграцію з потоковими платформами, такими як Spotify чи YouTube. Інші застосунки, як AudioFlyout, реалізують лише базові можливості (наприклад, регулювання гучності) та не дозволяють здійснювати гнучке керування мультимедійним відтворенням.

Таким чином, було зроблено висновок, що розробка власного модулю для керування медіавідтворенням є доцільною. Такий підхід дозволить:

– подолати недоліки існуючих рішень;

– інтегрувати підтримку кількох популярних медіасервісів;

– реалізувати компактний, адаптивний та інтуїтивно зрозумілий інтерфейс;

– забезпечити комфортне управління відтворенням без потреби відкривати сторонні застосунки.

Отже, створення власного модулю для керування медіавідтворенням є виправданим та обґрунтованим кроком, який спрямований на покращення користувацького досвіду шляхом об'єднання функціональності, зручності та широких можливостей налаштування в єдиному рішенні.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних даних системи

У процесі розробки програмного забезпечення для керування медіавідтворенням, що інтегрується з платформою Spotify, важливим етапом є аналіз вхідних даних, необхідних для належної роботи системи. Вхідні дані впливають не тільки на функціональність програми, але й на її взаємодію з користувачем та іншими компонентами системи, такими як аудіообладнання та API Spotify. Вони формують основну структуру програми, забезпечуючи її здатність працювати з зовнішніми платформами, налаштовувати інтерфейс і обробляти запити на відтворення та інші функціональні можливості [17].

Одним із основних типів вхідних даних є інформація про користувача, яка визначає доступ до певних функцій програми. Оскільки система інтегрована з Spotify, ці дані включають тип підписки користувача (Premium чи Free), який отримується через API Spotify. Тип підписки має суттєвий вплив на можливості додатка: користувачі з підпискою Premium мають доступ до таких функцій, як прослуховування без реклами, можливість пропускати треки без обмежень, а також офлайн-режим. Для користувачів без Premium доступ до деяких функцій обмежений. Інтерфейс програми автоматично адаптується до типу підписки, щоб забезпечити максимальний рівень досвіду користувача.

Інформація про користувача включає також аватар, ім’я та інші персональні дані, які можуть бути використані для персоналізації взаємодії з додатком. Це дозволяє створити орієнтований на користувача інтерфейс, що надає персоналізовані рекомендації на основі вподобань, слухацької історії та взаємодії з платформою Spotify. Програма здатна запам'ятовувати історію відтворення, улюблені треки та обрану музику, що дозволяє автоматично генерувати плейлисти та рекомендації на основі попередніх виборів користувача.

Метадані медіафайлів, що відтворюються через додаток, є другим важливим елементом вхідних даних. Це включає назву треку, ім’я виконавця, обкладинку альбому, тривалість треку та його позицію в поточному плейлісті. Ці дані постійно оновлюються в реальному часі, дозволяючи відображати актуальну інформацію на екрані користувача. Підтримка таких елементів, як ідентифікація поточного треку, пошук і фільтрація за жанром або виконавцем, значно полегшує навігацію в додатку і робить його більш зручним для користувача. Дані про трек отримуються через API Spotify, яке дозволяє запитувати інформацію про поточний трек, його метадані, такі як жанр, дата випуску та інші характеристики. Це критично важливо для високоякісного відтворення та синхронізації з музичною платформою.

Інтеграція з музичними рекомендаціями, що надаються через Spotify API, є ще однією важливою функцією. Рекомендації на основі слухацької активності користувача можуть включати нові релізи, поради від друзів, а також популярні треки в регіоні чи за певними вподобаннями. Це покращує досвід користувача, надаючи йому можливість відкривати нову музику, що відповідає його інтересам. Алгоритми машинного навчання аналізують слухацькі звички користувача і загальні тренди на платформі для створення таких рекомендацій.

Інтеграція з аудіообладнанням є критично важливою для системи. Параметри аудіообладнання, зокрема рівень гучності, стан вимкнення звуку та взаємодія з іншими аудіофункціями операційної системи, впливають на коректне відображення стану відтворення та адаптацію функціональних можливостей додатку. Для контролю рівня гучності використовуються зовнішні бібліотеки, такі як pycaw, що дозволяють коректно змінювати рівень звуку та реагувати на зміни в налаштуваннях гучності. Дані про рівень гучності отримуються безпосередньо з операційної системи і моніторяться в реальному часі, що дозволяє оперативно реагувати на зміни в налаштуваннях.

Система повинна дозволяти користувачеві змінювати гучність як через слайдер у інтерфейсі, так і за допомогою фізичних кнопок на пристрої. Крім того, необхідно враховувати стан вимкненого звуку, що дозволяє реалізувати безшумний

режим. Функціональність зміни гучності також повинна включати автоматичну корекцію гучності залежно від контексту (наприклад, зменшення гучності при підключенні до навушників).

Оновлення статусу медіаплеєра є ще одним важливим аспектом. Це охоплює зміни стану відтворення (пауза, відтворення, перемотка) та наявність активного з'єднання з сервером Spotify. Для цього використовуються API запити, які дозволяють отримувати дані про стан плеєра, активні плейлисти та інші параметри. Це дозволяє програмі бути більш чутливою до змін і забезпечує безперебійне відтворення медіафайлів, зберігаючи інтерактивний та стабільний досвід для користувача.

2.2 Розробка моделі системи

Розробка моделі системи є критично важливим етапом при створенні програмних компонентів для керування медіавідтворенням, оскільки саме на цьому етапі формується структура роботи програми, її взаємодія з користувачем, а також інтеграція з різноманітними зовнішніми сервісами. На цьому етапі було розроблено логіку роботи застосунку, яка забезпечує отримання медіаконтенту та інтеграцію з такими популярними платформами, як Spotify та YouTube. Це дозволяє створити універсальну платформу для управління різними джерелами медіа, забезпечуючи зручну взаємодію користувача з медіаконтентом незалежно від його джерела.

Першим кроком у розробці системи є налаштування основних компонентів, що включає підготовку середовища розробки, імпорт необхідних бібліотек для створення графічного інтерфейсу, обробки мультимедійних даних та інтеграції з зовнішніми сервісами. Для створення інтерфейсу використовуються Python-бібліотеки, зокрема customtkinter, яка дозволяє реалізувати інтуїтивно зрозумілий та функціональний інтерфейс. Важливою частиною є також інтеграція з API зовнішніх музичних сервісів, таких як Spotify і YouTube. Це дозволяє отримувати актуальні дані про медіаконтент, включаючи назви треків, виконавців, альбоми та інші метадані. Для обробки мультимедійних даних використовуються моделі, які

дозволяють програмі ефективно працювати з аудіофайлами та іншими типами медіа.

Одним із основних аспектів є створення головного вікна програми, яке має бути мінімалістичним, але зручним для користувача. Однією з ключових функцій є розташування компактного контролера в верхньому правому куті екрану. Це дозволяє користувачеві мати доступ до основних елементів управління медіавідтворенням навіть при роботі з іншими додатками чи вікнами операційної системи. Така організація забезпечує контроль відтворення медіа без необхідності постійного перемикання між вікнами, що значно підвищує зручність використання програми.

Після налаштування основних компонентів системи наступним етапом є авторизація користувача через API зовнішніх музичних сервісів. Для платформи Spotify застосовуються механізми авторизації на основі протоколу OAuth 2.0, що дозволяє забезпечити безпечний доступ до облікового запису користувача і отримати всі необхідні дані для інтеграції з програмою, такі як метадані треків, їх назви, виконавці, обкладинки альбомів та інша важлива інформація. Механізм авторизації через OAuth 2.0 є надійним та захищеним, що гарантує безпеку користувацьких даних [18]. Для сервісу YouTube також передбачено відповідний механізм авторизації та отримання медіаконтенту, що дозволяє додавати відео та музику з цієї платформи до списків відтворення користувача [19].

Після успішної авторизації користувач отримує доступ до всіх основних функцій програми, зокрема до інтерфейсу керування медіавідтворенням. Одним із перших етапів є реалізація елементів керування відтворенням, таких як кнопки для паузи, відновлення відтворення, а також перемикання між попереднім і наступним треками. Кожна з цих кнопок повинна бути інтерактивною і зручною для користувача, щоб він міг легко здійснювати основні операції з медіавідтворенням. Обробка подій, пов’язаних з натисканням кнопок, реалізується шляхом прив’язки команд до відповідних методів у коді програми, що дозволяє забезпечити належну логіку для керування відтворенням.

Одночасно з реалізацією елементів керування відтворенням важливим етапом є механізм отримання поточного стану відтворення. Програма повинна оновлювати інформацію про поточний трек, гучність та відображати обкладинку альбому та іншу метаінформацію в реальному часі. Це забезпечує актуалізацію даних на екрані, дозволяючи користувачеві завжди бачити, що саме відтворюється. Для цього в програмі передбачено регулярне оновлення стану відтворення при зміні треку або змінах у параметрах гучності, таких як зміна звуку через слайдер чи зовнішні аудіоінтерфейси.

Однією з особливостей цієї системи є її здатність працювати поверх усіх вікон операційної системи. Це дозволяє створити програму, яка завжди доступна для користувача, навіть коли він працює з іншими додатками або завданнями. Такий підхід забезпечує постійний доступ до елементів керування відтворенням, не відволікаючи користувача від основної роботи. Завдяки цьому програма стає надійним помічником у будь-якій діяльності користувача — чи то робота, навчання, чи розваги, дозволяючи йому легко контролювати медіавідтворення без перешкод.

Всі ці етапи розробки системи є необхідними для створення ефективного, стабільного та функціонального медіаплеєра, що інтегрується з такими популярними стрімінговими сервісами, як Spotify та YouTube. Послідовне проходження кожного з цих етапів дозволяє врахувати всі технічні вимоги, забезпечити надійність взаємодії з API сторонніх сервісів і реалізувати гнучку архітектуру, придатну для подальшого масштабування та оновлення. Підхід, орієнтований на зручність користувача, гарантує синхронну, узгоджену роботу всіх елементів системи, забезпечуючи інтуїтивно зрозумілий інтерфейс, швидкий відгук на дії користувача та найкращий досвід взаємодії з програмою.

Розроблена блок схема моделі системи зображенна на рисунку 2.1.

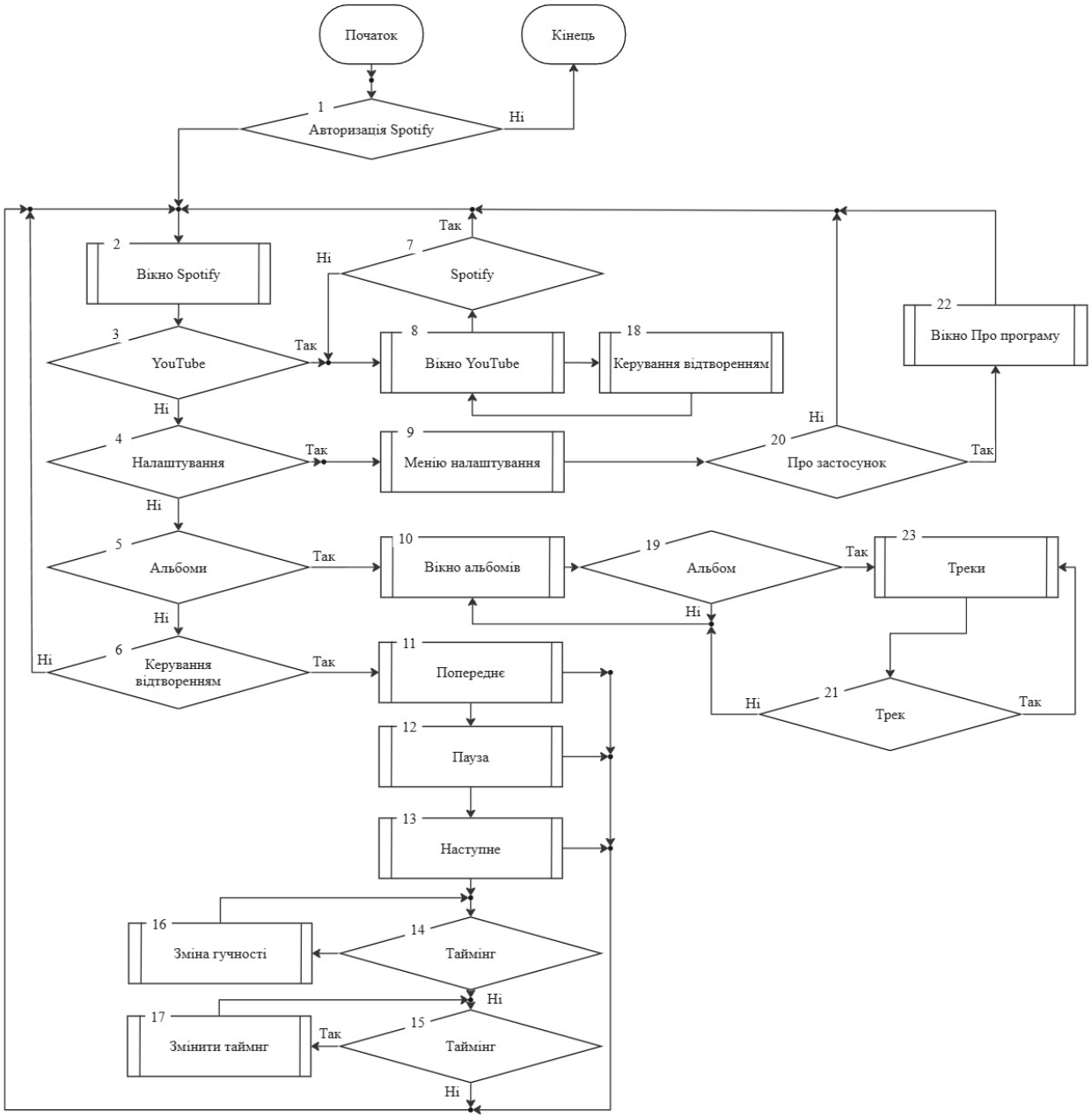

Рисунок 2.1 – Загальний модель роботи застосунку

Таким чином, розроблена модель системи ґрунтується на інтеграції даних із зовнішніх музичних сервісів, побудові простого та інтуїтивно зрозумілого інтерфейсу керування медіавідтворенням, обробці основних користувацьких дій (перемикання треків, зміна гучності, відтворення/пауза), а також забезпеченні постійної доступності контролера незалежно від активних вікон операційної системи. Такий підхід дозволяє створити зручний, ефективний та надійний

інструмент для взаємодії з мультимедійним контентом, орієнтований на комфорт і потреби кінцевого користувача.

2.3 Розробка алгоритмів роботи системи

Розробка алгоритмів функціонування програмного модуля медіаконтролера розпочалася з чіткого визначення послідовності ініціалізації та виконання основних операцій, необхідних для коректної роботи системи. Цей процес є критично важливим, оскільки від правильності початкової ініціалізації залежить стабільність усього застосунку.

На початковому етапі функціонування системи виконується завантаження середовища виконання, що включає в себе кілька ключових кроків. Перш за все, відбувається імпорт необхідних бібліотек, які відповідають за побудову графічного інтерфейсу користувача, зокрема елементів керування відтворенням, а також компонентів для виведення інформації про трек (обкладинки, назви композицій, імена виконавців). Окрім цього, імпортуються бібліотеки для реалізації інтеграції з зовнішніми музичними сервісами, такими як Spotify API та локальні механізми взаємодії з YouTube, що дозволяють отримувати метадані та здійснювати управління відтворенням.

Також на цьому етапі здійснюється підготовка інструментів для обробки графічних елементів, зокрема завантаження обкладинок альбомів, масштабування зображень, відображення статусу поточного відтворення тощо.

Одразу після імпорту бібліотек виконується зчитування конфігураційних параметрів, збережених у зовнішньому файлі settings.json [20]. Цей файл містить усі необхідні налаштування, що визначають поведінку застосунку — вибрану музичну платформу, останні авторизовані облікові дані, параметри інтерфейсу, рівень гучності та інші користувацькі уподобання. Для реалізації цього етапу використовується вбудований механізм збереження та відновлення налаштувань, який дозволяє забезпечити безперервність користувацького досвіду та уникнути необхідності повторного введення даних при кожному запуску програми.

