Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти

(повна назва)

Кафедра програмної інженерії

(повна назва)

**КВАЛІФІКАЦІЙНА РОБОТА**

**Пояснювальна записка**

рівень вищої освіти перший (бакалаврський)

Програмне забезпечення для обліку музичних творів

(тема)

Виконав:

здобувач \_\_\_\_\_\_\_2\_\_\_\_\_\_ року навчання

групи ПЗПІпз-23-1

Юлія ТОРОХТІЙ

(Власне ім’я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного\_ \_\_

забезпечення

(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Програмна інженерія

(повна назва освітньої програми)

Керівник ст.викл. Дмитро МАТВЄЄВ

(посада, Власне ім’я, ПРІЗВИЩЕ)

Допускається до захисту

Зав. кафедри \_\_\_\_\_\_\_\_\_\_\_ Юлія ТОРОХТІЙ

(підпис) (Власне ім’я, ПРІЗВИЩЕ)

2025 р.

Харківський національний університет радіоелектроніки

Факультет центр післядипломної освіти

Кафедра програмної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-професійна

Освітня програма Програмна Інженерія

(шифр і назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри \_\_\_\_\_\_\_\_\_

(підпис)

« » 2025 р.

**ЗАВДАННЯ**

НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Торохтій Юлії Володимирівни

(прізвище, ім’я, по батькові)

1. Тема роботи Програмне забезпечення для обліку музичних творів

Затверджена наказом по університету від 19.05.2025р. № 86Стз

2. Термін подання студентом роботи до екзаменаційної комісії 26.06.2025 \_

3. Вихідні дані до роботи Розробити програмне забезпечення для обліку музичних творів на мові Python з використанням Tkinter та SQL Server.

4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз предметної галузі, формування вимог до програмної системи, архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень, тестування розробленого програмного забезпечення, висновки, додатки.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
| 1 | Аналіз предметної галузі | 20.04.2025 | *виконано* |
| 2 | Створення специфікації ПЗ | 22.04.2025 | *виконано* |
| 3 | Проектування ПЗ | 24.04.2025 | *виконано* |
| 4 | Розробка ПЗ | 30.04.2025 | *виконано* |
| 5 | Тестування ПЗ | 05.05.2025 | *виконано* |
| 6 | Оформлення пояснювальної записки | 10.05.2025 | *виконано* |
| 7 | Підготовка презентації та доповіді | 20.05.2025 | *виконано* |
| 8 | Попередній захист | 21.06.2025 | *виконано* |
| 9 | Нормоконтроль, рецензування | 21.06.2025 | *виконано* |
| 10 | Здача роботи у електронний архів | 23.06.2025 | *виконано* |
| 11 | Допуск до захисту у зав. кафедри | 23.06.2025 | *виконано* |

Дата видачі завдання «07» « квітня » 2025р.

Здобувач \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Керівник роботи ст.викл. Дмитро МАТВЄЄВ

(підпис) (посада, Власне ім’я, ПРІЗВИЩЕ)

**РЕФЕРАТ**

Пояснювальна записка до кваліфікаційної роботи бакалавра, 64 стор., 27 рис., 5 табл., 12 джерел.

ГРАФІЧНИЙ ІНТЕРФЕЙС, ІНФОРМАЦІЙНА СИСТЕМА, ОБЛІК МУЗИКИ, PYTHON, SQL SERVER, TKINTER

Метою проекту є розробка інтуїтивно зрозумілої системи, яка дозволяє легко зберігати, оновлювати та відображати музичні записи через простий інтерфейс та ефективну структуру бази даних.

У реалізації використовувалися такі інструменти: Python, Visual Studio 2022, графічна бібліотека Tkinter, СУБД SQL Server та драйвер pyodbc, який полегшує взаємодію з базою даних.

Результатом є практична система для легкого керування обліку музичних творів: додавання нових записів, редагування існуючих записів, видалення небажаних записів та пошуку інформації у зручному форматі. Система розроблена для практичності та може бути адаптована до потреб музичних студій, організаторів подій чи особистого застосування.

GRAPHICAL INTERFACE, INFORMATION SYSTEM, MUSIC ACCOUNTING, PYTHON, SQL SERVER, TKINTER

The goal of this project is to develop an intuitive system that allows you to easily store, update and display musical records through a simple interface and an effective database structure.

The following tools were used in the implementation: Python, Visual Studio 2022, the Tkinter graphic library, SQL Server DBMS and the pyodbc driver, which facilitates interaction with the database.

The result is a practical system for easy management of musical records: adding new records, editing existing records, deleting unwanted records and searching for information in a convenient format. The system is designed for practicality and can be adapted to the needs of music studios, event organizers or personal use.

**ЗМІСТ**

Перелік скорочень 8

[Вступ](#_Toc105146293) 9

[1 Аналіз предметної галузі](#_Toc105146294) 10

[1.1 Аналіз предметної галузі](#_Toc105146295) 10

[1.2 Виявлення та вирішення проблем](#_Toc105146296) 15

[1.3 Постановка задачі](#_Toc105146297) 17

[2 Формування вимог до програмної системи](#_Toc105146300) 20

2.1 Загальні характеристики програмної системи 20

2.2 Класифікація користувачів та права доступу 20

2.3 Функціональні вимоги 20

2.4 Нефункціональні вимоги 21

2.5 Технічні вимоги до середовища виконання 22

[3 Архітектура та проєктування програмного забезпечення](#_Toc105146301) 23

[3.1 UML проєктування ПЗ](#_Toc105146302) 23

[3.2 Проєктування архітектури ПЗ](#_Toc105146303) 26

[3.3 Проєктування структури зберігання даних](#_Toc105146304) 28

[3.4 Приклади найцікавіших алгоритмів та методів](#_Toc105146305) 31

[3.5 Створення UI/UX](#_Toc105146306) 33

[4 Опис прийнятих програмних рішень](#_Toc105146307) 36

[4.1 Створення автентифікації, авторизації та керуванням ролями](#_Toc105146308) 36