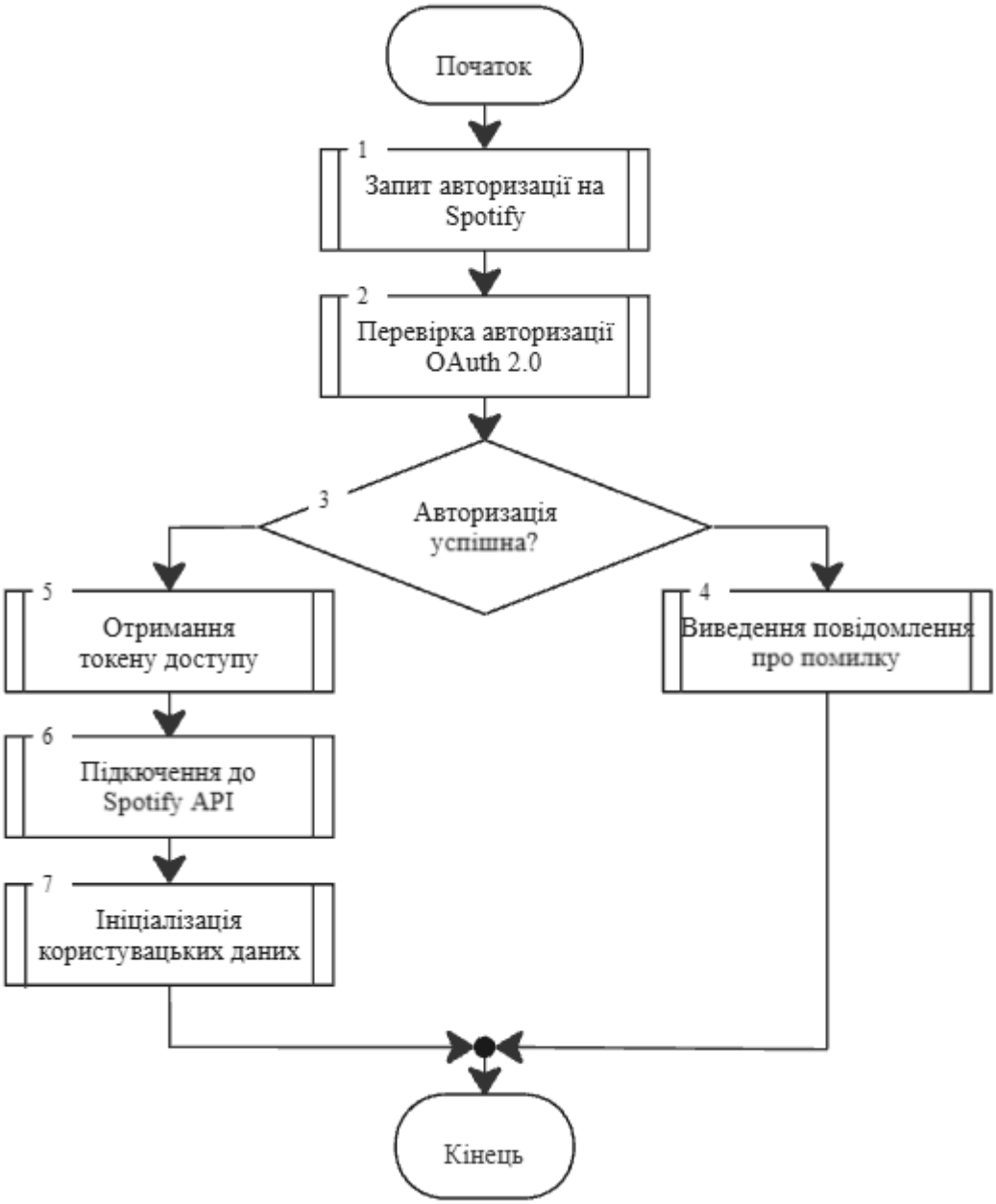

Рисунок 2.2 – Алгоритм авторизації Spotify

Процедура автентифікації користувача реалізована із застосуванням сучасних стандартів безпеки, зокрема протоколу OAuth 2.0 для музичної платформи Spotify та інструментів Google API для інтеграції з YouTube (рис. 2.2). Такий підхід гарантує безпечний обмін даними між користувачем і зовнішніми сервісами, оскільки доступ до особистої інформації надається лише після

отримання явної згоди користувача. Успішне проходження процедури авторизації дозволяє застосунку отримати тимчасові токени доступу, за допомогою яких забезпечується взаємодія з API відповідних платформ.

Після підтвердження користувачем відповідних дозволів на доступ, виконується автоматична ініціалізація графічного інтерфейсу застосунку. На цьому етапі створюється головне вікно програми, яке отримує пріоритетне відображення поверх інших активних вікон системи, що забезпечує постійний візуальний контроль над відтворенням медіа. Інтерфейс містить функціональний блок керування відтворенням, до якого входять кнопки: «Попередній трек», «Відтворити/Пауза» та «Наступний трек». Також у вікні розміщується регулятор гучності — як у вигляді повзунка, так і через кнопки зміни значення, а також елементи навігації, що дозволяють перемикатися між режимами роботи (наприклад, перехід на відтворення через YouTube), відкривати вікно з альбомами або запускати меню налаштувань.

Для забезпечення актуальності даних інтерфейсу в реальному часі, реалізовано паралельний моніторинг поточного стану відтворення треків і системного рівня гучності. Ці процеси виконуються у фонових потоках, що дозволяє не блокувати основний інтерфейс програми й забезпечити плавну взаємодію з користувачем.

Відстеження активного треку реалізується через періодичні запити до Spotify API. На кожному інтервалі часу отримується назва поточної композиції та виконується її порівняння з попереднім значенням. У разі виявлення змін (наприклад, якщо почав відтворюватися новий трек), формується подія оновлення, яка додається до черги подій і згодом обробляється інтерфейсом для оновлення відповідних віджетів.

Аналогічно, моніторинг системного рівня гучності здійснюється через інтерфейс AudioUtilities, що дозволяє отримати поточне значення гучності в операційній системі. У разі, якщо виявлено зміну гучності, яка перевищує визначений поріг (наприклад, ±1 %), генерується подія, яка також надходить до

черги. Це дозволяє забезпечити миттєве відображення змін у вікні керування та синхронізувати відображення гучності в інтерфейсі застосунку з фактичним станом системи.

Таким чином, комбінація авторизації через API, ефективного графічного інтерфейсу та фонових процесів моніторингу забезпечує стабільну, безпечну та зручну взаємодію користувача із застосунком.

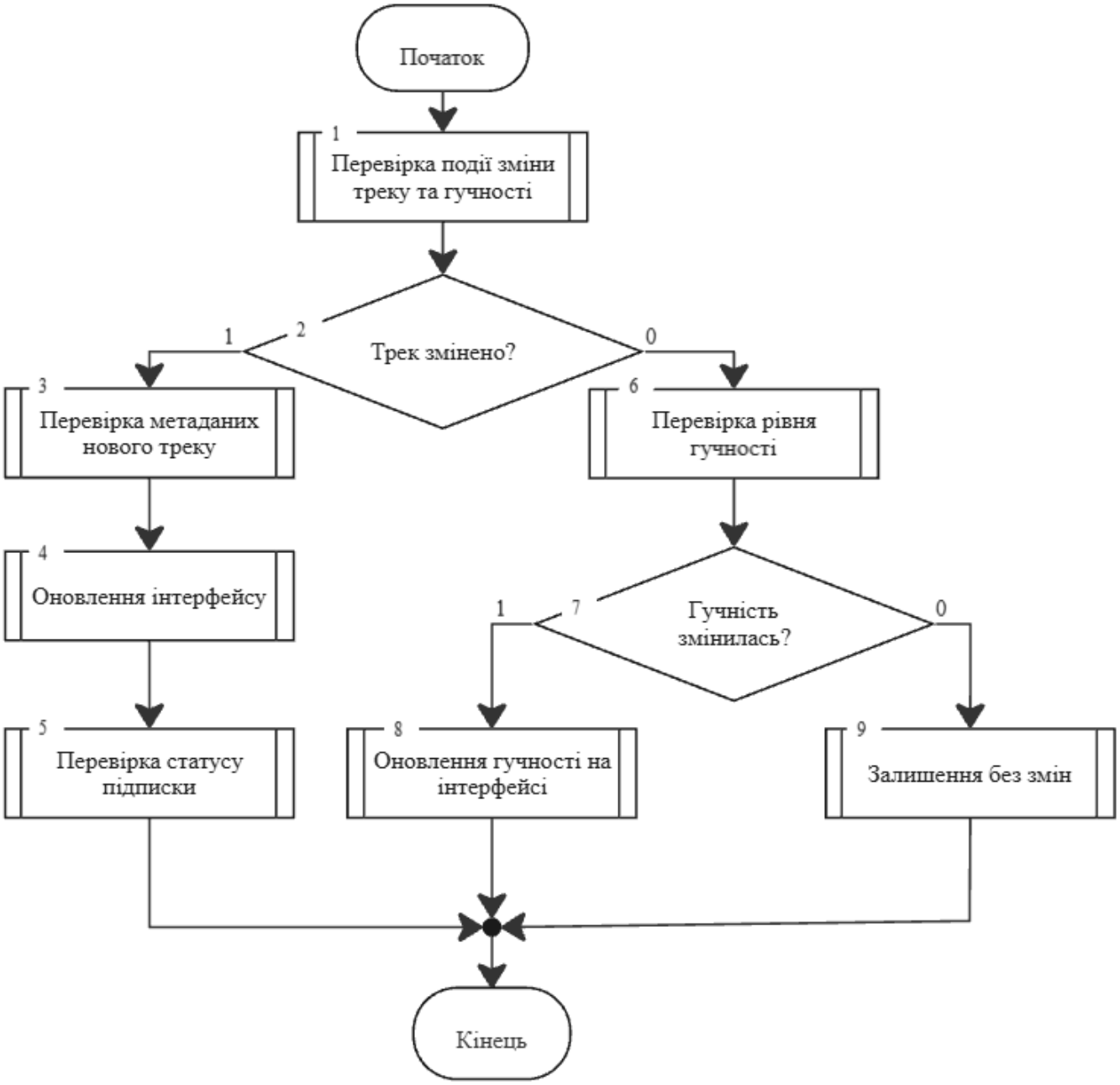

Рисунок 2.3 – Алгоритм управління аудіовідтворенням

Обробка подій здійснюється циклічно: кожна подія типу media_changed або volume_changed викликає відображення головного вікна, оновлення статусу підписки користувача (Premium/Free), а також рендеринг метаданих поточного треку – назви композиції, імені виконавця та обкладинки альбому – з автоматичним коригуванням розмірів інтерфейсу відповідно до нового вмісту.

Керування користувацькими діями реалізовано у вигляді окремих методів: запуск і призупинення відтворення, перемикання треків, зміна рівня гучності, відкриття додаткових вікон (перегляд альбомів, меню налаштувань) і перемикання в режим відтворення через YouTube.

Автоматичне приховування інтерфейсу здійснюється за відсутності активної взаємодії з користувачем протягом понад 5 секунд після втрати фокусу курсора. Такий підхід забезпечує мінімальне втручання у роботу інших програм і підвищує зручність використання.

Запропоновані алгоритми гарантують стабільне керування медіавідтворенням, адаптивне оновлення інтерфейсу та повну відповідність вимогам до ефективності й надійності сучасних програмних систем.

2.5 Розробка алгоритму асинхронного моніторингу медіа-стану та реактивного оновлення інтерфейсу

Алгоритм асинхронного моніторингу медіа-стану і реактивного оновлення інтерфейсу є центральним елементом інтелектуального медіа-застосунку, що забезпечує автоматичне виявлення змін у медіа-потоці та відповідне оновлення вікна застосунку без участі користувача. Архітектурна побудова алгоритму базується на паралельному запуску фонових потоків, використанні черги подій та подієво-орієнтованому механізмі оновлення інтерфейсу. Такий підхід дозволяє забезпечити мінімальний час реакції системи на зміну гучності або медіа-треку та оптимізує ресурси, задіяні в обробці подій користувацького середовища.

Ініціалізація алгоритму виконується в рамках функції monitor_volume_and_media, у якій здійснюється одночасний запуск двох

незалежних потоків: потоку моніторингу системної гучності та потоку моніторингу поточного відтворюваного треку. Крім того, паралельно з фоновими задачами активується періодична перевірка черги подій (метод process_queue) і періодичне оновлення прогресу відтворення треку (метод update_track_progress). Обидва моніторингові потоки працюють безперервно та виконують перевірки з заданою періодичністю в дискретні моменти часу.

Стан медіа-системи в момент часу t задається як вектор (формула 2.1):

$$S(t) = \{V(t), M(t)\} \tag{2.1}$$

де $V(t)$ – рівень системної гучності,

$M(t)$ – унікальний ідентифікатор поточного треку (наприклад, Spotify URI або хеш об’єкта треку).

Детекція зміни одного з параметрів відбувається на основі порівняння поточного значення з попереднім (формула 2.2, 2.3):

$$\Delta V = |V(t) - V(t-\Delta t)| > \varepsilon \tag{2.2}$$

$$\Delta M = M(t) \neq M(t-\Delta t) \tag{2.3}$$

де Δt – часовий інтервал дискретного опитування,

ε – граничне значення для виявлення значущої зміни гучності.

При реєстрації зміни одного з параметрів відповідний потік ініціює додавання повідомлення до глобальної черги подій. Зокрема, потік monitor_volume при виявленні $\Delta V > \varepsilon$ розміщує у черзі об’єкт "volume_changed", а потік monitor_media при виявленні ΔM додає "media_changed". Усі події обробляються у головному потоці за допомогою періодично викликуваного методу process_queue, який перевіряє вміст черги та виконує відповідні дії залежно від типу події.

Механізм обробки реалізується за допомогою дискретного відображення черги Q (формула 2.4):

$$Q = \{e1, e2, ..., en\}, \text{де } ei \in \{\text{"volume\_changed"}, \text{"media\_changed"}\} \quad (2.4)$$

Кожен елемент черги інтерпретується як тригер для функції оновлення інтерфейсу. Обробник подій аналізує тип події та викликає метод show_window у таких випадках: якщо подія дорівнює "volume_changed", або якщо подія дорівнює "media_changed" і не активовано suppress-режим. Suppress-режим може бути визначений у випадках, коли оновлення інтерфейсу є небажаним, наприклад, під час завантаження даних або відсутності мережі.

Функція show_window виконує формування та оновлення графічного інтерфейсу користувача на основі нового медіа-стану. Вона оновлює статус підписки (з використанням запиту до API медіасервісу), завантажує обкладинку треку, оновлює назву треку та виконавця, а також динамічно регулює розміри і положення вікна застосунку. Всі зміни виконуються на основі актуального значення вектора *S(t)*, яке формується під час обробки подій.

Візуальне оновлення інтерфейсу відбувається через реактивну функцію (формула 2.5):

$$UI(t) = f(M(t), U) \quad (2.5)$$

де *UI(t)* – поточний вигляд інтерфейсу,

M(t) – метадані активного треку,

U – статус підписки користувача.

При кожному оновленні інтерфейсу застосовується новий параметр *M(t)*, що забезпечує актуальність даних, представлених на екрані.

Загальний час реакції системи на зміну вхідного стану визначається як сума затримок (формула 2.6):

$$Treact = Tdetect + Tenqueue + Tprocess + Tui \quad (2.6)$$

де *Tdetect* – час виявлення зміни потоком,

Tenqueue – час постановки події в чергу,

Tprocess – час обробки події головним потоком,

Tui – час оновлення вікна.

За умови оптимальної реалізації всі компоненти цього виразу мають малу величину, а сумарний час *Treact* не перевищує 0,5–1,0 секунди, що забезпечує реальновчасне реагування системи на зміну медіа-контенту.

Алгоритм також володіє високою масштабованістю, оскільки дозволяє додати додаткові параметри моніторингу до вектору *S(t)*, наприклад (формула 2.7):

$$S(t) = \{V(t), M(t), B(t), N(t), ...\} \quad (2.7)$$

де *B(t)* – стан Bluetooth-пристрою,

N(t) – статус мережевого підключення.

Це створює передумови для побудови гнучких і адаптивних медіа-систем, здатних автономно реагувати на зміну зовнішнього середовища або дій користувача.

Застосування алгоритму дозволяє реалізувати принципи інтелектуального інтерфейсу, де реакція на зміну медіа-контексту відбувається без необхідності ручного втручання користувача. Результатом є покращена інтерактивність, скорочення часу реакції та підвищення зручності у взаємодії з медіа-застосунком.

2.4 Висновки

У результаті детального аналізу вхідних даних, необхідних для реалізації системи керування медіавідтворенням з інтеграцією платформи Spotify, було визначено кілька основних категорій інформації, які безпосередньо впливають на загальну функціональність, адаптивність та зручність використання розробленого

застосунку. Під час цього аналізу виявлено, що правильне оброблення вхідних даних є критично важливим для забезпечення ефективної взаємодії користувача з програмою та максимального задоволення його потреб.

Перше, що необхідно враховувати при розробці, це ідентифікація вхідних даних, які безпосередньо визначають основні функціональні можливості програми. Одним із важливих параметрів є інформація про користувача, зокрема тип його підписки на Spotify. Власники Premium-акаунтів мають доступ до розширених функцій, таких як прослуховування без реклами та можливість необмеженого пропуску треків, що значно підвищує комфорт використання програми. Це дозволяє створити безперебійний музичний досвід, що є важливим аспектом задоволення користувача. Крім того, система повинна мати можливість адаптувати інтерфейс до персоналізованих даних користувача, таких як аватар, ім’я та історія прослуховувань. Персоналізація є важливою складовою, оскільки дає можливість користувачу отримувати саме ті функції та можливості, які йому найбільше підходять. Це дозволяє значно покращити досвід взаємодії з програмою, створюючи індивідуалізоване середовище, яке відповідає уподобанням та потребам конкретного користувача.

Іншою важливою групою вхідних даних є метадані медіафайлів, які містять в собі інформацію про назву треку, ім’я виконавця, обкладинку альбому та інші ключові характеристики. Ці дані є необхідними для коректного відображення інформації в інтерфейсі користувача та синхронізації з платформою Spotify. Додатково, інтеграція з рекомендаційною системою Spotify відкриває перед користувачем нові можливості для відкриття музики, яка відповідає його вподобанням. Це значно підвищує цінність застосунку, оскільки система пропонує не тільки вже знайому музику, але й нові композиції, альбоми чи виконавців, які можуть сподобатися користувачеві на основі його попередніх виборів та переваг.

Інтерфейс застосунку має бути не лише функціональним, а й зручним для користувача. Одним із основних аспектів є логічне та інтуїтивно зрозуміле розміщення елементів управління, таких як кнопки «Пауза», «Наступний трек»,

«Попередній трек» та слайдер гучності. Це дозволяє користувачу швидко і без зайвих зусиль налаштовувати медіавідтворення, що покращує загальний досвід користування програмою. Також важливою є адаптивність інтерфейсу до різних типів пристроїв, зокрема до мобільних платформ. Програма повинна бути оптимізованою як для великих екранів комп'ютерів, так і для невеликих екранів смартфонів, щоб забезпечити однаково зручне користування незалежно від пристрою. Окрім цього, дизайн інтерфейсу має враховувати різні умови освітлення, що передбачає наявність темної та світлої теми. Це дозволяє користувачеві комфортно працювати з програмою як вдень, так і вночі.

Однією з ключових вимог до системи є інтеграція з популярними медіаплатформами, зокрема Spotify та YouTube. Це дозволяє не лише стабільно відтворювати актуальний контент, але й підтримувати динамічне оновлення даних, що робить користування програмою більш зручним і ефективним. Важливо також передбачити надійні механізми авторизації, зокрема використання протоколу OAuth 2.0, який гарантує захист особистих даних користувача під час доступу до його облікових записів на зовнішніх платформах. Це забезпечує високий рівень безпеки при взаємодії з платформами, що є важливою складовою довіри користувачів до застосунку.

Загалом, ефективність та стабільність функціонування системи безпосередньо залежить від злагодженої роботи всіх її компонентів. Узгоджена взаємодія між обробкою даних, інтерфейсом користувача та інтеграцією з зовнішніми сервісами створює основу для надійного, функціонального та зручного додатка, який здатний задовольнити потреби користувачів та забезпечити їм позитивний досвід взаємодії з медіаконтентом. Ефективна робота цієї системи залежить не лише від точного відображення та синхронізації даних, але й від гнучкості програми в умовах різноманітних пристроїв, підключень і уподобань користувачів. Всі ці фактори разом забезпечують стабільне функціонування програми, що відповідає вимогам сучасних користувачів та дозволяє створити найкращий досвід взаємодії з медіа.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ КЕРУВАННЯ МЕДІАВІДТВОРЕННЯМ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

При розробці програмного забезпечення для системи керування медіавідтворенням важливо правильно обрати технології для реалізації кожного модуля, враховуючи функціональні вимоги, продуктивність, масштабованість і здатність до інтеграції з існуючими платформами. Для забезпечення ефективної взаємодії з медіаплатформами, такими як Spotify та YouTube, технологічна база має включати інструменти для роботи з потоками даних, мультимедіа та динамічними інтерфейсами.

Для реалізації серверної частини було використано JavaScript у середовищі Node.js [21]. Node.js забезпечує високу продуктивність завдяки неблокуючій моделі введення/виведення, що робить його ідеальним для обробки одночасних запитів до зовнішніх сервісів. Для організації маршрутизації запитів та створення API було застосовано фреймворк Express.js, який дозволяє швидко налаштувати серверну логіку та обслуговувати запити до музичних платформ. Робота з API реалізується за допомогою бібліотек на зразок Axios, що забезпечують гнучке керування HTTP-запитами і зручне підключення до сторонніх сервісів[22].

Клієнтська частина застосунку реалізована з використанням JavaScript та фреймворка React. JavaScript дозволяє ефективно обробляти динамічні елементи інтерфейсу та забезпечує інтерактивність без перезавантаження сторінки [23]. React забезпечує побудову сучасного UI з високим рівнем продуктивності, завдяки використанню віртуального DOM і компонентного підходу. У поєднанні з Redux здійснюється централізоване управління станом програми, що спрощує масштабування та підтримання узгодженості між даними й візуальним представленням інтерфейсу [24].

Для взаємодії із зовнішніми музичними сервісами використовуються офіційні API Spotify та YouTube. Вибір цих API обумовлений їх детальною документацією, високою надійністю та широким функціоналом, що дозволяє реалізувати такі можливості, як пошук і відтворення треків, керування плейлистами та отримання інформації про виконавців. Spotify API дозволяє працювати з музичним каталогом, а також забезпечує доступ до метаданих і управління бібліотекою користувача. YouTube API дає змогу вбудовувати відео та реалізувати пошук та відтворення контенту безпосередньо з платформи.

Для відтворення мультимедійного контенту було використано бібліотеку Howler.js, яка підтримує різні формати аудіо та надає гнучкі можливості керування відтворенням — зупинка, перемотування, зміна гучності тощо. Для роботи з відео інтегровано бібліотеку Video.js, що дозволяє зручно відтворювати відеоконтент із таких джерел, як YouTube та Vimeo, у межах єдиного середовища медіаконтролера.

Інтерактивність і зручність інтерфейсу забезпечуються використанням бібліотеки Material-UI, яка надає набір адаптивних компонентів відповідно до стандартів Material Design. Це дозволяє створити сучасний, естетично привабливий та функціональний інтерфейс із підтримкою темної теми, адаптації під різні пристрої та швидкого реагування на дії користувача [25].

Сукупність обраних технологій дозволяє ефективно вирішувати завдання, поставлені перед програмою: забезпечення інтеграції з музичними платформами, якісне відтворення контенту, гнучке керування інтерфейсом і забезпечення позитивного користувацького досвіду. Node.js з Express для серверної логіки, React з Redux для клієнтської частини, Howler.js та Video.js для обробки медіа, а також офіційні API забезпечують стабільну, масштабовану та функціонально багату систему керування медіавідтворенням.

3.2 Розробка модуля керування відтворенням для Spotify

Модуль управління відтворенням у Spotify є ключовим компонентом при розробці програмного забезпечення для інтеграції музичних платформ у

мультимедійні системи. Його основна мета — забезпечити зручне та ефективне керування відтворенням аудіоконтенту шляхом використання Spotify Web API. Це API надає широкий набір функціональних можливостей, зокрема запуск, зупинку відтворення, перемикання треків та регулювання гучності, що дозволяє реалізувати повноцінний контроль над медіаплеєром платформи.

Ключовим етапом реалізації модуля є інтеграція з системою авторизації, що базується на протоколі OAuth 2.0. Завдяки цьому користувач має змогу надати застосунку доступ до свого облікового запису Spotify. Після проходження авторизації модуль отримує дозволи на керування відтворенням, доступ до особистих плейлистів та перегляд метаданих поточного треку, що забезпечує тісну інтеграцію з користувацькою бібліотекою.

Відтворення музики реалізується шляхом виклику відповідних методів API, зокрема start_playback для запуску і pause_playback для зупинки. Ці методи дають змогу керувати відтворенням обраних треків або плейлистів за допомогою унікальних ідентифікаторів URI, що забезпечує точне позиціонування контенту серед великого обсягу музичного каталогу.

Окрім базового відтворення, важливою функцією модуля є перемикання між композиціями. Це реалізується за допомогою методів skip_to_next та skip_to_previous, які дають змогу переміщатися між треками в межах активного альбому або списку відтворення. Додатково, через API можна отримати інформацію про поточний трек: назву, виконавця, обкладинку та інші метадані, що необхідні для відображення у графічному інтерфейсі користувача.

Ще однією суттєвою функцією є регулювання гучності — для цього використовується метод volume, що дозволяє змінювати рівень гучності в діапазоні від 0 до 100%. Це забезпечує користувачам можливість гнучкого налаштування звукового середовища відповідно до індивідуальних потреб.

У процесі реалізації було створено клас SpotifyPlayer, який інкапсулює всю логіку взаємодії з API Spotify (рис. 3.1). Цей клас містить методи для управління відтворенням, обробки даних про стан плеєра та отримання актуальної інформації

про відтворюваний контент. Однією з основних функцій класу є керування токенами доступу, які є необхідними для авторизованої взаємодії з платформою та постійно оновлюються відповідно до вимог протоколу OAuth 2.0.

У процесі розробки модуля особливу увагу було приділено обробці помилок та забезпеченню стабільної роботи системи. У разі втрати з'єднання з сервером або виникнення інших непередбачуваних ситуацій модуль здатен коректно реагувати, інформуючи користувача про проблему за допомогою відповідних повідомлень. Також реалізовано механізми автоматичного відновлення з'єднання з API без потреби у перезапуску застосунку, що сприяє безперервності використання програми.

Отже, модуль управління відтворенням для Spotify реалізує повний набір функціональних можливостей, необхідних для інтеграції з платформою Spotify. Він забезпечує ефективне керування музичним відтворенням, перемикання треків, зупинку, паузу та налаштування рівня гучності. Завдяки своїй гнучкості та масштабованості, модуль може бути адаптований для використання як у простих музичних плеєрах, так і в складних мультимедійних системах.

3.3 Розробка модулей керування відтворенням для YouTube та синхронізації

Під час розробки модуля керування відтворенням для YouTube основною метою було забезпечення можливості локального управління потоковим відеоконтентом за допомогою доступних інструментів API і механізмів контролю відтворення на стороні клієнта. Зважаючи на особливості платформи YouTube, де значна частина функціональності обмежена авторизацією або орієнтована виключно на роботу через вебінтерфейс, реалізація модуля потребувала пошуку альтернативних рішень для забезпечення локального контролю без повної інтеграції з офіційним API потокового управління [26].

Для реалізації цієї задачі було вирішено використати вбудований браузерний функціонал на основі бібліотеки pytube, що дозволяє отримувати прямі посилання

на відео з YouTube. У поєднанні з локальним медіаплеєром на базі бібліотеки vlc, це рішення дозволяє відтворювати відео за прямим посиланням безпосередньо у застосунку (рис. 3.2). Таким чином, архітектура модуля базується на двох ключових компонентах: механізмі обробки URL-адрес відео (рис. 3.3) та системі локального керування відтворенням через VLC-плеєр.

Реалізація модуля передбачає створення класу YouTubePlayer, який відповідає за пошук, підготовку та управління відтворенням контенту. Основні методи класу включають load_video(), що здійснює отримання прямого посилання на відео з YouTube за допомогою pytube, та play_video(), який передає це посилання VLC-плеєру для відтворення у локальному середовищі.

Особливу увагу було приділено обробці потоку даних. Після отримання об'єкта відео через бібліотеку pytube, проводиться вибір оптимального потоку за характеристиками якості та сумісності з відтворенням. Найчастіше вибирається потокове посилання на відео у форматі MP4 середньої роздільної здатності, що забезпечує гармонійний баланс між якістю відео та швидкістю завантаження.

Авторизація у даному випадку не є необхідною, оскільки відео доступне за загальнодоступними посиланнями, не захищеними механізмами аутентифікації. Це дозволяє значно знизити час затримки на старті відтворення та спростити архітектуру програмного модуля.

У рамках реалізації відображення додаткової інформації про відео (наприклад, назва, автор, мініатюра), розроблений модуль використовує функції pytube для отримання метаданих відео. Ці метадані зберігаються як атрибути класу YouTubePlayer, що дозволяє виводити їх на інтерфейс користувача паралельно з процесом відтворення.

Для забезпечення синхронізації між платформами Spotify та YouTube було реалізовано клас MediaController (рис. 3.4). Цей клас дозволяє здійснювати синхронізацію відтворення та гучності між двома платформами, що дає змогу зручно керувати мультимедійним контентом на обох платформах одночасно.

Цей клас дозволяє здійснювати одночасне керування відтворенням на Spotify та YouTube, синхронізуючи основні функції, такі як пауза, відтворення, перемикання треків та зміна гучності на обох платформах. Це забезпечує зручність у керуванні мультимедійним контентом без необхідності окремого контролю для кожної платформи.

3.4 Розробка модулю автоматичного виявлення та реакції на зміну медіа-стану

Мультимедійні застосунки нового покоління повинні не лише якісно відтворювати контент, а й забезпечувати інтуїтивну та адаптивну взаємодію з користувачем. Однією з важливих складових такої взаємодії є здатність програми автоматично реагувати на зміни у стані відтворення медіа, незалежно від джерела цих змін. Саме для цього у розробленому застосунку реалізовано спеціальний модуль асинхронного моніторингу та реактивного оновлення інтерфейсу, який дозволяє своєчасно відображати актуальний стан і підвищує комфорт користування.

Модуль автоматичного виявлення та реакції на зміну медіа-стану у цьому застосунку є прикладом сучасного підходу до побудови інтерактивних мультимедійних систем, де особлива увага приділяється не лише функціональності, а й зручності користувача. Його реалізація базується на принципах асинхронності та реактивності, що дозволяє забезпечити максимально швидку та коректну реакцію на будь-які зміни у стані відтворення медіа.

Завдяки використанню окремих потоків для моніторингу гучності та зміни треку, застосунок може постійно відстежувати ці параметри незалежно від основного графічного інтерфейсу. Це означає, що навіть якщо користувач взаємодіє з іншими програмами або змінює гучність через апаратні кнопки, застосунок миттєво реагує на ці події. Такий підхід дозволяє уникнути затримок та блокування інтерфейсу, що особливо важливо для мультимедійних програм, де швидкість реакції напряму впливає на користувацький досвід.

Важливою особливістю є також використання черги подій, яка виступає посередником між потоками моніторингу та головним потоком інтерфейсу. Це дозволяє безпечно передавати інформацію про зміни стану без ризику виникнення конфліктів або помилок, пов’язаних із багатопотоковістю. Головний потік періодично перевіряє цю чергу та виконує відповідні дії, такі як оновлення інформації про трек, обкладинку альбому, статус підписки або автоматичне відкриття вікна програми.

Ще одним важливим аспектом є можливість налаштування поведінки застосунку відповідно до індивідуальних вподобань користувача. Наприклад, користувач може вимкнути автоматичне відкриття вікна при зміні треку, якщо не бажає, щоб інтерфейс відволікав його під час прослуховування музики. Це реалізовано через перемикач у меню налаштувань, стан якого зберігається у конфігураційному файлі. Таким чином, навіть після перезапуску програми обрані налаштування залишаються актуальними.

Завдяки такій архітектурі модуль не лише виконує свої основні функції, а й створює основу для подальшого розвитку застосунку. У майбутньому можна додати нові типи подій, наприклад, автоматичне реагування на зміну підключених аудіопристроїв, інтеграцію з іншими музичними сервісами або впровадження інтелектуальних рекомендацій на основі аналізу історії прослуховування. Крім того, асинхронна модель дозволяє легко масштабувати застосунок, додаючи нові функції без ризику зниження продуктивності чи стабільності роботи.

Загалом, модуль автоматичного виявлення та реакції на зміну медіа-стану є не лише технічним рішенням для забезпечення базової функціональності, а й важливим елементом, що визначає якість взаємодії користувача із застосунком. Його гнучкість, надійність та можливість розширення роблять його невід’ємною частиною сучасних мультимедійних систем, орієнтованих на комфорт та індивідуальні потреби кожного користувача.

3.5 Розробка інтерфейсу програмного застосунку

Розробка інтерфейсу для програмного застосунку є критичним етапом у створенні продукту для управління медіавідтворенням. Інтерфейс повинен бути не лише естетично привабливим, а й інтуїтивно зрозумілим та зручним для користувачів, забезпечуючи безперешкодну взаємодію з усіма елементами програми.

Перше, на що слід звернути увагу при розробці інтерфейсу, це розуміння потреб користувачів і того, що вони очікують від застосунку. Важливо забезпечити легкий доступ до основних функцій, таких як відтворення музики, регулювання гучності, перемикання треків, додавання нових пісень у плейлисти. Інтерфейс повинен бути простим у використанні і не перевантажувати користувача зайвими елементами. Створення комфортної атмосфери для користувача, де все працює швидко та зручно, допоможе досягти успіху програми.

Інтеграція з платформою Spotify вимагає правильного балансу між функціональністю та простотою використання. На головному екрані має бути відображена інформація про поточний трек і додаткові функції, які допомагають користувачу без зусиль керувати плейлистами, переглядати улюблені пісні, не відволікаючись від основної мети – прослуховування музики.

Розташування елементів управління має бути продуманим, зручним і доступним. Наприклад, кнопки для відтворення, паузи та перемикання між треками повинні бути великими і легко доступними для натискання, особливо на мобільних пристроях. Адаптивний дизайн є важливим аспектом для мобільних пристроїв, де розмір елементів управління має бути достатнім для зручної взаємодії.

Регулювання гучності є ще однією важливою частиною інтерфейсу. Слайдер для гучності має бути точним і відображати поточний рівень звуку. Слайдер повинен бути інтерактивним, і його значення оновлюватиметься в реальному часі, що дозволяє користувачеві комфортно налаштувати звук.

Що стосується колірної палітри, то важливо створити атмосферу, яка буде комфортною для користувачів. Тема може бути темною для нічного використання

та світлою для денних умов, з можливістю перемикатися між цими режимами. Це дозволить забезпечити комфортний досвід використання на різних пристроях.