[4.2 Створення під’єднання до СКБД та архітектура транзакцій](#_Toc105146309) 38

[4.3 Створення алгоритму формування пошукового запиту](#_Toc105146310) 38

4.4 Створення графічного інтерфейсу таблиць 39

4.5 Створення гіперпосилань та взаємодія з веб-браузером 42

4.6 Розширення перспективи застосунку 43

[5 Тестування програмного забезпечення](#_Toc105146311) 45

[5.1 Тестування програмного забезпечення](#_Toc105146312) 45

[6 Впровадження програмного забезпечення](#_Toc105146314) 50

[6.1 Визначення плану впровадження](#_Toc105146315) 50

[6.2 Середовище застосування та підключення](#_Toc105146316) 50

[Висновки](#_Toc105146317) 52

[Перелік джерел посилання](#_Toc105146318) 54

[Додаток А](#_Toc105146319) 56

[Додаток Б](#_Toc105146319) 58

**ПЕРЕЛІК СКОРОЧЕНЬ**

ПЗ – Програмне забезпечення

БД – База даних

СУБД – Система управління базами даних

SQL – Structured Query Language

GUI – Graphical User Interface

ODBC – Open Database Connectivity

**ВСТУП**

Музика супроводжує людей майже щодня і всюди, вдома, на вулиці, у транспорті чи на роботі. Зараз вона найчастіше використовується в цифровому форматі, що робить її зручнішою для зберігання, прослуховування та поширення. Ще з давніх часів музика мала великий вплив на людей, але сьогодні її роль стала ще більш помітною: вона активно використовується в організації заходів, освітньому процесі, лікуванні, рекламних кампаніях, бізнесі й навіть у повсякденному житті. Фактично, музика стала звичною частиною будь-якої діяльності. Вона створює настрій, викликає емоції, надихає або навпаки, заспокоює. Для багатьох людей вона стала способом висловити себе або краще зрозуміти інших. Її можна розглядати не просто як розвагу, а як окрему форму спілкування, яка часом говорить глибше за слова. Тому розвиток музичної індустрії завжди росте, а з нею зростає кількість інформації про пісні, починаючи з виконавців, альбомів закінчуючи жанрами, і при цьому зростає потреба у зручному обліку збору та зберіганню музичних творів та інформації до них.

Метою роботи є розробка системи під назвою «Програмне забезпечення для обліку музичних творів». Система буде зберігати данні, редагувати та видаляти твори, а також сортирувати по таким критеріям назвою пісні, виконавцем, країною та роком. Сама ж система написана на мові програмування Python з використання базою даних Microsoft SQL Server. У проекті досліджувалась предметна область, визначалися основні вимоги до системи та створено структуру бази даних з таблицями, котра зберігає ключову інформацію. Самі можливості програми передбачають зручний пошук, сортуванню даних по інтересам користувачів, зручне прослуховування музики через посилання, а також надання певних прав доступу до системи, такі як, адміністратор, користувач та гість. Додавати та редагувати пісні може тільки адміністратор, користувач тільки додавати, а гість переглядати без прав на зміни.

Проект зроблений у середовищі Visual Studio 2022, інтерфейс збудований через бібліотеку Tkinter [1], а з базою даних програма працює через модуль pyodbc, а зберігання інформації робиться через базу SQL Server.

**1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ**

1.1 Аналіз предметної галузі

Музичні твори вже давно стали методом передавання інформації у людьскій культурі, адже вона супроводжує нас усе життя, починаючи з простих ритуальних та культурних пісень кожного народа, закінчуючи сучасними електронними творами, котрі відтворюються по всьому світі через глобальні цифрові платформи.

З часом музику почали переводити на комп’ютери за допомогою сучасних технологій, в свою же чергу вона привела до ще більшого впливу користувачів по всьому світі, адже тепер кожна людина може послухати те що йому хочеться. А на сьогоднішній день люди перейшли вже на мобільні пристрої що позволило в свою чергу коли завгодно включати твори, адже у кожного із нас зараз у кармані є телефон. Споживання контенту більш за все проходить у потокових, локальних сервісах і приватних колекціях та бібліотеках.

Сама ж музика історично переїжджала з одного формату до іншого. Спочатку це були грамплатівки, потім касетами, через час компакт-дисками. І тільки з релізом МР3 [2] людство перейшло на передачу інформації по інтернету, але на сьогодні це вже застаріло і почало витісняти навіть звичайне скачування пісень. З кожним разом і вдосконаленням технологій люди поступово забували минулі формати, а так само перестаючи так дбайливо ставиться до носіїв музики, наприклад до вінілових платівки і диски, кожен з людей зберігав вдома дбайливо ці формати, намагаючись щоб вони не зламалися і не подряпалися, або наприклад касетний плеєр [3].

Нині прослуховування музики це вже давно не про плеєри та диски та завантажені пісні на комп'ютер. Людство перейшло на онлайн сервіси, найпопулярніші з них, Spotify, YouTube Music, Apple Music [3]. Але кожен із цих додатків має свою особливість не схожу на інших, у Spotify працює автоматична пропозиція музики щодо настрою, давніх прослуховувань. YouTube Music надає доступ до живих виступів та рідкісних записів, а також фанатські кавери. Apple Music використовується на пристроях Apple із роботою екосистеми компанії, надаючи користувачам особливу зручність.

Але якими б вони не були популярними, навіть у Spotify, YouTube Music, Apple Music є свої плюси і мінуси наведено в таблиці 1.1

Таблиця 1.1 – Переваги та недоліки онлайн сервісів (таблиця виконана самостійно)

|  |  |  |
| --- | --- | --- |
| Сервіси | Переваги | Недоліки |
| Spotify | Найкраща система персональних рекомендацій | Немає офіційного запуску в Україні |
| Величезна спільнота, готові плейлисти | Часто відсутні локальні релізи чи рідкісні альбоми |
| Підкасти та аудіокниги в одному додатку | Безкоштовна версія з рекламою та обмеженням |
| Lossless-якості поки що немає |
| YouTube Music | Найшвидше знаходить концертні записи, кавери, фан-віджи | Без підписки не можна слухати на з виключеним екраном |
| Плейлисти автоматично підтягують відео з основного YouTube | Багато реклами навіть усередині пісень |
| Якість звуку помітно нижча, ніж у конкурентів |
| Плейлисти часто засмічуються  незв'язаними роликами та мемами |
| Apple Music | Глибока інтеграція з iPhone, Apple Watch, Siri | На Android та Windows додаток «важкий», працює повільніше |
| Підтримка Lossless та Dolby Atmos без доплати | Немає безкоштовної версії зовсім |
| Редакційні добірки та ексклюзивні релізи | Неможливо ділитися посиланням на пісню, якщо друг не має передплати |

Але незважаючи на сучасні зручності використання нових сервісів, деякі слухачі не перестають зберігати музику локально, тому що рідкісні твори не завжди зберігаються через проблеми ліцензій в каталогах [4]. А так само, при поганому з'єднанні зв'язку інтернету, буває з'являється переривання пісні, тоді як офлайн-файл завжди грає без перебоїв, пов'язаних з мережею. Крім того, локальні плеєри на прикладі Foobar2000, MusicBee, AIMP [4] надають великий спосіб налаштування еквалайзера, на відміну від онлайн сервісів.