Простота інтерфейсу також є важливою. Перенавантаження екрана зайвими елементами ускладнює користування додатком. Мінімалістичний дизайн, що чітко розділяє елементи управління і відображення інформації про трек, дозволить уникнути безладу на екрані та зробить програму зручною та продуктивною.

Забезпечення інтеграції з іншими функціями, такими як пошук, фільтрація та керування плейлистами, також є важливою частиною. Це дозволяє користувачам швидко знаходити потрібну музику, організовувати свої треки і створювати персоналізовані списки відтворення, не витрачаючи час на складні маніпуляції.

У результаті успіх розробки інтерфейсу залежить від того, як добре поєднані всі елементи програми в єдину цілісну систему. Надійний і зручний інтерфейс забезпечить користувачам позитивний досвід, що дозволить додатку працювати довгий час без зайвих проблем.

3.6 Висновки

Отже, розробка програмного забезпечення для системи керування медіавідтворенням вимагає ретельного підбору технологічних інструментів та архітектури. Вибір Python з фреймворком Flask для серверної частини та JavaScript з фреймворком React для клієнтської частини виявився вдалим. Python є потужним інструментом для роботи з веб-сервісами та API, що дає можливість швидко та ефективно налаштовувати сервер, забезпечуючи стабільну взаємодію з зовнішніми платформами, такими як Spotify та YouTube. Flask дозволяє налаштовувати маршрути, обробляти запити та інтегрувати музичні сервіси через REST API, що дає змогу безперешкодно обробляти запити на відтворення медіаконтенту, пошук треків і керування плейлистами.

Для клієнтської частини було обрано JavaScript з фреймворком React, що дозволяє створювати динамічний, інтерактивний інтерфейс з мінімальними затримками під час взаємодії. React дозволяє оновлювати лише ті частини

інтерфейсу, що змінилися, що суттєво знижує навантаження на браузер і прискорює відображення контенту. Для управління станом додатку використано Redux, який дозволяє централізовано управляти даними і забезпечує консистентність між компонентами інтерфейсу.

Вибір API для Spotify та YouTube був здійснений з огляду на їхню функціональність та можливості інтеграції. Офіційне Spotify Web API надає доступ до величезного музичного каталогу і дозволяє користувачам виконувати операції з треками та плейлистами, такі як відтворення, пауза, перемикання та зміна гучності. За допомогою цього API можна забезпечити гнучке керування музичним контентом і отримувати актуальні дані про поточний трек, що є важливим для надання користувачам зручного інтерфейсу.

Модуль керування відтворенням для YouTube був розроблений з використанням бібліотек pytube і vlc, що дозволяє уникнути необхідності інтеграції з офіційним API YouTube для управління потоковим медіа. Завдяки pytube можна отримувати прямі посилання на відеоконтент з YouTube, що дозволяє обробляти і передавати ці відеофайли до локального VLC-плеєра. Такий підхід дозволяє обійти обмеження, пов'язані з авторизацією та зовнішнім доступом до API, знижуючи затримки на початку відтворення відео.

Інтеграція платформ Spotify і YouTube підвищує функціональність програми. Завдяки розробці класу MediaController вдалося забезпечити синхронізацію між відтворенням на Spotify та YouTube, що дозволяє користувачам контролювати як музичне, так і відео-відтворення в одному інтерфейсі. Це надає додаткову зручність, оскільки дає можливість одночасно управляти відтворенням медіаконтенту на обох платформах, синхронізуючи основні функції, такі як пауза, відтворення, перемикання треків і регулювання гучності.

Особлива увага була приділена обробці помилок та стабільності роботи системи. Усі модулі передбачають наявність механізмів для обробки збоїв, таких як втрата з’єднання з сервером або непередбачувані проблеми з потоками медіа. У

випадку таких збоїв система здатна відновити з’єднання без перезавантаження програми, що підвищує надійність та зручність для користувача.

Розробка програмного забезпечення продемонструвала ефективність обраного технологічного стеку. Python, Flask, React і Redux стали потужними інструментами для створення масштабованого і гнучкого додатку. Реалізація модулів для Spotify та YouTube забезпечує ефективне керування відтворенням медіаконтенту, а також інтеграцію між двома платформами, що гарантує безперебійну роботу програми. Це програмне забезпечення має великий потенціал для подальшого розширення функціональності та впровадження нових можливостей для керування медіаконтентом з інших платформ.

Таким чином розроблене програмне забезпечення відповідає всім функціональним вимогам, є стабільним, ефективним і готовим до подальшого масштабування.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування розробленої системи є важливою складовою етапу розробки програмного забезпечення. Його основною метою є виявлення помилок, дефектів і недоліків функціонування додатку, а також перевірка відповідності роботи системи початковим вимогам.

На початковому етапі тестування застосунок було встановлено на персональні комп'ютери, що працюють під управлінням операційних систем Windows 10 і Windows 11. Це дозволило перевірити стабільність роботи програми в актуальних операційних середовищах.

Перший етап тестування включав функціональну перевірку коректності роботи основних можливостей застосунку. Було перевірено, як система реагує на різні вхідні дані, а також як вона обробляє запити на зміну гучності, перемикання треків та зчитування інформації про поточне відтворення зі сторонніх платформ, таких як Spotify та YouTube. На рисунку 4.1 зображено приклад тестування функціональності перемикання треків через інтеграцію з Spotify.

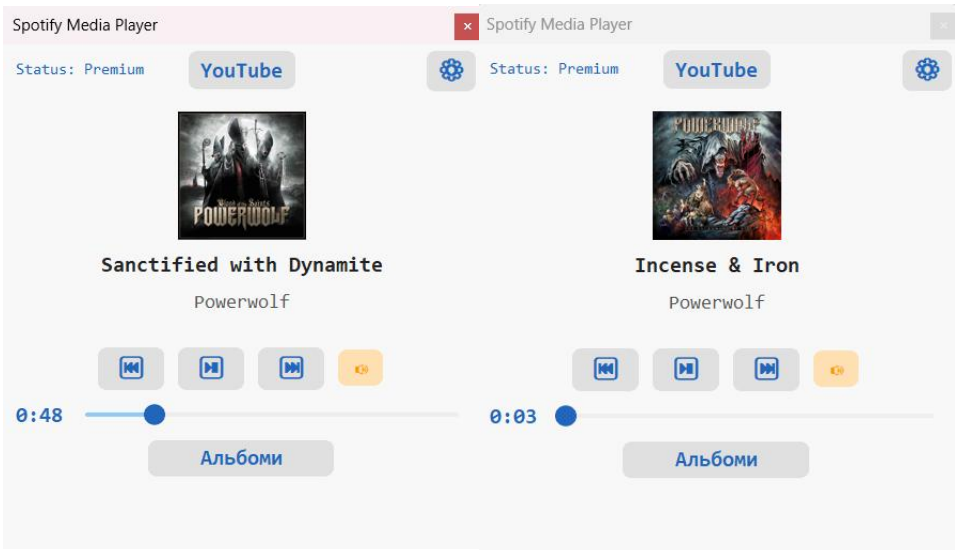

Рисунок 4.1 – Тестування перемикання треків

Також було проведено тестування на обробку некоректних даних: зокрема, система перевірялась у випадках, коли Spotify не був увімкнений. У таких ситуаціях застосунок коректно виводить повідомлення про помилку.

Наступним важливим етапом стало тестування відмовостійкості програми. Було змодельовано сценарії розриву з'єднання із зовнішніми сервісами, примусової зупинки процесу відтворення медіа або втрати інформації про поточне відтворення. Під час цих перевірок оцінювалася здатність системи відновити нормальну роботу без суттєвої втрати функціональності або з мінімальними незручностями для користувача.

Окрема увага приділялася перевірці продуктивності системи. Вимірювався час відгуку інтерфейсу на дії користувача — зокрема, натискання кнопок паузи, переходу до наступного чи попереднього треку, а також зміни гучності. За результатами тестування було встановлено, що середній час відповіді становить менше 1 секунди, що забезпечує комфортну взаємодію з інтерфейсом без відчутних затримок. На рисунку 4.2 наведено приклад миттєвої реакції інтерфейсу при зміні гучності.

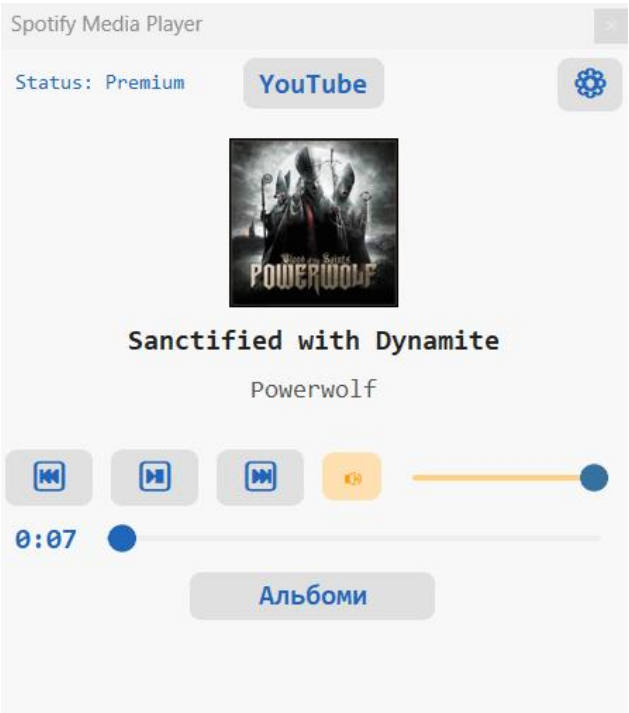

Рисунок 4.2 – Тестування зміни гучності

Також було проведено тестування роботи застосунку, для зміни треку та альбому, що програється, що зображено на рисунку 4.3.

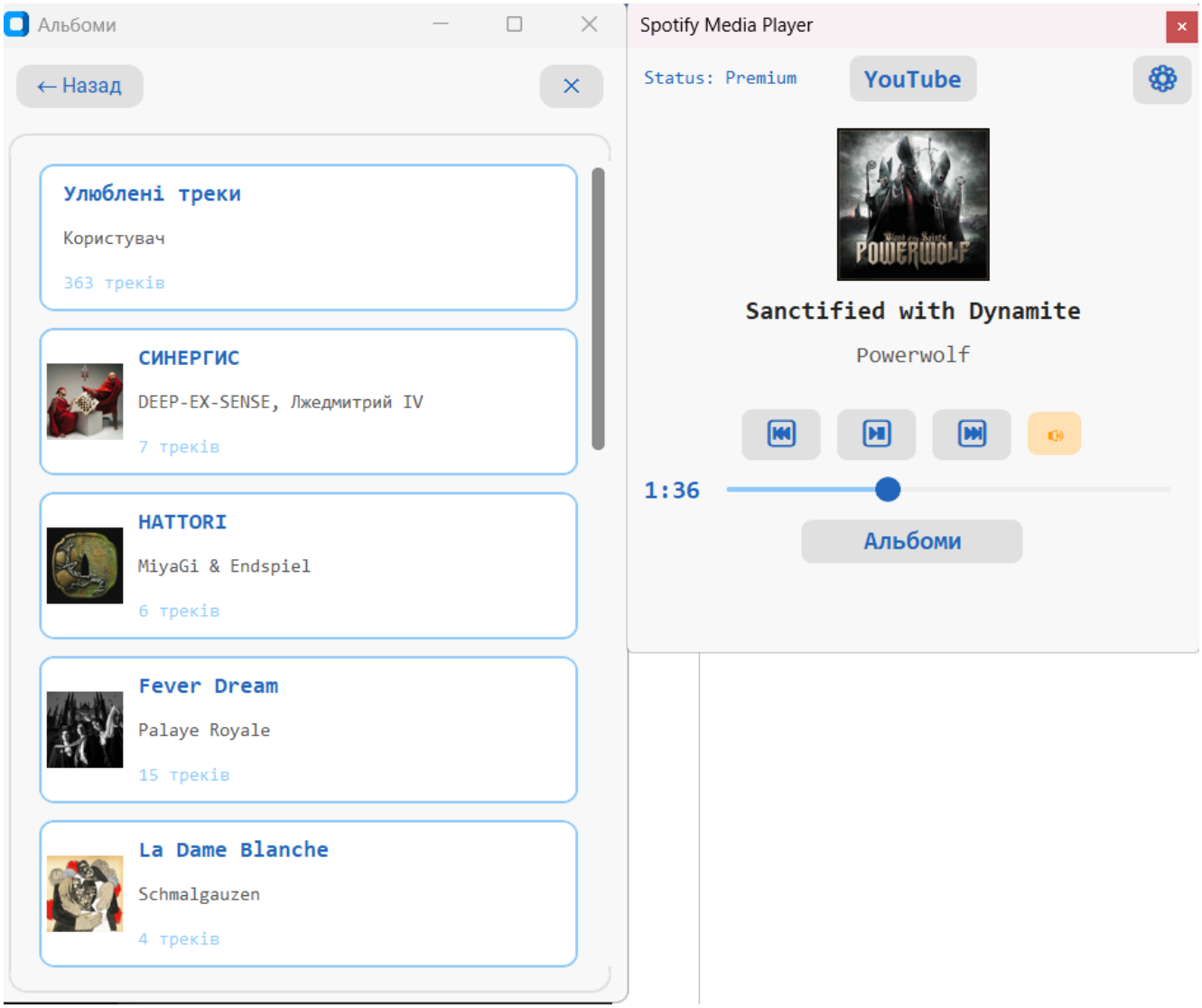

Рисунок 4.3 – Відображення альбомів та зміни треків через застосунок

Під час проведення тестування були перевірені різні сценарії підключення до платформ для відтворення музики. Зокрема, здійснювалася перевірка авторизації через Spotify API, взаємодії з відтворенням треків YouTube за допомогою локальних браузерних компонентів, а також обробки аудіофайлів. У разі успішного підключення система коректно зчитувала метадані треків, відображала обкладинки альбомів, назви композицій і імена виконавців.

Крім того, перевірялася функціональність вікна керування, яке з’являється у верхньому правому куті екрана поверх усіх вікон під час зміни треку або гучності. Результати тестування підтвердили, що незалежно від активного застосунку чи поточного режиму роботи операційної системи, це вікно зберігає видимість і дозволяє користувачеві ефективно керувати відтворенням. На рисунку 4.4 продемонстровано приклад функціонування цього вікна під час одночасного використання іншої програми.

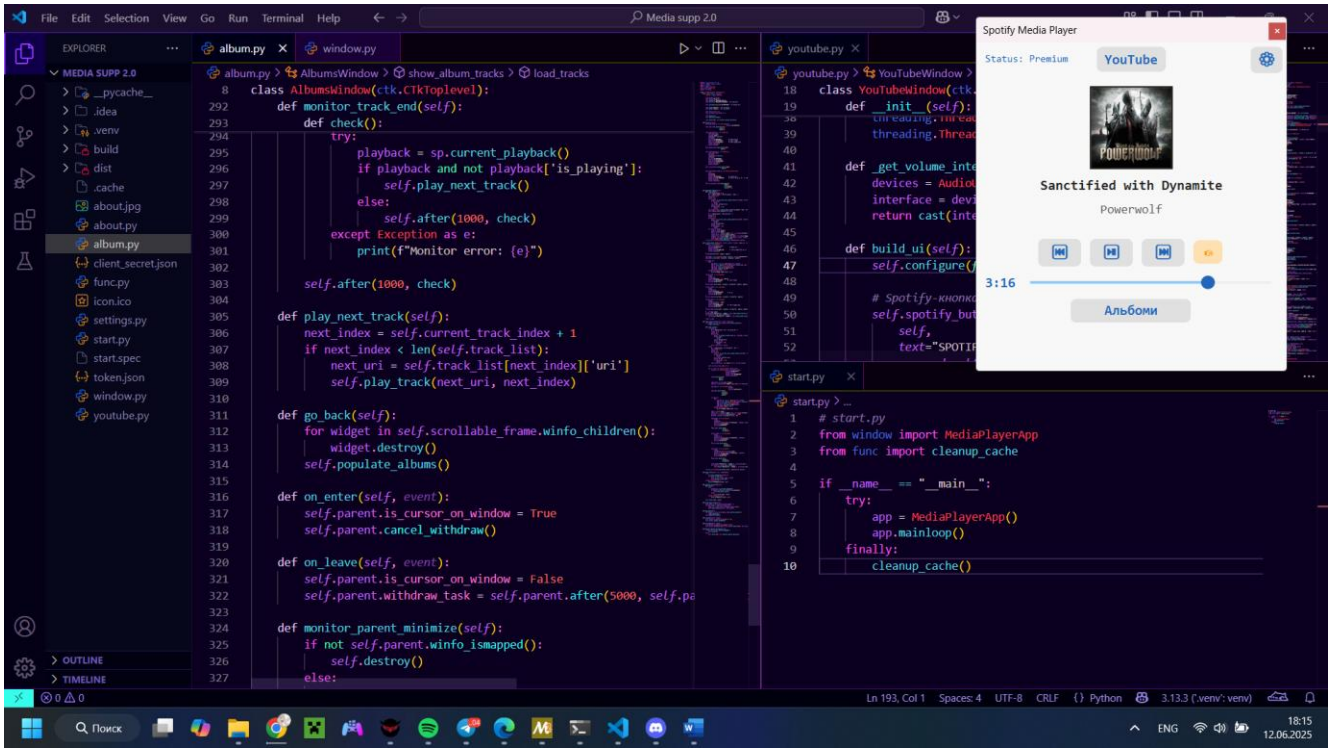

Рисунок 4.4 – Робота вікна при активному користуванні іншого

Результати проведеного тестування підтвердили, що функціональні можливості системи відповідають встановленим вимогам. Виявлені під час перевірки незначні помилки були оперативно виправлені, що дає підстави стверджувати про готовність розробленого застосунку до використання кінцевими користувачами.

4.2 Розробка інструкції користувача

Інструкція користувача включає опис технічних вимог для запуску створеного застосунку керування медіавідтворенням. У таблиці 4.1 наведено мінімальні та рекомендовані вимоги до апаратного забезпечення.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i5 або AMD Ryzen 5	Intel Core i3 або AMD Ryzen 3
Оперативна пам'ять	8 ГБ RAM	4 ГБ RAM
Вільний простір на диску	10 ГБ	2 ГБ
Відеокарта	Інтегрована графіка або краще	Інтегрована графіка
Операційна система	Windows 10 або Windows 11	Windows 7
Монітор	1920×1080 пікселів, 15" або більше	1280×720 пікселів, 13" або більше

Після інсталяції та запуску застосунку користувач потрапляє на головний екран програми. Першим кроком у роботі є авторизація через обрану музичну платформу, таку як Spotify або YouTube. У разі успішного проходження авторизації завантажується основний робочий інтерфейс.

На головному екрані користувач бачить обкладинку поточного треку, його назву та ім’я виконавця. Управління відтворенням здійснюється через кнопки «Пауза», «Наступний трек» та «Попередній трек». Для регулювання гучності передбачено повзунок або спеціальні кнопки. Зміни параметрів відтворення супроводжуються появою вікна керування, яке розміщується у правому верхньому куті екрана поверх усіх відкритих вікон (рис. 4.9).

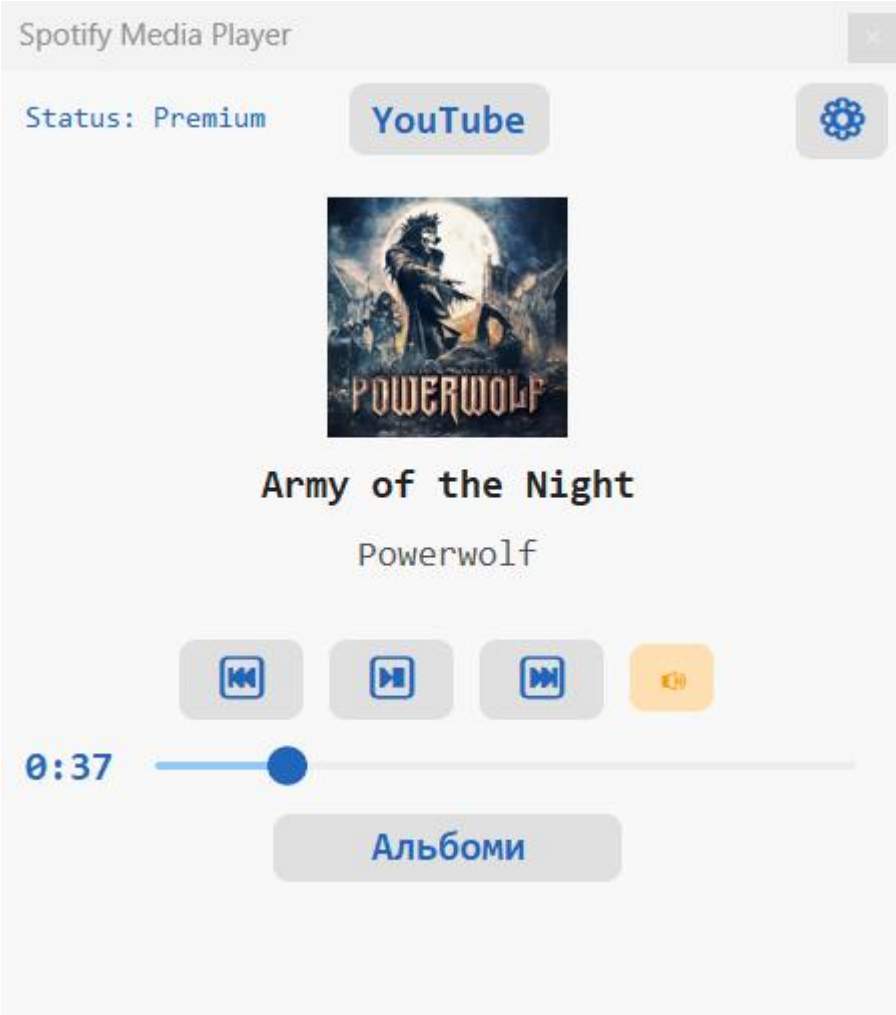

Рисунок 4.9 – Вікно керування відтворенням поверх робочого столу

У випадку виникнення помилок під час авторизації або втрати з'єднання з музичним сервісом застосунок автоматично відображає відповідне повідомлення з рекомендаціями щодо вирішення проблеми. Після відновлення з'єднання користувач має можливість продовжити керування медіавідтворенням без необхідності перезапуску програми.

4.3 Висновки

Отже, тестування створеного програмного забезпечення також включало перевірку роботи в умовах некоректних вхідних даних, зокрема відсутності з’єднання зі Spotify або іншими музичними платформами. Було встановлено, що система правильно реагує на подібні ситуації, відображаючи інформативні

повідомлення про помилки та пропонуючи можливі шляхи їх усунення. Наприклад, при відсутньому з’єднанні зі Spotify система повідомляє про потребу авторизації або підключення до мережі. Це підвищує зручність користування, навіть у непередбачуваних ситуаціях.

Окремо було протестовано роботу інтерфейсу в умовах багатозадачності, коли користувач одночасно працює з іншими програмами. Вікно керування медіавідтворенням, яке з’являється поверх усіх інших вікон, продемонструвало високу стабільність, зберігаючи функціональність навіть при активному використанні сторонніх застосунків. Це забезпечує додатковий комфорт, дозволяючи керувати відтворенням музики без переривання основної діяльності користувача.

До процесу тестування також входило вимірювання швидкості реакції інтерфейсу на дії користувача. Результати показали, що програма оперативно реагує на команди без відчутних затримок, забезпечуючи плавну взаємодію. Зокрема, високу швидкість показали операції зміни гучності, перемикання треків та інші основні функції, що є критично важливими для зручного керування.

Також було здійснено перевірку інтеграції з зовнішніми музичними сервісами. Усі тести на взаємодію з Spotify API, а також обробку відео- та аудіоконтенту з YouTube пройшли успішно. Застосунок коректно зчитував метадані треків, відображав обкладинки, назви композицій та імена виконавців, що дозволяло користувачеві легко орієнтуватися в музичному контенті та керувати його відтворенням.

Загалом, результати тестування дозволили визначити ключові переваги розробленої системи: стабільність, надійність і зручність у використанні. Незначні помилки, виявлені під час перевірок, були оперативно виправлені, що підтверджує високу готовність застосунку до використання в реальному середовищі. Отже, програмний продукт відповідає основним вимогам щодо якості, функціональності й ергономіки та може бути рекомендований для кінцевих користувачів.

ВИСНОВКИ

У межах виконання бакалаврської кваліфікаційної роботи було створено програмні модулі системи для керування медіавідтворенням. Розробка здійснювалася з використанням мови програмування Python, яка була обрана завдяки своїй гнучкості, популярності серед розробників, а також завдяки широкому вибору бібліотек для роботи з мультимедійними даними та графічними інтерфейсами. Для розробки клієнтської частини застосунку було використано середовище PyCharm, яке є ефективним інструментом для Python-розробки, забезпечуючи зручний інтерфейс і можливості для тестування коду.

У ході роботи було проведено поглиблений аналіз сучасного стану розвитку програмних засобів для керування медіавідтворенням. Були розглянуті існуючі рішення на ринку, зокрема застосунки, які дозволяють керувати музикою через такі платформи, як Spotify та YouTube. Вивчено переваги й недоліки кожного з інструментів, а також їх сумісність із різними операційними системами та пристроями. На основі цього аналізу було сформульовано основні функціональні вимоги до нового застосунку: зручне керування відтворенням, отримання метаданих з музичних сервісів, а також можливість налаштування гучності.

Під час реалізації бакалаврської роботи були досягнуті такі ключові результати:

– Сформульовано основні функціональні вимоги до системи відображення інформації про відтворення музики та керування цим процесом, включаючи можливість перемикання треків, паузи, регулювання гучності та відображення актуального музичного контенту.

– Розроблено ефективні алгоритми обробки подій зміни треків і регулювання гучності, що забезпечують швидку реакцію системи на дії користувача.

– Створено програмні модулі для інтеграції з сервісами Spotify та YouTube, які дозволяють отримувати метадані про музику — назви треків, імена виконавців, обкладинки альбомів та іншу релевантну інформацію.

– За допомогою бібліотеки CustomTkinter реалізовано графічний інтерфейс користувача, що забезпечує зручне й естетичне керування медіавідтворенням.

– Проведено тестування основних функцій програми відповідно до технічного завдання, яке підтвердило відповідність функціоналу заявленим вимогам.

У процесі розробки було також створено загальні алгоритмічні схеми системи за допомогою UML-діаграм. Це дало змогу наочно відобразити ключові процеси у системі керування відтворенням, зокрема логіку подій при зміні треків, оновлення інтерфейсу й керування рівнем гучності. Такі діаграми сприяли кращому узгодженню між компонентами програми та її оптимізації.

Результати тестування програмного забезпечення підтвердили його стабільність і відповідність усім визначеним вимогам. Застосунок коректно виконує усі необхідні функції: перемикання треків, відображення метаданих і взаємодію з зовнішніми музичними сервісами. Також було перевірено можливості взаємодії із сторонніми програмами та пристроями, що засвідчило надійність та функціональну ефективність системи. Для забезпечення зручності користування було підготовлено базову інструкцію користувача, яка допомагає швидко освоїти основні можливості та налаштування застосунку.

У підсумку, реалізований проєкт демонструє ефективність обраних технологій і алгоритмів, а також дозволяє створити зручний та стабільний інструмент для керування медіавідтворенням. Розроблене програмне забезпечення повністю відповідає заявленим вимогам, а результати тестування підтверджують його готовність до повноцінного використання в реальному середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Microsoft. Media Flyout Overview. Microsoft Docs, 2023. URL: https://learn.microsoft.com/en-us/windows/apps/design/shell/media-flyout (дата звернення: 10.06.2025)

2. ModernFlyouts – open-source media flyout replacement. GitHub, 2024. URL: https://github.com/ModernFlyouts-Community/ModernFlyouts (дата звернення: 10.06.2025)

3. E. Freeman, E. Robson. Head First Design Patterns: Building Extensible and Reactive Interfaces. O’Reilly Media, 2020.

4. Spotify API for music data analysis. Spotify for Developers, 2021. URL: https://developer.spotify.com (дата звернення: 10.06.2025)

5. YouTube Data API. Google Developer, 2021. URL: https://developers.google.com/youtube/v3 (дата звернення: 10.06.2025)

6. ReactiveX. Reactive Programming Overview. ReactiveX.io, 2023. URL: https://reactivex.io (дата звернення: 10.06.2025)

7. Щерба А.О., Карась О.В. - Розробка програмних модулів системи керування медіавідтворенням за допомогою API. Всеукраїнська науово-технічна конференція факультету інформаційних технологій та комп’ютерної інфенерії. – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/author

8. /submission/24267

9. Щерба А.О., Карась О.В. - розробка програмного модуля для збору та обробки статистики відтворення медіа з використанням Api. Молодь в науці: дослідження, проблеми, перспективи. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/24846

10. Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 3rd ed. O'Reilly Media, 2022. – 856 p.

11. Spotify for Developers. Web Playback SDK. – [Електронний ресурс]. – Режим доступу: https://developer.spotify.com/documentation/web-playback-sdk

12. Windows 10 Media Player Modern UI. – [Електронний ресурс]. – Режим доступу: https://github.com/Microsoft/Modern-Windows-Media-Player

13. Audio Controls for Windows. – [Електронний ресурс]. – Режим доступу: https://github.com/Sindresorhus/controls

14. Media Controls for Windows 10. – [Електронний ресурс]. – Режим доступу: https://github.com/WoC5/MediaControlSpotify Web API Documentation. Spotify for Developers. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.spotify.com/documentation/web-api (дата звернення: 03.03.2025)

15. FluentFlyout. unchihugo. GitHub Repository. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/unchihugo/FluentFlyout (дата звернення: 29.04.2025)

16. AudioFlyout. ADeltaX. GitHub Repository. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ADeltaX/AudioFlyout (дата звернення: 05.03.2025)

17. Hardt, D. OAuth 2.0 Simplified. Second Edition. Indie Author, 2022. – 170 p.

18. YouTube Data API Overview [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/youtube/v3 (дата звернення: 27.03.2025)

19. Python unittest – Unit testing framework. Python Documentation. [Електронний ресурс] – Режим доступу: https://docs.python.org/3/library/unittest.html (дата звернення: 29.04.2025)

20. Lutz, M. Learning Python. 5th Edition. O'Reilly Media, 2013.

21. Grinberg, M. Flask Web Development: Developing Web Applications with Python. O'Reilly Media, 2018.

22. Banks, A., Metz, S. Practical Object-Oriented Design in Ruby. Addison-Wesley, 2013.

23. Kettner, D. Learning Redux. Packt Publishing, 2017.

24. Downey, A. B. Think JavaScript: How to Think Like a Computer Scientist. Green Tea Press, 2016.

25. Mielke, M. Building JavaScript Applications: Learn to Build JavaScript Web Applications from Scratch. Packt Publishing, 2016.

26. Material UI. Material-UI: React components for faster and easier web development. [Електронний ресурс] – Режим доступу до ресурсу: https://mui.com/ (дата звернення: 10.04.2025)

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
" 25 " березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка програмного забезпечення для керування медіавідтворенням на персональному комп'ютері» за спеціальністю –121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
д-р. філос., асист каф. ПЗ О.В. Карась
" 24 " березня 2025 р.

Виконав:
студент гр.2ПІ-21б А.О. Щерба
" 24 " 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмного забезпечення для керування медіавідтворенням на персональному комп’ютері».

Галузь застосування – медіа.

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ № 97 від «20» березня 2025 р. ректора ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки

Мета та призначення розробки полягає у створенні компактного застосунку для зручного керування відтворенням музики з платформ Spotify та YouTube, що дозволяє переглядати трек, обкладинку та керувати відтворенням без перемикання між вікнами.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Щерба А.О., Карась О.В. - Розробка програмних модулів системи керування медіавідтворенням за допомогою API. Всеукраїнська науово-технічна конференція факультету інформаційних технологій та комп’ютерної інфенерії. – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/author/submission/24267

2. Щерба А.О., Карась О.В. - розробка програмного модуля для збору та обробки статистики відтворення медіа з використанням Api. Молодь в науці: дослідження, проблеми, перспективи. Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/24846

5. Технічні вимоги

Вихідні дані до роботи: середовище програмної розробки – Sublime Text 3; мова програмування – Python; фреймворки – Node; допоміжні технології – JS; персональний ПК з ОС Windows 10 - 11; медіавідтворення.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до БКР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ

9. Стадії та етапи розробки:

№ п\п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз цифрових платформ для керування медіавідтворенням на ПК	25.03.2025-30.03.2025
2	Дослідження методів персоналізованих рекомендацій музичного контенту	31.03.2025-12.04.2025
3	Побудова моделей функціонування системи оренди та обміну музичними файлами	13.04.2025-30.04.2025
4	Розробка модулів авторизації, пошуку та рекомендацій з використанням Spotify API	01.05.2025-08.05.2025
5	Створення інтерфейсу користувача застосунку для медіавідтворення	09.05.2025-16.05.2025
6	Тестування програмного забезпечення та перевірка функціональності	17.05.2025-20.05.2025
7	Оформлення матеріалів до захисту БКР	21.05.2025-30.05.2025

10. Порядок контролю та прийняття

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

Додаток Б – Протокол перевірки бакалаврської кваліфікаційної роботи на наявність текстових запозичень

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Розробка програмного забезпечення для керування медіавідтворенням на ПК