А якщо твір завантажено за допомогою FLAC, користувач на додаток отримує і студійну якість програвання, якого досі немає у безкоштовних версіях онлайн сервісах.

У зв'язку з цим нинішні користувачі мають інше уявлення про музику, ніж люди в минулому, зараз кожен може послухати те, що йому хочеться в будь-якому місці та часі, нинішня музика стала формою зручності.

Впливання музики на людей можна помітити неозброєним поглядом, тому що вона впливає на емоційний стан, може створювати різну атмосферу, створювати культурне середовище, допомагати у роботі, сприяти навчанню, а також є способом розваження, та натхнення. Також активне застосування отримала в психології, спеціально підібрані розслаблюючі плейлисти знижують тривожність у пацієнтів і допомагають впоратися з безсонням.

З давніх часів протягом усієї історії людства, музика займає важливу роль у житті нашого світу. Завдяки їй люди пізнавали інформіцію та передавали її своїм дітям. А племена використовували її у своїх обрядах і традиціях, Але с появою нотацій люди перейшли на «оцифрування» звуку, що дало змогу дати шлях до массового копіювання та архівації всіх творів з усіх куточків світу.

Поступово з розвитком технологій з’явилися фоноциліндри, магнітні стрічки, оптичні диски, а вони почали збільшували щільність даних, не змінюючи парадигму: музика і так далі залишалась фізичним об’єктом . Тільки ера цифрових форматів (MP3, FLAC, AAC) зникли у звичайному розумінні, а доступність електронних творів почала визначатися пропускною здатністю мережі та ефективністю метаданих.

Керування накопиченою кількістю музичних творів не завжди є простою роботою для користувачів. Зазвичай люди зберігають музичні твори файлами на комп’ютерах, а також на потокових сервісах у списках для відтворення або розкидану інформацію по різним каталогам і носіям. А для повного та ефективного організаційного зберігання, пошуку, а також відтворення творів необхідний інструмент, який вирішить цю проблему.

Розкладання користувачем музики по поличкам, не завжди є хорошою спробою, ось як це виглядає, спочатку жанр, а потім піджанр далі субкультура, а далі лейбл, що призводить до швидкої плутанини, а якщо це вручну сортувати то ще простіше понабирати помилок. Униканням таких проблем можна зробити так, Необхідно скористуватися принципом упорядкування даних, наприклад усунути повтори, уніфікувати дати виходу пісень і правильно підписати жанри. У такий спосіб роботи буде видно більш точні рекомендації та аналітику того, що люди люблять слухати.

Треба зазначити що з огляду різноманітних тем, варто приділити увагу що системою музики використовують і приватно, так і професійно. Можна привести приклад, вчителі музики можуть на уроках включати пісні для поглиблення навчання, або студенти та дослідники часто використовують твори як приклади або натхнення, а також для академічних і творчих занять. Діджеї також дуже часто користуються такими системами для обтирання потрібних творів, а також організатори культурних заходів та бібліотекарів за для привертання уваги публіки.

З професійної сфери там ще все складніше, бо музичні компанії почали часто використовувати код (ISRC), щоб мати можливість відстежувати пісні, а так само вони зберігають окремі БД з додатковою інформацією та контактами, з особливими обмеженнями, наприклад не дозволяючи дізнатися який темп і тональність та інструменти використовуються в даному творі.

А сучасні колектори не лише записують назву та виконавця. Але також в базі зберігаються удари за хвилину, бо діджею необхідно в короткий термін підібрати треки зі схожим треком для програвання на публіку. Вони ж, у свою чергу, додають тональність, щоб пісні зводилися без дисонансу. Окремо зберігаючи код ISRC, спрощуючи реєстрацію авторських прав. Щоб система була точнішою, метадані повинні бути багатшими за обсягом, у такому випадку у звітах буде менша кількість випадкових помилок.

А прогнозування експертів показує три головні тренди [2]. Перший це розумні алгоритми настроювання настрою протягом дня, програма аналізує, що користувач слухав, наприклад пропонує бадьорий плейлист, якщо бачить, що користувач зайшов з фітнес-браслетом. Другий є просторовим звуком, студії вже зараз випускають у реліз Dolby Atmos, а за кілька років це взагалі стане стандартом. Третій це прослуховування через розумні колонки та мультимедійні системи у машині. За допомогою Alexa, Google Home, Apple CarPlay вони автоматично підтягують плейлисти з «хмари», але якщо зв'язок зник, музика обривається. Власники далекобоїв та дачники часто завантажують треки в пам'ять пристрою, щоб нічого не залежало від мобільного інтернету. У такому випадку тут необхідна зручна локальна база, яка працюватиме за голосовою командою. Створення програмного забезпечення для зберігання, організації та управління інформацією про музичні твори. Системі потрібно відповідати поширеним користувачами вимогам: зберігання назв пісень, інформації про виконавця, року випуску, жанру, країни походження та альбому, а також забезпечення прямого посилання на твори.

Технологія програмної реалізації системи базується таким чином: вибрана мова програмування Python, вона є практичною та простою для створення логіки додатків; бібліотека Tkinter для розробки графічних інтерфейсів, та SQL Server, надійна реляційна система керування базами даних [1]. Встановлення з базою данних з’єднання буде проходити через драйвер pyodbc, який легко керує SQL-запитами та зв'язками між таблицями та сутностями.

Вибираючи ці технології в роботі не тільки враховувалася їхня популярність, а й у зв'язку з їх зручністю поєднання та меншою кількістю труднощів при налаштуванні. Перспектив у Python багато, а також і самих рішень, які можуть у майбутньому стати в нагоді для створення значно складніших завдань, наприклад можна включити обробку звуку та побудову графіків. З приводу Tkinter, він хоч і не модний зараз, зате простий у виконанні, ідеально підходить для створення простих вікон, і що має важливу роль, має можливість працювати на всіх комп'ютерах. А для зберігання даних підходить SQL Server 2022, завдяки підтримці звичайних таблиць, так само можна зберігати нестандартні дані.

А сам користувач отримує звичайний і простий консольний додаток на Windows, в ньому можна шукати пісні, додавати нові, редагувати і дивитися список і сортувати за потрібними характеристиками. Також у програмі є поділ прав за ролями.

1.2 Виявлення та вирішення проблем

Щоб проаналізувати потреби різних груп користувачів, треба зробити дослідження, та врахувати усі потреби різних користувачів, таких як досвідчених експертів, меломанів а також простих людей. Також виявити популярні проблеми, з котрими стискаються усі види користувачів під час користуванням інших музичних платформ.

По-перше, це відсутність централізованого способу зберігання данних . Багато користувачів зазвичай стискаються с такими труднощами, як зберігання пісень в окремих файлах, текстових файлах, нотатках або у вигляді паперових списків. При цьому користувачі мають справу с декількома джерел сховищ що утворює ускладнений пошук потрібної інформації, а й робить дублювання, а також призводить до утрати інформації та плутанини. Щоб вирішити цю проблему треба створити централізовану систему, котра буде вирішувати труднощі зі сховищем і надаватиме лише цільну базу музичних творів [2].

Також слід відмітити відсутність ролей для користувачів. На даний момент існує мало додатків, котрі передбачають різні права доступу, зазвичай у більшості простих системах користувачі мають однакові права. Що призводить до ризику випадкового видалення або зміни важливої інформації, особливо це видно коли системою користуються кілька людей одночасно. Вирішення цієї проблеми є реалізація чіткої ієрархії прав доступу, таку як адміністратор, користувач та гість. Система таким способом зможе дозволити адаптувати інтерфейс таким чином, щоб розділити права доступу під кожного користувача [2].

Через зростання великого обсягу пісень зараз користувачу надзвичайно складно швидко знайти потрібну музику. Зараз не завжди ефективний традиційний пошук по окремим параметрам. Користувачів такий спосіб не задовольняє їх вимогам. Вирішення проблеми є реалізація багаторівневої системи пошуку данних, яка буде враховувати декілька критеріїв, таку як, назву, жанр, рік тощо. Таким чином користувач зможе формувати запити під конкретні вимоги.

Відсутність інтеграції із зовнішніми ресурсами – дуже важливий критерій, який необхідно врахувати при створені програмних систем. Як показує практика, багато користувачів люблять слухати музику онлайн. Тому було б дуже зручно, якби пісню можна було безпосередньо пов’язати з YouTube або іншими сервісами. Це дозволило б уникнути трудомісткого ручного пошуку.

Незручний інтерфейс користувача може стати перепоною для розповсюдження програмного забезпечення. Існуючі бази даних, СУБД, утилити, середовища для управління базами даних (наприклад, SQL Server Management Studio) не підходять для звичайних користувачів. Вони вимагають спеціальних знань. Тому потрібен простий та зручний інтерфейс з кнопками, полями та вікнами пошуку.

Пісні повинні бути пов’язані з альбомами, а пісні – з виконавцями. Це дозволяє структуровано зберігати інформацію та уникати дублікатів. У багатьох системах ці зв’язки або не реалізовані, або реалізовані неправильно.

Відсутність історії та підрахунку прослуховувань не дозволяє відслідковувати популярність та вести статистику. У багатьох простих програмах, особливо локальних, взагалі немає жодної історії того, що людина слухала. Адже іноді хочеться повернутися до якогось треку, який сподобався, але вже й не згадати його назву. Також ніде не видно, що слухаєш найчастіше, і насправді ніякої аналітики немає. Це все робить використання таких програм не дуже зручним. Було б добре, якби система могла хоча б зберігати час останнього прослуховування та скільки разів програвався той чи інший трек. Тоді можна було б, наприклад, робити свої улюблені добірки на кшталт «Часто слухане за тиждень» або «Недавні треки», щоб не шукати все вручну.

Запропонована система вирішуватиме указані проблеми: реалізація функцій авторизації та ролей, зручний інтерфейс Tkinter [1], система пошуку та фільтрації, можливість додавання, редагування та видалення пісень, відкриття посилань та робота з SQL Server.

1.3 Постановка задачі

Задачею роботи є створення зручного настільного застосунку для обліку музичних творів, який буде не просто ще однією базою даних, а стане справжнім помічником у впорядкуванні музичної інформації. Мова йде не тільки про технічне збереження записів, а й про створення дружнього інтерфейсу, який буде зрозумілим і простим для найширшого кола користувачів: студентів, викладачів, працівників музичних шкіл, організаторів заходів, колекціонерів, меломанів та навіть тих, хто просто любить слухати музику вдома.

Враховуючи, що багато сучасних сервісів або потребують постійного підключення до Інтернету, або мають обмежений безкоштовний функціонал, виникає потреба у створенні локального рішення, яке буде працювати офлайн, без зайвих складнощів, а головне — швидко, стабільно та безкоштовно. Також не кожен користувач має досвід роботи з SQL чи іншими складними системами, тому необхідно, щоб інтерфейс був інтуїтивно зрозумілим і не викликав страху навіть у новачків.

Для досягнення цієї мети були сформульовані конкретні задачі:

* розробити структуру бази даних. Передбачено створення таблиць для пісень, виконавців, альбомів, студій звукозапису та користувачів. Кожен елемент повинен мати логічний зв'язок із іншими наприклад, пісня має бути прив’язана до альбому, а альбом — до виконавця. Завдяки зовнішнім ключам база буде надійною та впорядкованою;
* реалізувати віконний інтерфейс за допомогою бібліотеки Tkinter. Він повинен містити основне вікно з таблицею, поля для введення інформації, кнопки додавання, редагування та видалення записів, а також повідомлення про помилки або підтвердження дій. Інтерфейс має бути приємним візуально та не вимагати спеціальних навичок для роботи;
* впровадити систему ролей користувачів. Для забезпечення безпеки й обмеження доступу до певних функцій буде реалізовано три типи ролей: адміністратор, користувач та гість. Адміністратор має доступ до всіх можливостей, користувач до базових функцій, а гість зможе лише переглядати інформацію;
* створити зручну систему пошуку та сортування. Важливо, щоб користувач міг знайти потрібну пісню за назвою, виконавцем, роком або жанром. Також бажано передбачити сортування за алфавітом, датою додавання чи країною;
* передбачити можливість зберігати посилання на зовнішні ресурси, наприклад на відео на YouTube. Подвійне натискання на рядок у таблиці повинно відкривати відповідне посилання в браузері;
* здійснити з'єднання з базою даних SQL Server через модуль pyodbc. Програма повинна підтримувати основні запити на додавання, оновлення, видалення та вибір даних, забезпечуючи при цьому надійність збереження інформації;
* реалізувати механізм збереження історії дій або прослуховувань. Корисно, якщо буде видно, які пісні відкривалися найчастіше, коли востаннє вони були відтворені, та на основі цього можна буде формувати плейлисти — наприклад, «Улюблене за тиждень» або «Нещодавно слухане»;
* забезпечити можливість резервного копіювання бази даних. Це дозволить зберігати інформацію на зовнішніх носіях або переносити її на інші пристрої без втрати даних. Для цього потрібно додати окрему кнопку або функцію, яка буде створювати файл резервної копії;
* продумати розширення функціоналу на майбутнє. Наприклад, у подальшому можна буде додати підтримку обкладинок альбомів, створення звітів або експорт даних у форматах Excel або PDF. Це не є обов’язковим зараз, але залишає можливість для розвитку застосунку надалі.