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ ПЗ, ІТКІ, 2ПІ-21б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 6,5 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Карась О.В. д-р. філос., асист каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Щерба А. О.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

window.py

```
import queue
import ctypes
from func import sp
from album import AlbumsWindow
from settings import load_settings, save_settings
import customtkinter as ctk
import threading
import time
import requests
from PIL import Image, ImageTk
from io import BytesIO
from ctypes import POINTER, cast
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume



ctk.set_appearance_mode("dark")
ctk.set_default_color_theme("blue")

class MediaPlayerApp(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("Spotify Media Player")
        self.geometry("360x180+1200+20")
        self.attributes("-topmost", True)
        self.withdraw()
        self.configure(fg_color="black")

        self.queue = queue.Queue()
        self.is_premium = self.has_premium()
        self.previous_track = None
        self.track_duration = 0
        self.track_position = 0
        self.is_cursor_on_window = False
        self.withdraw_task = None
        self.is_muted = False
        self.last_volume = 50

        self.build_ui()
        self.monitor_volume_and_media()

        self.bind("<Enter>", self.on_cursor_enter)
        self.bind("<Leave>", self.on_cursor_leave)
```

```
    self.set_window_flags()
    self.settings = load_settings()

def set_window_flags(self):
    self.update_idletasks()
    hwnd = ctypes.windll.user32.GetParent(self.winfo_id())
    extended_style = 0x00000008 | 0x00000080 | 0x08000000
    ctypes.windll.user32.SetWindowLongW(hwnd, -20, extended_style)
    ctypes.windll.user32.SetWindowPos(hwnd, 0, 0, 0, 0,
                                      0x0010 | 0x0002 | 0x0001 | 0x0040)

def on_cursor_enter(self, event):
    self.is_cursor_on_window = True
    self.cancel_withdraw()

def on_cursor_leave(self, event):
    self.is_cursor_on_window = False
    self.withdraw_task = self.after(5000, self.withdraw_if_cursor_left)
def cancel_withdraw(self):
    if self.withdraw_task is not None:
        self.after_cancel(self.withdraw_task)
        self.withdraw_task = None

def withdraw_if_cursor_left(self):
    if not self.is_cursor_on_window:
        self.withdraw()

def build_ui(self):
    self.youtube_button = ctk.CTkButton(
        self, text="YouTube",
        fg_color="red",
        text_color="white",
        hover_color="#cc0000",
        command=self.switch_to_youtube_mode
    )
    self.youtube_button.place(relx=0.5,
                              rely=0.0,
                              anchor="n",
                              y=5)

    self.subscription_label = ctk.CTkLabel(
        self, text="Status",
        font=ctk.CTkFont(size=12),
        text_color="white"
    )
    self.subscription_label.pack(pady=(10, 5),
                                 anchor="w",
                                 padx=10)

    self.album_cover_label = ctk.CTkLabel(self,
```

```
                                text="")
self.album_cover_label.pack(pady=(0, 10))

self.track_label = ctk.CTkLabel(
    self,
    text="",
    font=ctk.CTkFont(size=16,
                     weight="bold"),
    wraplength=280,
    justify="center",
    text_color="white"
)
self.track_label.pack(pady=(0, 3),
                      padx=10)

self.artist_label = ctk.CTkLabel(
    self, text="",
    font=ctk.CTkFont(size=14),
    wraplength=280,
    justify="center",
    text_color="white"
)
self.artist_label.pack(pady=(0, 10),
                       padx=10)

self.controls_frame = ctk.CTkFrame(self,
                                   fg_color="black")
self.controls_frame.pack(pady=(0, 5))

self.prev_btn = ctk.CTkButton(
    self.controls_frame,
    text="⏮",
    width=60,
    fg_color="green",
    hover_color="#00cc00",
    text_color="black",
    command=self.previous_track_action
)
self.prev_btn.grid(row=0,
                   column=0,
                   padx=5)

self.play_pause_btn = ctk.CTkButton(
    self.controls_frame,
    text="⏯",
    width=60,
    fg_color="green",
    hover_color="#00cc00",
    text_color="black",
    command=self.toggle_play_pause
```

```
)
self.play_pause_btn.grid(row=0,
                         column=1,
                         padx=5)

self.next_btn = ctk.CTkButton(
    self.controls_frame,
    text="⏭",
    width=60,
    fg_color="green",
    hover_color="#00cc00",
    text_color="black",
    command=self.next_track_action
)
self.next_btn.grid(row=0,
                   column=2,
                   padx=5)

self.volume_menu_btn = ctk.CTkButton(
    self.controls_frame,
    text="🔊",
    width=40,
    fg_color="green",
    hover_color="#00cc00",
    text_color="black",
    command=self.toggle_volume_menu
)
self.volume_menu_btn.grid(row=0,
                          column=3,
                          padx=(5, 0))
self.volume_menu_btn.bind("<Button-3>", self.toggle_mute)

self.volume_slider = ctk.CTkSlider(
    self.controls_frame,
    from_=0,
    to=100,
    width=100,
    button_color="red",
    progress_color="red",
    fg_color="gray",
    command=self.set_volume
)
self.volume_slider.grid(row=0,
                        column=4,
                        padx=(5, 10))
try:
    playback = sp.current_playback()
    if playback and playback.get('device'):
        self.volume_slider.set(playback['device']['volume_percent'])
    else:
```

```
        self.volume_slider.set(50)  # стандартне значення
except Exception as e:
    print(f"Error setting initial volume: {e}")
    self.volume_slider.set(50)

self.volume_slider.grid_remove()

self.time_frame = ctk.CTkFrame(self,
                               fg_color="black")
self.time_frame.pack(pady=(5, 5),
                     padx=10,
                     fill="x")

self.current_time_label = ctk.CTkLabel(
    self.time_frame,
    text="0:00",
    text_color="white"
)
self.current_time_label.pack(side="left")

self.progress_slider = ctk.CTkSlider(
    self.time_frame,
    from_=0,
    to=100,
    button_color="green",
    progress_color="green",
    fg_color="gray",
    command=self.seek_position
)
self.progress_slider.pack(side="left",
                          fill="x",
                          expand=True,
                          padx=(10, 0))

self.albums_button = ctk.CTkButton(
    self,
    text="Альбоми",
    fg_color="green",
    hover_color="#00cc00",
    text_color="black",
    command=self.open_albums_window
)
self.albums_button.pack(pady=(0, 10))

self.settings_btn = ctk.CTkButton(
    self, text="⚙",
    width=30,
    height=30,
    fg_color="green",
    hover_color="#00cc00",
```

```
        text_color="black",
        command=self.toggle_settings_menu
    )
    self.settings_btn.place(relx=1.0,
                            rely=0.0,
                            anchor="ne",
                            x=-5,
                            y=5)

def toggle_volume_menu(self):
    if self.volume_slider.winfo_viewable():
        self.volume_slider.grid_remove()
    else:
        self.volume_slider.grid()

def toggle_mute(self, event):
    try:
        if not self.is_muted:
            self.last_volume = self.volume_slider.get()
            sp.volume(0)
            self.volume_slider.set(0)
            self.is_muted = True
        else:
            sp.volume(int(self.last_volume))
            self.volume_slider.set(self.last_volume)
            self.is_muted = False
    except Exception as e:
        print(f"Error toggling mute: {e}")

def has_premium(self):
    try:
        user_info = sp.current_user()
        if 'product' in user_info and user_info['product'] == 'premium':
            return True
        else:
            return False
    except Exception as e:
        print(f"Error checking premium status: {e}")
        return False

def get_current_track(self):
    try:
        current_playback = sp.current_playback()
        if current_playback and current_playback.get('item'):
            track = current_playback['item']
            name = track['name']
            artist = ', '.join(a['name'] for a in track['artists'])
            cover_url = track['album']['images'][0]['url']
            self.track_duration = track['duration_ms'] // 1000
            self.track_position = current_playback['progress_ms'] // 1000
```

```
            return name, artist, cover_url
        return None
    except Exception as e:
        print(f"Error fetching current track: {e}")
        return None

def monitor_volume_and_media(self):
    threading.Thread(target=self.monitor_volume,
                     daemon=True).start()
    threading.Thread(target=self.monitor_media,
                     daemon=True).start()
    self.after(100, self.process_queue)
    self.after(1000, self.update_track_progress)

def monitor_volume(self):
    devices = AudioUtilities.GetSpeakers()
    interface = devices.Activate(IAudioEndpointVolume._iid_, 0, None)
    volume = interface.QueryInterface(IAudioEndpointVolume)
    current_volume = volume.GetMasterVolumeLevelScalar()

    while True:
        new_volume = volume.GetMasterVolumeLevelScalar()
        if new_volume != current_volume:
            current_volume = new_volume
            self.queue.put("volume_changed")
        time.sleep(1)

def monitor_media(self):
    while True:
        track_data = self.get_current_track()
        if track_data:
            track_name, _, _ = track_data
            if track_name != self.previous_track:
                self.previous_track = track_name
                self.queue.put("media_changed")
        time.sleep(1)

def process_queue(self):
    while not self.queue.empty():
        msg = self.queue.get()
        if msg in ["volume_changed", "media_changed"]:
            self.show_window()
    self.after(100, self.process_queue)

def update_track_progress(self):
    try:
        playback = sp.current_playback()
        if playback and playback['is_playing']:
            self.track_position += 1
            minutes = self.track_position // 60
```

```
            seconds = self.track_position % 60
            self.current_time_label.configure(text=f"{minutes}:{seconds:02d}")
            if self.track_duration:
                percent = (self.track_position / self.track_duration) * 100
                self.progress_slider.set(percent)
        except:
            pass
        self.after(1000, self.update_track_progress)

    def seek_position(self, value):
        if self.track_duration:
            new_position = int((value / 100) * self.track_duration) * 1000
            try:
                sp.seek_track(position_ms=new_position)
                self.track_position = new_position // 1000
            except Exception as e:
                print(f"Error seeking position: {e}")

    def show_window(self):
        if not self.is_cursor_on_window:
            self.deiconify()
            self.after(5000, self.withdraw_if_cursor_left)

        try:
            self.is_premium = self.has_premium()
            self.subscription_label.configure(text=f"Status: {'Premium' if self.is_premium else 'Free'}")
        except Exception as e:
            print(f"Error updating subscription status: {e}")
            self.subscription_label.configure(text="Status: Free")

        track_data = self.get_current_track()
        if track_data:
            name, artist, cover_url = track_data

            self.track_label.configure(text=name)
            self.artist_label.configure(text=artist)

            img_data = requests.get(cover_url).content
            img = Image.open(BytesIO(img_data)).resize((120, 120))
            img_tk = ImageTk.PhotoImage(img)

            self.album_cover_label.configure(image=img_tk,
                                             text="")
            self.album_cover_label.image = img_tk

            self.update_idletasks()
            new_height = self.track_label.winfo_height() + self.artist_label.winfo_height() + 310
            self.geometry(f"360x{new_height}+1400+20")

    def simulate_keypress(self, key_code):
```

```
    VK = key_code
    ctypes.windll.user32.keybd_event(VK, 0, 0x0001, 0)
    ctypes.windll.user32.keybd_event(VK, 0, 0x0001 | 0x0002, 0)

def previous_track_action(self):
    if self.is_premium:
        sp.previous_track()
    else:
        self.simulate_keypress(0xB1)

def toggle_play_pause(self):
    if self.is_premium:
        playback = sp.current_playback()
        if playback and playback['is_playing']:
            sp.pause_playback()
        else:
            sp.start_playback()
    else:
        self.simulate_keypress(0xB3)

def next_track_action(self):
    if self.is_premium:
        sp.next_track()
    else:
        self.simulate_keypress(0xB0)

def set_volume(self, value):
    try:
        sp.volume(int(value))
        self.last_volume = value
        self.is_muted = False
    except Exception as e:
        print(f"Error setting volume: {e}")

def open_albums_window(self):
    if not hasattr(self, "albums_window") or not self.albums_window.winfo_exists():
        self.albums_window = AlbumsWindow(self)

def toggle_settings_menu(self):
    if hasattr(self, "settings_frame") and self.settings_frame.winfo_exists():
        self.settings_frame.destroy()
    else:
        self.show_settings_menu()

def show_settings_menu(self):
    self.settings_frame = ctk.CTkFrame(self,
                                       width=200,
                                       height=60,
                                       corner_radius=8)
    self.settings_frame.place(relx=1.0,
```

```
                    rely=0.0,
                    anchor="ne",
                    x=-45,
                    y=45)

    self.suppress_switch = ctk.CTkSwitch(
        self.settings_frame,
        text="Не відкривати при зміні треку",
        command=self.toggle_suppress_mode
    )



    self.suppress_switch.pack(padx=10,
                    pady=10)

    about_btn = ctk.CTkButton(
        self.settings_frame,
        text="Про програму",
        fg_color="green",
        command=self.open_about_window
    )
    about_btn.pack(padx=10,
                pady=(0, 10))

    if self.settings.get("suppress_on_track_change"):
        self.suppress_switch.select()
    else:
        self.suppress_switch.deselect()

def open_about_window(self):
    import about
    about.AboutWindow(self)

def toggle_suppress_mode(self):
    self.settings["suppress_on_track_change"] = self.suppress_switch.get()
    save_settings(self.settings)

def process_queue(self):
    while not self.queue.empty():
        msg = self.queue.get()
        if msg == "volume_changed":
            self.show_window()
        elif msg == "media_changed" and not self.settings.get("suppress_on_track_change", False):
            self.show_window()
    self.after(100, self.process_queue)

def switch_to_youtube_mode(self):
    self.destroy()  # повністю закриваємо Spotify вікно
    import youtube
```

```
        youtube.YouTubeWindow().mainloop()

    def release_resources(self):
        try:
            devices = AudioUtilities.GetSpeakers()
            interface = devices.Activate(
                IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
            volume = cast(interface, POINTER(IAudioEndpointVolume))
            volume.Release()  # Звільнення ресурсу вручну
        except Exception as e:
            print(f"Помилка при звільненні ресурсів: {e}")
```

album.py

```
import customtkinter as ctk
from PIL import Image, ImageTk
import requests
from io import BytesIO
from window import sp

class AlbumsWindow(ctk.CTkToplevel):
    def __init__(self, parent):
        super().__init__(parent)

        self.parent = parent
        self.title("Альбоми")
        self.geometry("400x600+900+20")
        self.protocol("WM_DELETE_WINDOW", self.destroy)

        self.configure(fg_color="black")  # чорний фон

        self.bind("<Enter>", self.on_enter)
        self.bind("<Leave>", self.on_leave)

        self.track_list = []
        self.current_track_index = -1

        self.build_ui()
        self.populate_albums()

        self.after(500, self.monitor_parent_minimize)

    def build_ui(self):
        self.top_frame = ctk.CTkFrame(self,
                                      fg_color="black")
```

```
        self.top_frame.pack(fill="x",
                            padx=10,
                            pady=10)

        self.back_button = ctk.CTkButton(
            self.top_frame,
            text="← Назад",
            command=self.go_back,
            width=80,
            fg_color="green",
            text_color="black",
            hover_color="#006400",
            corner_radius=10
        )
        self.back_button.pack(side="left",
                              padx=5)

        self.close_button = ctk.CTkButton(
            self.top_frame,
            text="✕",
            command=self.destroy,
            width=40,
            fg_color="green",
            text_color="black",
            hover_color="#006400",
            corner_radius=10
        )
        self.close_button.pack(side="right",
                               padx=5)

        self.scrollable_frame = ctk.CTkScrollableFrame(
            self,
            width=380,
            height=500,
            fg_color="black"
        )
        self.scrollable_frame.pack(padx=10,
                                   pady=5,
                                   fill="both",
                                   expand=True)

    def populate_albums(self):
        self.album_frames = []

        fav_frame = self.create_album_frame("Улюблені треки", "Користувач", 50, None)
        self.album_frames.append(fav_frame)

        albums = sp.current_user_saved_albums(limit=10)['items']
        for item in albums:
            album = item['album']
```

```
        name = album['name']
        artist = ", ".join(a['name'] for a in album['artists'])
        track_count = album['total_tracks']
        image_url = album['images'][0]['url'] if album['images'] else None
        frame = self.create_album_frame(name, artist, track_count, image_url, album['id'])
        self.album_frames.append(frame)

def create_album_frame(self, title, artist, tracks, image_url, album_id=None):
    frame = ctk.CTkFrame(self.scrollable_frame,
                         fg_color="black",
                         corner_radius=10)
    frame.pack(fill="x",
               pady=5,
               padx=5)

    img_label = ctk.CTkLabel(frame,
                             text="",
                             fg_color="black")
    img_label.grid(row=0,
                   column=0,
                   rowspan=3,
                   padx=5)

    if image_url:
        img_data = requests.get(image_url).content
        img = Image.open(BytesIO(img_data)).resize((60, 60))
        img_tk = ImageTk.PhotoImage(img)
        img_label.configure(image=img_tk)
        img_label.image = img_tk

    title_label = ctk.CTkLabel(
        frame,
        text=title,
        font=ctk.CTkFont(size=14,
                         weight="bold"),
        text_color="white",
        fg_color="black"
    )
    title_label.grid(row=0,
                     column=1,
                     sticky="w")

    artist_label = ctk.CTkLabel(
        frame,
        text=artist,
        font=ctk.CTkFont(size=12),
        text_color="white",
        fg_color="black"
    )
    artist_label.grid(row=1,
```

```
                column=1,
                sticky="w")

    tracks_label = ctk.CTkLabel(
        frame,
        text=f"{tracks} треків",
        font=ctk.CTkFont(size=11),
        text_color="white",
        fg_color="black"
    )
    tracks_label.grid(row=2,
                column=1,
                sticky="w")

    frame.bind("<Button-1>", lambda e: self.show_album_tracks(title, album_id))
    for child in frame.winfo_children():
        child.bind("<Button-1>", lambda e: self.show_album_tracks(title, album_id))

    return frame

def show_album_tracks(self, title, album_id):
    for widget in self.scrollable_frame.winfo_children():
        widget.destroy()

    if album_id:
        self.track_list = sp.album_tracks(album_id)['items']
    else:
        items = sp.current_user_saved_tracks(limit=20)['items']
        self.track_list = [t['track'] for t in items]

    self.current_track_index = -1

    for i, track in enumerate(self.track_list):
        frame = ctk.CTkFrame(self.scrollable_frame,
                        fg_color="black",
                        corner_radius=10)
        frame.pack(fill="x",
                pady=5,
                padx=5)

        album_data = track.get('album')
        img_url = album_data['images'][0]['url'] if album_data and album_data.get('images') else None

        img_label = ctk.CTkLabel(frame,
                            text="",
                            fg_color="black")
        img_label.grid(row=0,
                    column=0,
                    rowspan=3,
                    padx=5)
```

```
if img_url:
    img_data = requests.get(img_url).content
    img = Image.open(BytesIO(img_data)).resize((60, 60))
    img_tk = ImageTk.PhotoImage(img)
    img_label.configure(image=img_tk)
    img_label.image = img_tk

name = track['name']
artist = ", ".join(a['name'] for a in track['artists'])
duration = int(track['duration_ms'] / 1000)
minutes, seconds = divmod(duration, 60)

name_label = ctk.CTkLabel(
    frame,
    text=name,
    font=ctk.CTkFont(size=13,
                     weight="bold"),
    text_color="white",
    fg_color="black"
)
name_label.grid(row=0,
                column=1,
                sticky="w")

artist_label = ctk.CTkLabel(
    frame,
    text=artist,
    font=ctk.CTkFont(size=12),
    text_color="white",
    fg_color="black"
)
artist_label.grid(row=1,
                  column=1,
                  sticky="w")

time_label = ctk.CTkLabel(
    frame,
    text=f"{minutes}:{seconds:02d}",
    font=ctk.CTkFont(size=11),
    text_color="white",
    fg_color="black"
)
time_label.grid(row=2,
                column=1,
                sticky="w")

frame.bind("<Button-1>", lambda e, uri=track['uri'], idx=i: self.play_track(uri, idx))
for child in frame.winfo_children():
    child.bind("<Button-1>", lambda e, uri=track['uri'], idx=i: self.play_track(uri, idx))
```

```
    def play_track(self, uri, index=None):
        try:
            sp.start_playback(uris=[uri])
            if index is not None:
                self.current_track_index = index
                self.monitor_track_end()
        except Exception as e:
            print(f"Playback error: {e}")

    def monitor_track_end(self):
        def check():
            try:
                playback = sp.current_playback()
                if playback and not playback['is_playing']:
                    self.play_next_track()
                else:
                    self.after(1000, check)
            except Exception as e:
                print(f"Monitor error: {e}")

        self.after(1000, check)

    def play_next_track(self):
        next_index = self.current_track_index + 1
        if next_index < len(self.track_list):
            next_uri = self.track_list[next_index]['uri']
            self.play_track(next_uri, next_index)

    def go_back(self):
        for widget in self.scrollable_frame.winfo_children():
            widget.destroy()
        self.populate_albums()

    def on_enter(self, event):
        self.parent.is_cursor_on_window = True
        self.parent.cancel_withdraw()

    def on_leave(self, event):
        self.parent.is_cursor_on_window = False
        self.parent.withdraw_task = self.parent.after(5000, self.parent.withdraw_if_cursor_left)

    def monitor_parent_minimize(self):
        if not self.parent.winfo_ismapped():
            self.destroy()
        else:
            self.after(500, self.monitor_parent_minimize)
```