У сукупності все повинно вийде як єдина система, у свою чергу повинна бути зрозуміла для користувача, який не сильно розуміється на техніці. Найголовніше, щоб ПЗ працювало стабільно і без проблем, а так само логічно і дійсно мало користь для людей у роботі з музичними даними.

**2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ**

2.1 Загальні характеристики програмної системи

Розроблене програмне забезпечення є настільним додатком на базі Windows. Система базується на Python, використовує бібліотеку Tkinter для створення графічного інтерфейсу та Microsoft SQL Server як основну платформу зберігання даних [6, 7]. Підключення до бази даних встановлюється через драйвер pyodbc.

Основною метою є дати простий та інтуїтивно зрозумілий інтерфейс для використання, це в свою чергу дозволить користувачам мати можливість фільтрувати, додавати, редагувати та отримувати відкритий доступ через гіперпосилання YouTube [8].

2.2 Класифікація користувачів та права доступу

Програмна система підтримує модель доступу на основі ролей та пропонує три типи користувачів, розмежування ролей допомагає захистити базу даних від випадкового або зловмисного втручання:

* адміністратор: має повний доступ до всіх функцій системи: додавання, редагування, видалення, перегляд записів та керування обліковими записами користувачів;
* зареєстровані користувачі: можуть додавати та редагувати записи, але не видаляти їх;
* гості: мають право лише переглядати записи, але не вносити зміни.

2.3 Функціональні вимоги

Системні функції включають в себе такі вимоги:

Функціональні вимоги включають такі системні функції:

* авторизація користувача за допомогою логіна та пароля (час виконання — до 0.5 с);
* визначення ролі користувача після входу в систему (до 0.2 сек);
* відображення таблиці з інформацією про пісні, виконавців, альбоми, роки, країни та посилань (до 2 сек при 1000 записів);
* пошук за різними критеріями: назва пісні, виконавець, альбом (до 1 сек);
* додавання нових пісень та їх зв'язування з виконавцями та альбомами (до 2 с);
* редагування даних для існуючих пісень, виконавців та альбомів (до 1.5 сек);
* видалення записів (доступно лише для адміністратора, до 1 сек);
* відкриття посилань на пісні в браузері (миттєво через браузер);
* сортувати дані в різних стовпцях (до 1 сек);
* зберігати зміни в базі даних без помилок (до 1 сек на операцію).

2.4 Нефункціональні вимоги

Система також повинна відповідати наступним нефункціональним характеристикам:

* простий інтерфейс для користувача: графічний інтерфейс має бути інтуїтивно зрозумілим та зручним для користувача і не вимагати спеціального навчання. А також повинен мати змогу виконати основну дію у 1–2 кліки, без потреби у додатковому навчанні;
* надійність: усі операції з даними повинні включати перевірку правильності введених даних та повідомлення про помилки. У разі помилки з'являється відповідне повідомлення менше ніж за 0.5 с;
* швидкість роботи: реакція системи на введення користувача має бути якомога коротшою (з максимальною затримкою в одну секунду);
* масштабованість. Система повинна мати можливість керувати великою кількістю обсягу до 10 000 пісень без ніякого погіршення продуктивності (допустиме зниження не більше ніж на 20%);
* модифікація. Програмне забезпечення має бути достатньо гнучким, щоб забезпечити майбутні розширення (поля, кнопки, додаткові типи користувачів тощо). У майбутньому можна реалізувати обмеження кількості спроб входу (наприклад, не більше 5 спроб за 10 хвилин).

2.5 Технічні вимоги до середовища виконання

Операційна система: Windows 10.

Мова програмування: Python 3.9.

Бібліотеки: Tkinter, pyodbc.

База даних: Microsoft SQL Server (2017 або версії по новіші).

Драйвер підключення: ODBC 17 для SQL Server.

**3. АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

3.1 - UML проєктування ПЗ

Use Case діаграма продумувалася так щоб побачити як користувачі зможуть користуватися програмою. У самій програмі є типи ролей зображених на (див. рис. 3.1) по котрим можна відстежити права доступу до системи гість, зареєстрований користувач та адміністратор. Це все враховувалось щоб надати системі безпеку і щоб користувачі не робили хаос:

* гості можуть лише переглядати пісні, шукати та переглядати посилання на YouTube;
* зареєстровані користувачі мають доступ до розширених функцій, таких як додавання пісень;
* адміністратори мають повний доступ, вони можуть редагувати, додавати або видаляти пісні, виконавців та альбоми, а також переглядати всю базу користувачів.

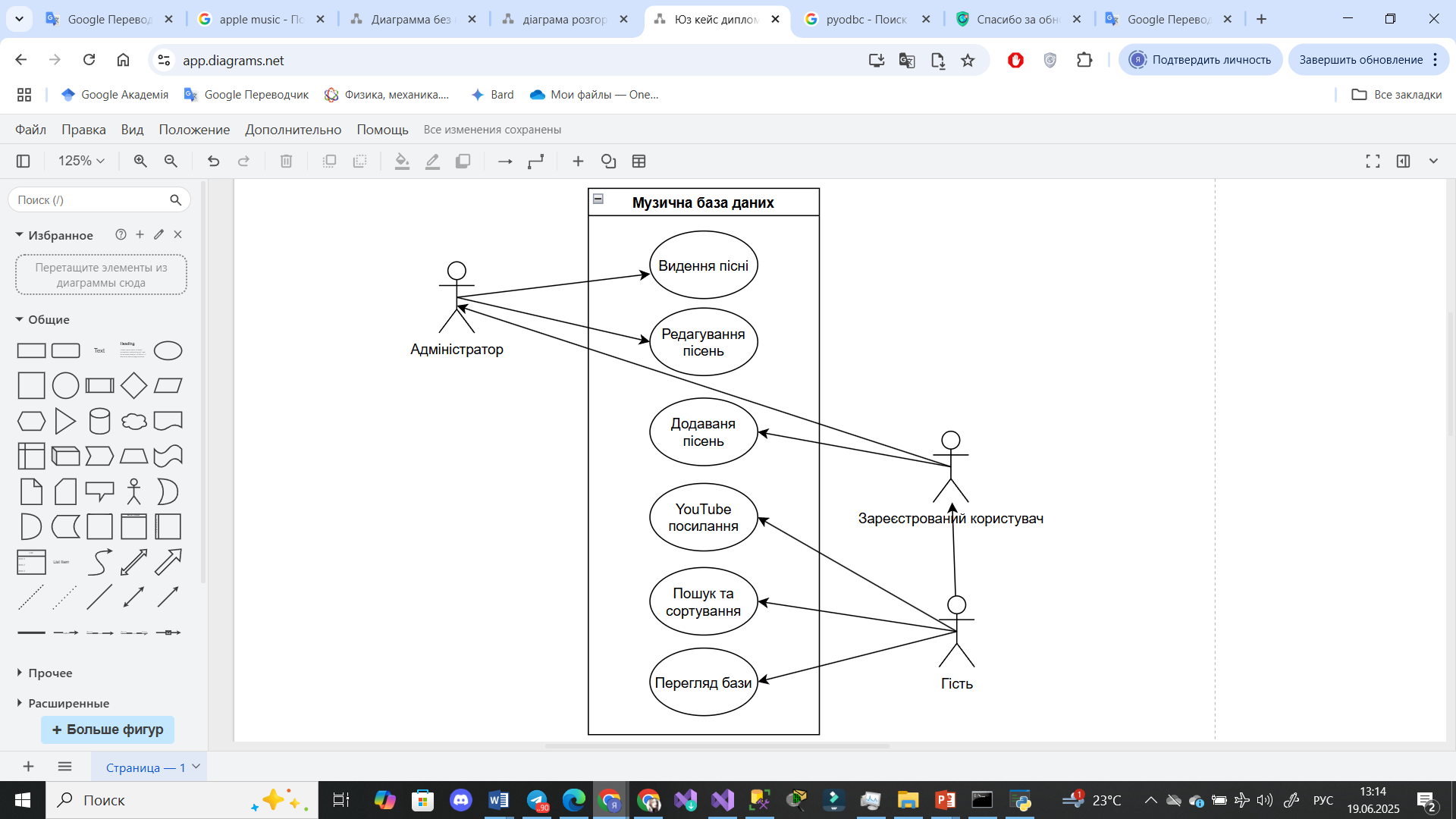


Рисунок 3.1 – Use Case діаграма (рисунок виконано самостійно)

Відображення логіки взаємодії користувача з системою зображено нижче (див.рис. 3.2) а самі головні сутності виглядають так:

* User : тут зберігається ID, логін, пароль та роль;
* Artist: ім’я, країна, жанр;
* Album: назва альбому, рік, ID виконавця, студія;
* Song: жанр, альбом;
* Studio: назва, країна, рік.

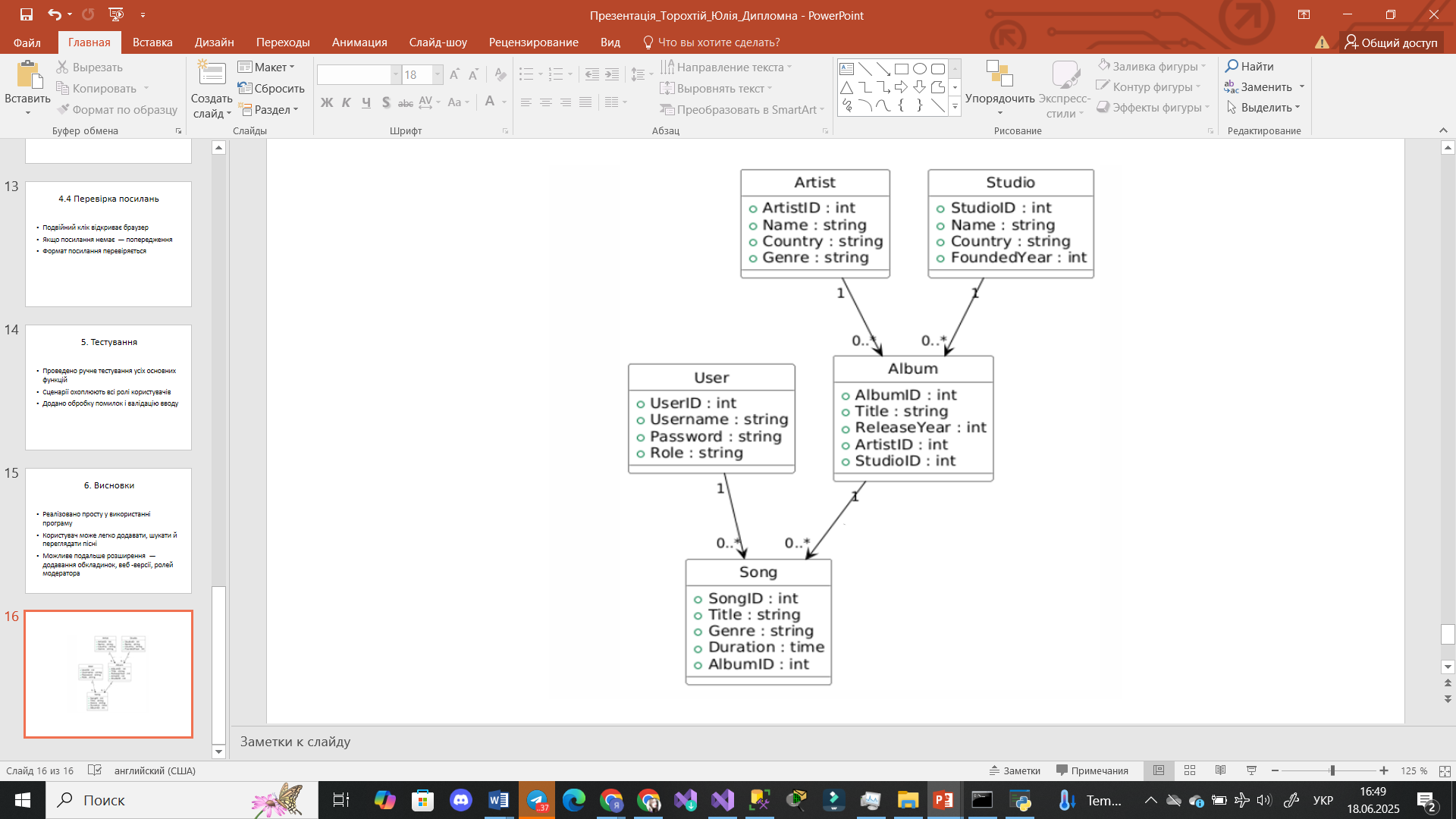


Рисунок 3.2 – Діаграма класів (рисунок виконано самостійно)

Кожна із цих речей пов'язана логічно. Виконавець може створювати кілька альбомів (1 → \*), кожен із яких містить кілька пісень, що належать до одного альбому, і альбом може бути створений у певній студії. Ціль полягає в тому, щоб зробити все логічно, щоб потім було легше зробити через базу.