settings.py

```
import json
import os

SETTINGS_FILE = "settings.json"

DEFAULT_SETTINGS = {
    "suppress_on_track_change": False
}

def load_settings():
    if os.path.exists(SETTINGS_FILE):
        with open(SETTINGS_FILE, "r") as f:
            try:
                return json.load(f)
            except json.JSONDecodeError:
                return DEFAULT_SETTINGS.copy()
    return DEFAULT_SETTINGS.copy()

def save_settings(settings: dict):
    with open(SETTINGS_FILE, "w") as f:
        json.dump(settings, f, indent=4)
```

start.py

```
from window import MediaPlayerApp
from func import cleanup_cache

if __name__ == "__main__":
    try:
        app = MediaPlayerApp()
        app.mainloop()
    finally:
        cleanup_cache()
```

func.py

```
import os
import spotipy
from spotipy.oauth2 import SpotifyOAuth
import ctypes

client_id = "e6fcb8404eb641a9abaf146d39f526d5"
client_secret = "72cb70c70a7c46d58127007a97e7bccc"
redirect_uri = "http://127.0.0.1:8888/callback"

sp = spotipy.Spotify(auth_manager=SpotifyOAuth(client_id=client_id,
```

```
                                client_secret=client_secret,
                                redirect_uri=redirect_uri,
                                scope=["user-read-private user-read-email user-read-playback-state user-
modify-playback-state user-library-read"]))

def get_current_track():
    try:
        track = sp.current_playback()
        if track is None:
            return None
        track_name = track["item"]["name"]
        artist_name = track["item"]["artists"][0]["name"]
        album_cover = track["item"]["album"]["images"][0]["url"]
        return track_name, artist_name, album_cover
    except Exception as e:
        print(f"Error fetching current track: {e}")
        return None

def has_premium():
    try:
        user = sp.current_user()
        return user.get("product") == "premium"
    except Exception as e:
        print(f"Error checking premium status: {e}")
        return False

def cleanup_cache():
    cache_file = ".cache"
    if os.path.exists(cache_file):
        try:
            os.remove(cache_file)
            print("Файл .cache видалено.")
        except Exception as e:
            print(f"Не вдалося видалити файл .cache: {e}")

def simulate_keypress(key_code):
    # key_code: 0xB1 (Prev), 0xB3 (Play/Pause), 0xB0 (Next)
    VK_MEDIA = key_code
    KEYEVENTF_EXTENDEDKEY = 0x0001
    KEYEVENTF_KEYUP = 0x0002

    ctypes.windll.user32.keybd_event(VK_MEDIA, 0, KEYEVENTF_EXTENDEDKEY, 0)
    ctypes.windll.user32.keybd_event(VK_MEDIA, 0, KEYEVENTF_EXTENDEDKEY |
KEYEVENTF_KEYUP, 0)
```

about.py

```
import customtkinter as ctk
from PIL import Image, ImageTk

class AboutWindow(ctk.CTkToplevel):
```

```
def __init__(self, parent):
    super().__init__(parent)
    self.title("Про програму")
    self.geometry("600x500")
    self.configure(fg_color="black")

    # Головний фрейм
    main_frame = ctk.CTkFrame(self,
                              fg_color="black")
    main_frame.pack(expand=False,
                    fill="x",
                    padx=20,
                    pady=(20, 10))

    # Завантаження фото
    try:
        img = Image.open("about.jpg").resize((200, 250))
        photo = ImageTk.PhotoImage(img)
    except Exception as e:
        print(f"Не вдалося завантажити зображення: {e}")
        photo = None

    # Ліва частина — фото
    left_frame = ctk.CTkFrame(main_frame,
                              fg_color="black")
    left_frame.grid(row=0,
                    column=0,
                    padx=(0, 20),
                    sticky="n")

    if photo:
        img_label = ctk.CTkLabel(left_frame,
                                 image=photo,
                                 text="")
        img_label.image = photo
        img_label.pack()

    # Права частина — текст
    right_frame = ctk.CTkFrame(main_frame,
                               fg_color="black")
    right_frame.grid(row=0,
                     column=1,
                     sticky="nw")

    info_text = (
        "Ім’я: Антон\n"
        "Прізвище: Щерба\n"
        "По батькові: Олегович\n"
        "Група: 2ПІ-21б\n"
        "Факультет: ФІТКІ\n"
```

```
        "Версія застосунку: 1.01"
    )

    info_label = ctk.CTkLabel(
        right_frame,
        text=info_text,
        font=ctk.CTkFont(size=14),
        text_color="white",
        justify="left"
    )
    info_label.pack(anchor="w")

    # Пропуск перед заголовком
    ctk.CTkLabel(self,
                 text="",
                 height=10,
                 fg_color="black").pack()

    # Заголовок Про програму
    title_label = ctk.CTkLabel(
        self,
        text="Про програму",
        font=ctk.CTkFont(size=20,
                         weight="bold"),
        text_color="white"
    )
    title_label.pack(pady=(0, 10))

    # Нижній блок — опис програми
    description = (
        "Ця програма — це графічний медіаплеєр для Windows, створений з використанням
бібліотеки "
        "customtkinter, який взаємодіє з обліковим записом користувача Spotify через офіційний
API. "
        "Вона дозволяє керувати відтворенням музики: перемикати треки, ставити на паузу,
змінювати гучність, "
        "переглядати обкладинку альбому та інформацію про композицію. Також програма
перевіряє, чи має користувач "
        "підписку Spotify Premium, і залежно від цього змінює поведінку плеєра (наприклад,
керування треками через API або "
        "емуляція натискання клавіш).\n\n"
        "Додатково програма підтримує приховування вікна, коли курсор миші не знаходиться в
межах інтерфейсу, "
        "а також відображає вікно з налаштуваннями, дозволяє переглядати альбоми та
перемикатися в режим перегляду YouTube. "
        "Вона постійно слідкує за змінами гучності або треку й автоматично реагує на ці події."
    )

    description_label = ctk.CTkLabel(
        self,
```

```
            text=description,
            wraplength=560,
            font=ctk.CTkFont(size=12),
            text_color="white",
            justify="left"
        )
        description_label.pack(padx=20, pady=(0, 20))
```

youtube.py

```
import customtkinter as ctk
import threading
import os
import time
import requests
import ctypes
from PIL import Image, ImageTk
from io import BytesIO
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
from googleapiclient.discovery import build
from google.oauth2.credentials import Credentials
from ctypes import POINTER, cast
from comtypes import CLSCTX_ALL
from pycaw.pycaw import AudioUtilities, IAudioEndpointVolume

class YouTubeWindow(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("YouTube Media Player")
        self.geometry("360x450+1400+20")
        self.attributes("-topmost", True)

        self.visible = True
        self.last_volume = self.get_system_volume()
        self.youtube_service = None
        self.thumbnail_image = None

        self.build_ui()

        threading.Thread(target=self.authenticate_youtube,
                         daemon=True).start()
        threading.Thread(target=self.auto_hide_timer,
                         daemon=True).start()
        threading.Thread(target=self.monitor_volume_change,
                         daemon=True).start()

    def build_ui(self):
        self.configure(fg_color="#000000")
```

```
self.spotify_button = ctk.CTkButton(
    self,
    text="Spotify",
    command=self.switch_to_spotify,
    fg_color="#1db954",  # Зелений фон
    text_color="black",  # Чорний текст
    hover_color="#1ed760"  # Трохи яскравіший зелений при наведенні
)
self.spotify_button.place(relx=0.5,
                          rely=0.0,
                          anchor="n",
                          y=5)

self.thumbnail_label = ctk.CTkLabel(self,
                                    text="")
self.thumbnail_label.pack(pady=(40, 10))

self.video_title_label = ctk.CTkLabel(
    self,
    text="Завантаження...",
    font=ctk.CTkFont(size=18,
                     weight="bold"),
    wraplength=320,
    justify="center",
    text_color="white"  # Білий текст
)
self.video_title_label.pack(pady=(5, 2))

self.channel_title_label = ctk.CTkLabel(
    self,
    text="",
    font=ctk.CTkFont(size=14),
    wraplength=300,
    justify="center",
    text_color="white"  # Білий текст
)
self.channel_title_label.pack()

self.button_frame = ctk.CTkFrame(self,
                                 fg_color="transparent")
self.button_frame.pack(pady=10)

# Червоні кнопки
button_style = {
    "fg_color": "#ff4c4c",  # Червоний фон
    "hover_color": "#ff6666",  # Світліший червоний при наведенні
    "text_color": "white",
    "width": 50
}
```

```
    self.play_pause_button = ctk.CTkButton(
        self.button_frame,
        text="⏯️",
        command=self.toggle_play_pause,
        **button_style
    )
    self.play_pause_button.pack(side="left", padx=10)

    self.volume_button = ctk.CTkButton(
        self.button_frame,
        text="🔊",
        command=self.toggle_volume_slider,
        **button_style
    )
    self.volume_button.pack(side="left", padx=10)

    self.volume_slider = ctk.CTkSlider(
        self,
        from_=0,
        to=1,
        command=self.change_volume,
        button_color="#ff4c4c",  # Червона точка
        progress_color="#ff4c4c"  # Червона лінія прогресу
    )
    self.volume_slider.set(self.get_system_volume())
    self.volume_slider.pack(pady=5)
    self.volume_slider.pack_forget()

    self.volume_button.bind("<Button-3>", self.toggle_mute)

def authenticate_youtube(self):
    creds = None
    scopes = ["https://www.googleapis.com/auth/youtube.readonly"]

    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token.json', scopes)

    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            try:
                creds.refresh(Request())
            except Exception as e:
                print(f"Помилка оновлення токену: {e}")
                creds = None

        if not creds:
            try:
                flow = InstalledAppFlow.from_client_secrets_file(
                    'client_secret.json',
```

```
                scopes=scopes
            )
            creds = flow.run_local_server(port=0)
        except Exception as e:
            print(f"Помилка авторизації: {e}")
            return

    with open('token.json', 'w') as token_file:
        token_file.write(creds.to_json())

  self.youtube_service = build('youtube', 'v3', credentials=creds)
  threading.Thread(target=self.update_video_info,
                   daemon=True).start()

def update_video_info(self):
  while True:
    try:
      request = self.youtube_service.videos().list(
        part="snippet",
        myRating="like",
        maxResults=1
      )
      response = request.execute()

      if response["items"]:
        video = response["items"][0]["snippet"]
        title = video["title"]
        channel_title = video["channelTitle"]
        thumbnail_url = video["thumbnails"]["medium"]["url"]

        self.update_ui(title, channel_title, thumbnail_url)
      else:
        self.video_title_label.configure(text="Немає відео")
        self.channel_title_label.configure(text="")
        self.thumbnail_label.configure(image=None)

    except Exception as e:
      print(f"Помилка отримання інформації про відео: {e}")

    time.sleep(10)

def update_ui(self, title, channel_title, thumbnail_url):
  self.video_title_label.configure(text=title)
  self.channel_title_label.configure(text=f"Автор: {channel_title}")

  try:
    response = requests.get(thumbnail_url)
    img_data = response.content
    img = Image.open(BytesIO(img_data))
    img = img.resize((320, 180))
```

```
            self.thumbnail_image = ImageTk.PhotoImage(img)
            self.thumbnail_label.configure(image=self.thumbnail_image)
        except Exception as e:
            print(f"Помилка завантаження обкладинки: {e}")

    def switch_to_spotify(self):
        self.destroy()
        from window import MediaPlayerApp
        app = MediaPlayerApp()
        app.mainloop()

    def hide_window(self):
        if self.visible:
            self.withdraw()
            self.visible = False

    def show_window(self):
        if not self.visible:
            self.deiconify()
            self.visible = True

    def auto_hide_timer(self):
        while True:
            time.sleep(5)
            if not self.is_cursor_inside():
                self.hide_window()

    def is_cursor_inside(self):
        self.update_idletasks()
        x, y = self.winfo_pointerx(), self.winfo_pointery()
        win_x = self.winfo_rootx()
        win_y = self.winfo_rooty()
        win_width = self.winfo_width()
        win_height = self.winfo_height()

        return win_x <= x <= win_x + win_width and win_y <= y <= win_y + win_height

    def toggle_play_pause(self):
        # Симулюємо натискання Play/Pause
        self.simulate_keypress(0xB3)

    def toggle_volume_slider(self):
        if self.volume_slider.winfo_ismapped():
            self.volume_slider.pack_forget()
        else:
            self.volume_slider.pack(pady=5)

    def change_volume(self, volume_level):
        devices = AudioUtilities.GetSpeakers()
        interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
```

```
    volume = cast(interface, POINTER(IAudioEndpointVolume))
    volume.SetMasterVolumeLevelScalar(volume_level, None)

def toggle_mute(self, event=None):
    # Симулюємо натискання Mute
    self.simulate_keypress(0xAD)

def simulate_keypress(self, key_code):
    ctypes.windll.user32.keybd_event(key_code, 0, 0, 0)  # натиснули клавішу
    ctypes.windll.user32.keybd_event(key_code, 0, 2, 0)  # відпустили клавішу

def get_system_volume(self):
    devices = AudioUtilities.GetSpeakers()
    interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL, None)
    volume = cast(interface, POINTER(IAudioEndpointVolume))
    return volume.GetMasterVolumeLevelScalar()

def monitor_volume_change(self):
    while True:
        current_volume = self.get_system_volume()
        if abs(current_volume - self.last_volume) > 0.01:
            self.last_volume = current_volume
            self.show_window()
        time.sleep(0.5)
```