А на діаграмі активності можна побачити імітування входу до системи зображеної на (див.рис. 3.3). Саму діаграму було умовно розділено за для наглядної демонстрації того що робить користувач при вході у систему, а що система робить.

* користувач вводить свій логін та пароль;
* натискає кнопку «Увійти» ;
* система перевіряє, чи заповнені обов'язкові поля;
* якщо поля порожні, відображається повідомлення про помилку;
* якщо поля заповнені, відбувається SQL-запит для перевірки даних;
* якщо користувача не знайдено, відображається повідомлення про те, що ім'я користувача та пароль неправильні;
* якщо користувача знайдено, система визначає роль та відкриває відповідний інтерфейс користувача.

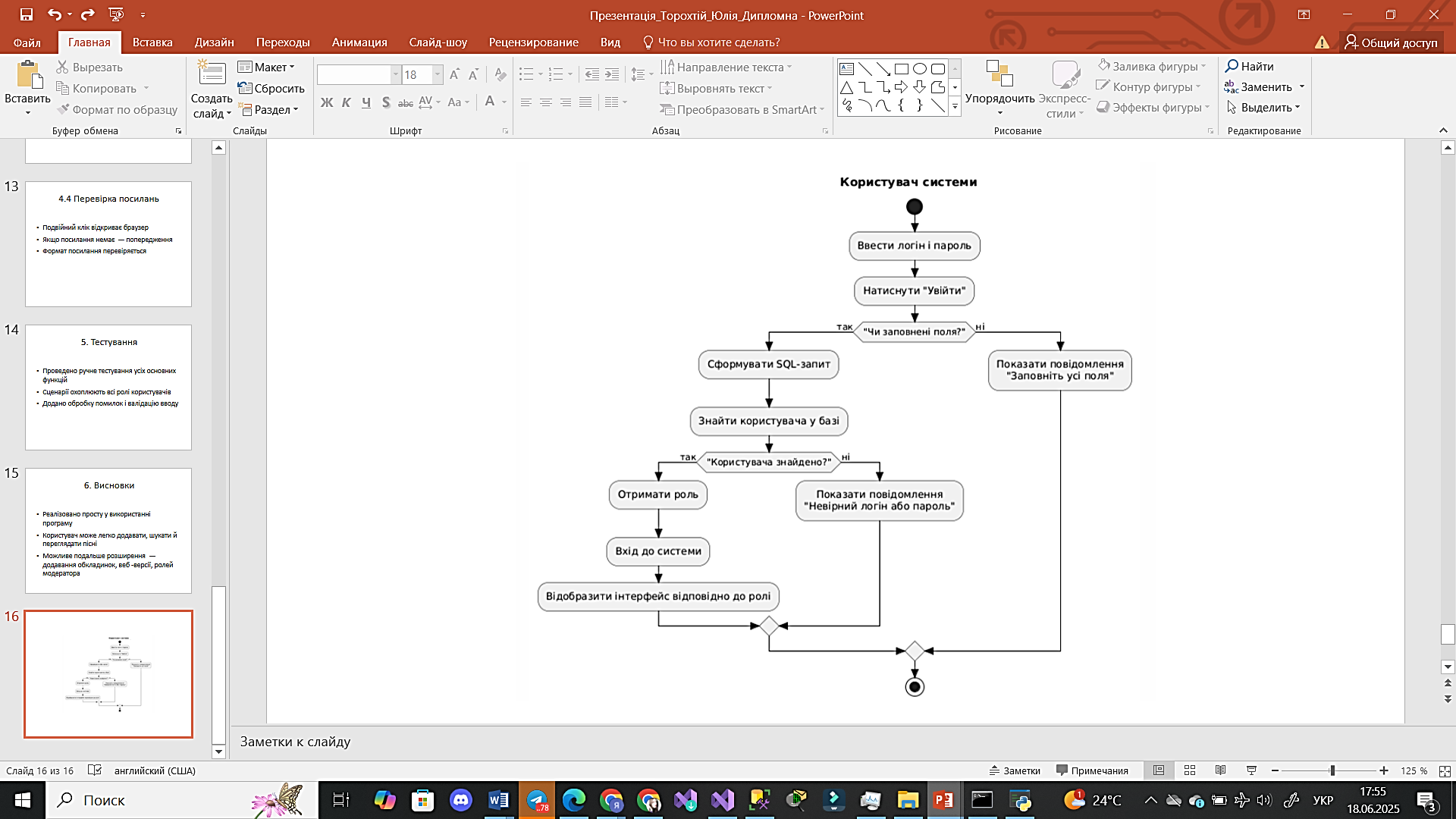


Рисунок 3.3 – Діаграма діяльності для сценарію «Авторизація користувача» (рисунок виконано самостійно)

3.2 Проектування архітектури програмного забезпечення

Архітектуру ПЗ побудовано за класичною моделлю клієнт-сервер, оскільки вона найкраще підходить для таких задач, де графічний інтерфейс відокремлений від бази даних, а дані зберігаються зовнішньо у цьому випадку у SQL Server. Завдяки такому рішенню вдалось досягти чіткого поділу між частиною, з якою працює користувач, і тією, що відповідає за зберігання та обробку інформації. Окрім того, це дозволяє у майбутньому без особливих труднощів перенести базу на інший сервер або масштабувати систему. Застосунок реалізовано мовою Python. Інтерфейс створений за допомогою стандартної бібліотеки Tkinter [1], яка дозволяє швидко створювати вікна, кнопки та інші елементи керування без складних налаштувань, а для підключення до SQL Server використано бібліотеку pyodbc, що працює через драйвер ODBC 17 та забезпечує стабільне з’єднання і пряме виконання SQL-запитів.