Додаток Г – Ілюстративна частина

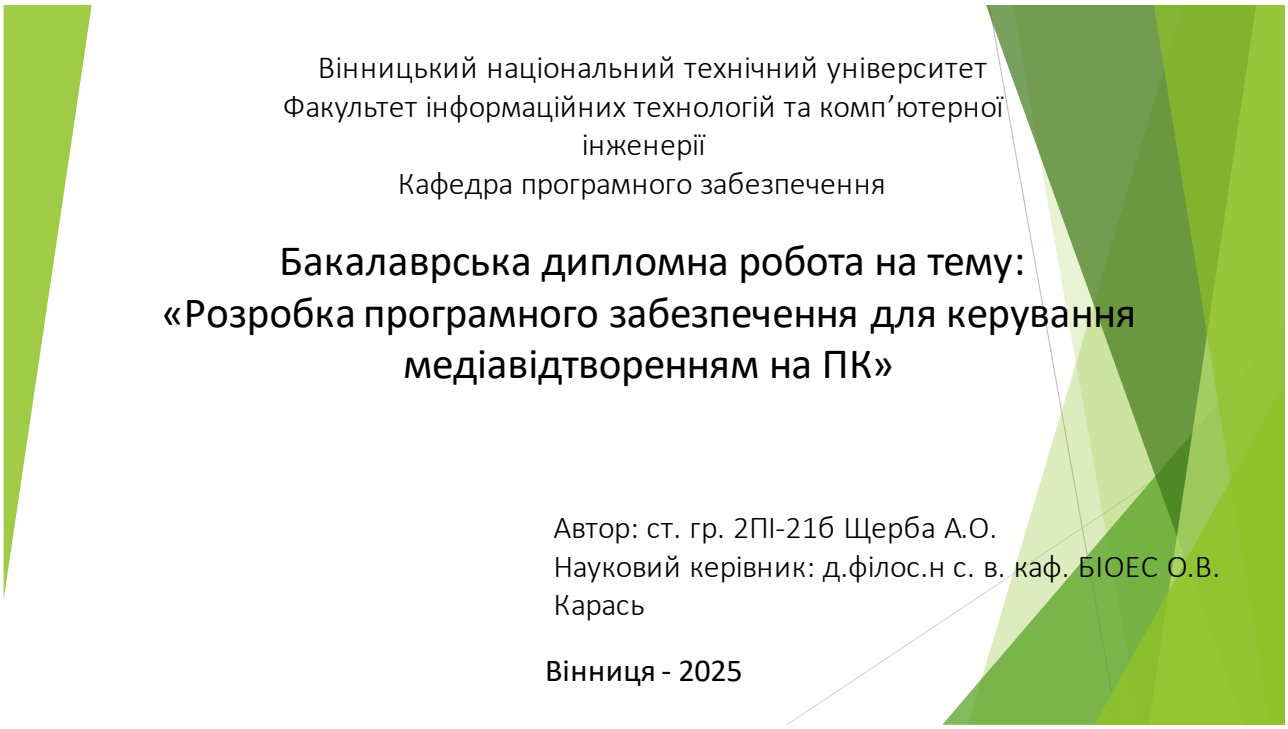

Рисунок Г.1 – Титульний слайд

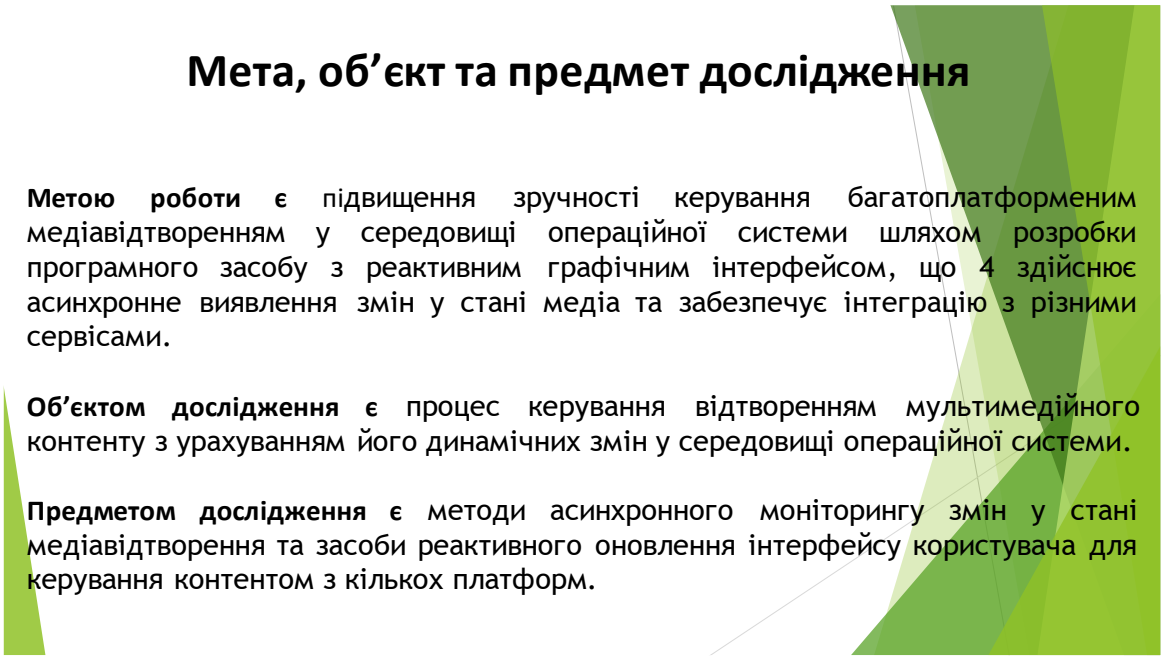

Рисунок Г.2 – Мета, об'єкт та предмет дослідження

Задачі дослідження

– провести аналіз сучасних інтерфейсних рішень для керування медіавідтворенням;

– сформувати функціональні та нефункціональні вимоги до системи асинхронного контролю медіа;

– розробити архітектуру програмного багатопотокового моніторингу змін параметрів медіа;

– реалізувати подієву модель реактивного оновлення інтерфейсу користувача;
– забезпечити інтеграцію з API Spotify, YouTube та SoundCloud для отримання актуального медіаконтенту;
– реалізувати графічний інтерфейс з автоматичним відображенням інформації про трек та елементами керування;
– провести тестування працездатності системи в умовах реального часу;

– розробити інструкцію користувача.

Рисунок Г.3 – Задачі дослідження

Новизна і практична цінність отриманих результатів

Запропоновано асинхронну архітектуру для автоматичного виявлення та реакції на зміну медіа-стану (гучність, трек).
Багатопотоковий моніторинг: два потоки незалежно відстежують гучність і зміну треку у Spotify.
Черга подій: при зміні стану в чергу надсилаються події типу volume_changed , media_changed .
Автоматична реакція: головний потік інтерфейсу обробляє події та оновлює UI
Швидкодія: затримка реакції < 1 с, що створює ефект "миттєвого оновлення"

Практична цінність отриманих результатів полягає у створенні зручного та ефективного інтерфейсу для керування медіавідтворенням через декілька платформ, що дозволяє користувачам зекономити час і покращити взаємодію з медіаконтентомпри багатозадачності

Рисунок Г.4 – Новизна і практична цінність отриманих результатів

Порівняльний аналіз аналогів

Застосунок	Переваги	Недоліки
ModernFlyouts	Гнучкість налаштувань, низьке споживання ресурсів	Відсутність інтеграції з медіасервісами
FluentFlyout	Мінімалістичний дизайн, ефективність на слабких пристроях	Обмежена функціональність, відсутність кастомізації
AudioFlyout	Легкість, швидкість, стабільність	Відсутність інтеграції з сервісами, базова функціональність

Рисунок Г.5 – Порівняльний аналіз аналогів

Архітектура застосунку

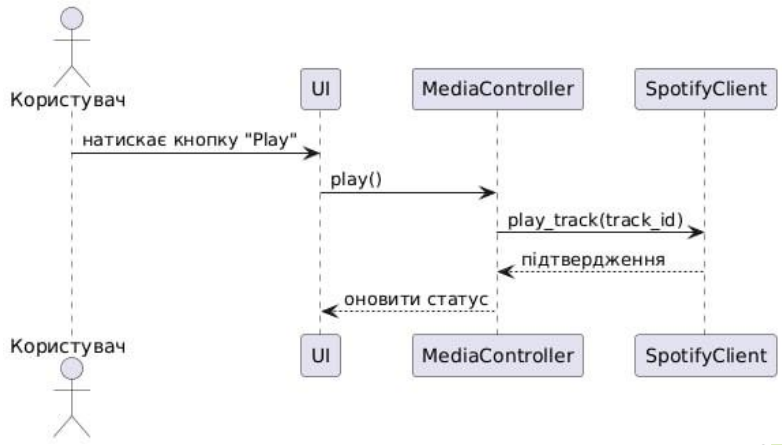

Рисунок Г.6 – Архітектура застосунку

Загальний алгоритм роботи системи

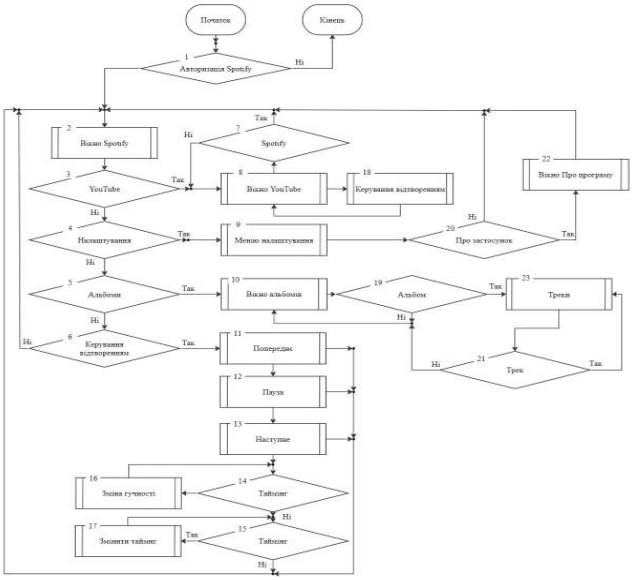

Рисунок Г.7 – Загальний алгоритм роботи системи

Алгоритм завантаження та відображення інформації про поточний трек

1. Отримання даних про трек

- Викликається функція, що використовує API для отримання даних про поточний трек (назва, артист, обкладинка).
- Перевірка на наявність треку
- Якщо трек знайдено, відбувається оновлення елементів інтерфейсу

2. Відображення вікна:

- Вікно показується або активується, якщо це необхідно
- У разі необхідності, вікно автоматично приховується через 5 секунд після того, як курсор залишає вікно.

3. Обробка поточного прогресу треку

- Відстежується поточний час відтворення
- Оновлюється текст для поточного часу і прогрес-бар.

4. **Зміна позиції треку**

- Користувач може переміщатися по треку за допомогою слайдера .

5. **Обробка черги подій**

- Черга обробляє події, такі як зміна гучності або зміна треку.

Назва функції	Опис функції
get_current_track	Отримує дані про поточний трек (назва, артист, обкладинка, прогрес).
update_track_progress	Оновлює прогрес треку, відображаючи поточний час і оновлюючи прогрес -бар.
seek_position	Переміщує позицію в треку на задану позицію за допомогою слайдера.
show_window	Відображає або активує вікно, оновлюючи інформацію про трек.
monitor_media	Перевіряє, чи змінився поточний трек, і виводить нову інформацію про трек.
process_queue	Обробляє чергу повідомлень про зміни гучності та медіа і оновлює вікно.

Рисунок Г.8 – Алгоритм завантаження та відображення інформації про поточний трек

Демонстрація інтерфейсу застосунку «Початковий екран»

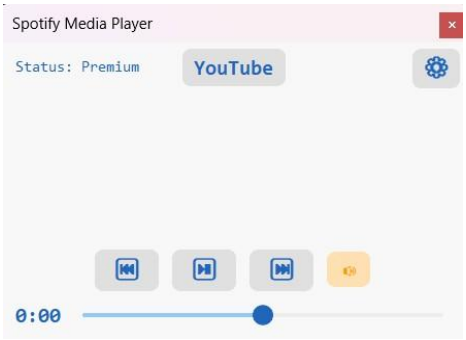

Рисунок Г.9 – Демонстрація інтерфейсу застосунку «Початковий екран»

Демонстрація інтерфейсу застосунку «Режим Spotify»

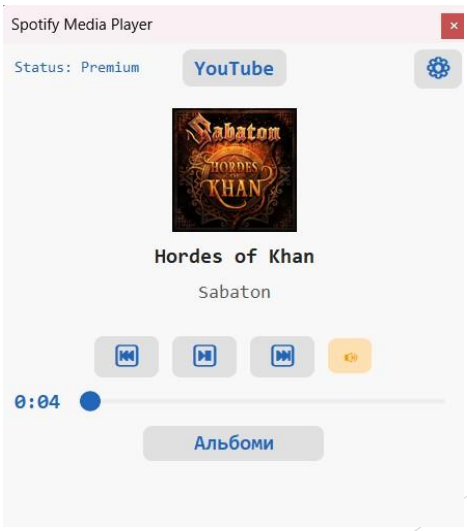

Рисунок Г.10 – Демонстрація інтерфейсу застосунку «Режим Spotify»

Демонстрація інтерфейсу застосунку «Режим Youtube»

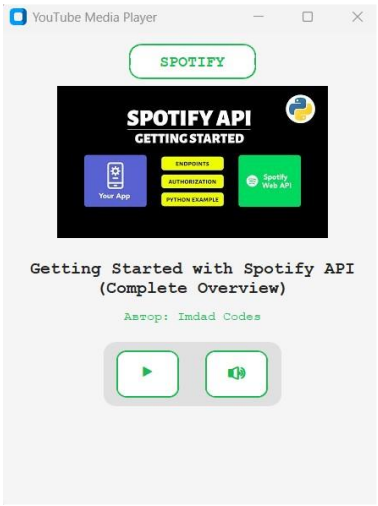

Рисунок Г.11 – Демонстрація інтерфейсу застосунку «Режим Youtube»

Демонстрація інтерфейсу застосунку «Вибір альбому»

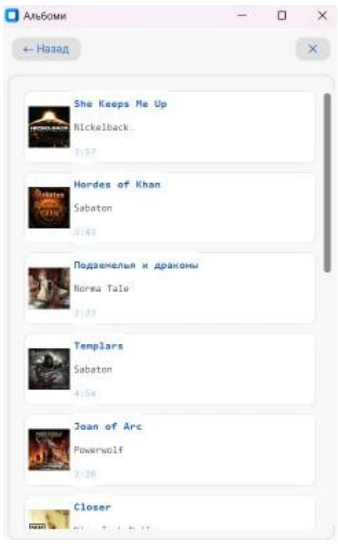

Рисунок Г.12 – демонстрація інтерфейсу застосунку «Вибір альбому»

Тестування авторизації Spotify

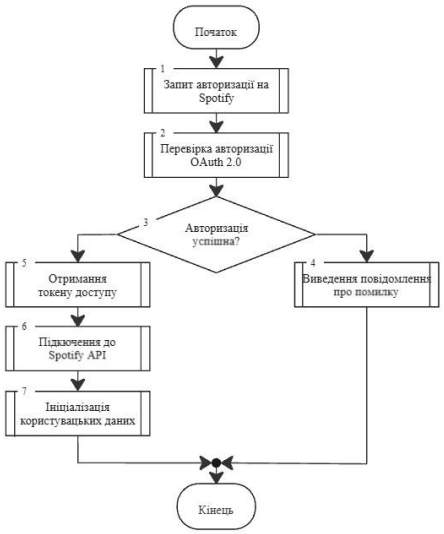

Рисунок Г.13 – Тестування авторизації Spotify

Тестування алгоритму управління аудіовідтворенням

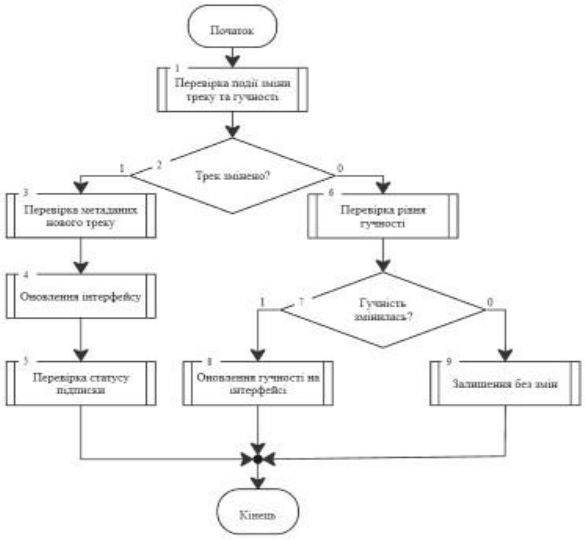

Рисунок Г.14 – Тестування алгоритму управління аудіовідтворенням

Апробація та публікація матеріалів бакалаврської дипломної роботи

Результати роботи доповідалися на:

- Молодь в науці: дослідження, проблеми, перспективи (МН-2025), Вінниця 2025.
- Всеукраїнська науово-технічна конференція факультету інформаційних технологій та комп'ютерної інфенерії

За тематикою дослідження опубліковано дві наукові праці у збірниках матеріалів конференцій.

Рисунок Г.15 – Апробація та публікація матеріалів бакалаврської дипломної роботи

Дякую за увагу!

Рисунок Г.15 – Фінальний слайд