Хоча застосунок є монолітним, у ньому чітко простежуються три логічні шари: інтерфейс, логіка та доступ до бази даних. Інтерфейс відповідає за виведення даних і взаємодію з користувачем, логіка реалізує обробку подій, авторизацію, перевірки, а частина доступу до даних виконує з’єднання з базою, формує запити й повертає результати. Такий розподіл дозволяє легше підтримувати проект, а при необхідності змінювати або розширювати окремі частини без впливу на інші. При додаванні нової пісні система спочатку перевіряє, чи існує такий виконавець. Якщо ні, додається новий, після цього перевіряється, чи існує відповідний альбом, і лише після цього додається сама пісня. Усі дії відбуваються послідовно, з обов’язковим контролем на кожному етапі. Збереження в базу виконується тільки після того, як усі кроки пройдено успішно, що дозволяє уникнути часткового внесення даних у разі помилки. У програмі реалізовано авторизацію з трьома типами ролей гість, користувач і адміністратор. Гість має змогу лише переглядати дані, здійснювати пошук і переходити за посиланням, користувач, окрім цього, може ще й додавати нові пісні, а адміністратор отримує повний доступ до всіх дій, зокрема редагування та видалення. В інтерфейсі реалізована функція update\_interface\_by\_role(), яка активує або вимикає певні елементи залежно від поточної ролі, що дозволяє зробити систему безпечнішою та зручною у використанні.

Для уникнення неочікуваних збоїв і зручності у діагностиці всі критичні операції, що звертаються до бази, обгорнуті в конструкцію try/except. Виводить користувачеві не системні помилки, а зрозумілі повідомлення, наприклад, якщо з’єднання з базою не встановлено або запис не знайдено. Середовище розгортання Windows, однак важливо лише щоб на комп’ютері були встановлені Python, SQL Server і драйвер ODBC. Оскільки застосунок не прив’язаний до конкретного пристрою, його можна запускати на будь-якому ПК за наявності правильно налаштованого підключення до бази. Діаграма розгортання навернена (див.рис. 3.4)

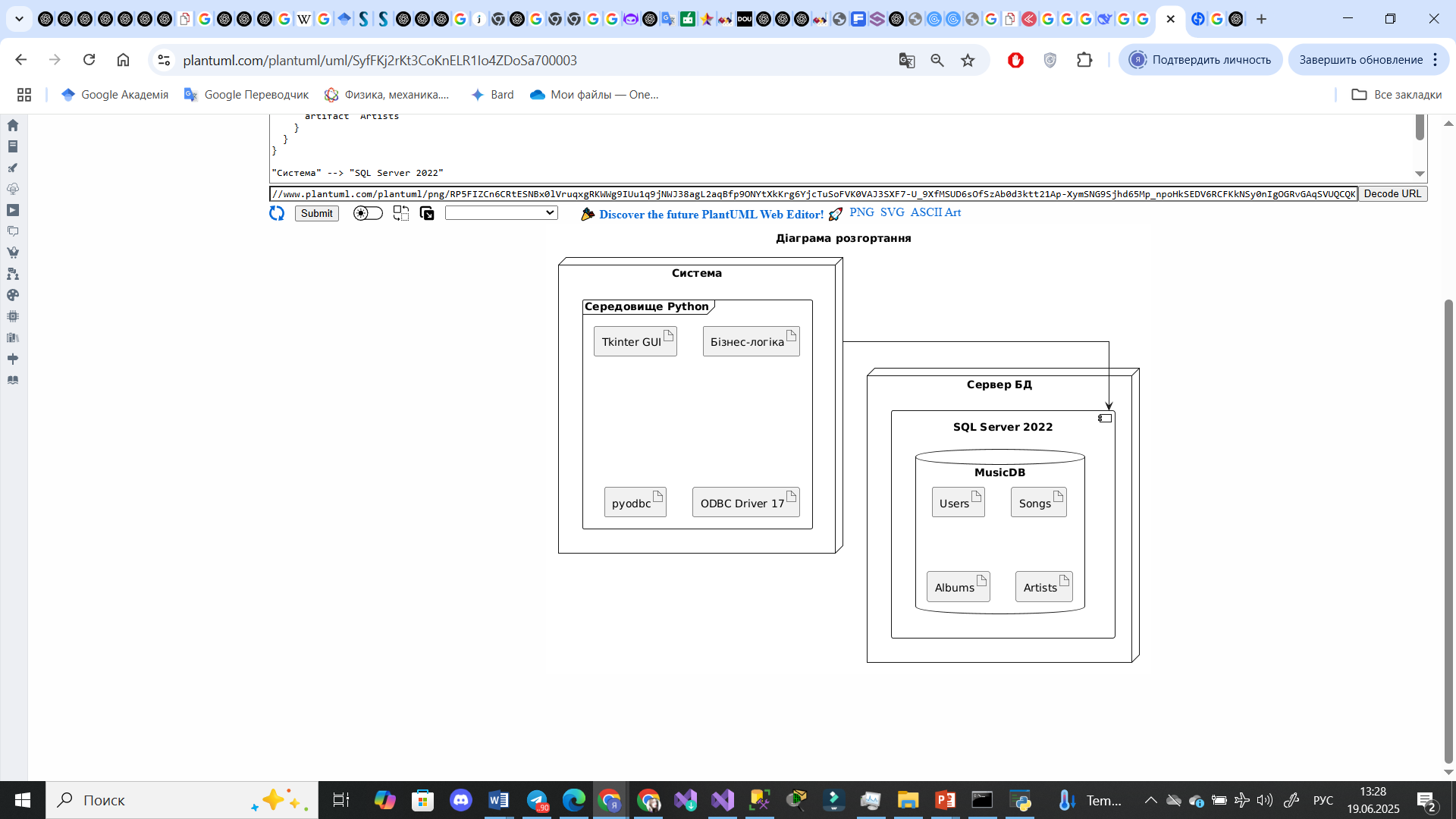


Рисунок 3.4 – Діаграма розгортання (рисунок виконано самостійно)

Щоб програму було легше підтримувати з боку логіки, внутрішню структуру чітко поділено залежно від того, за що кожна її частина відповідає наведено на таблиці 3.1.

Таблиця 3.1 – Шарова архітектура

|  |  |  |
| --- | --- | --- |
| Шар | Відповідає за | Реалізовано |
| Інтерфейс | Вікна, кнопки, поля введення | Tkinter, Treeview |
| Логіка | Дії користувача, авторизація, перевірки | add\_song, edit\_song, update\_interface\_by\_role |
| Доступ до БД | SQL-запити, з’єднання, збереження | pyodbc, get\_connection, load\_data |

Архітектурно застосовано кілька класичних шаблонів проєктування, по суті, структура дотримується підходу MVC інтерфейс, логіка і доступ до даних реалізовані окремо. Template Method реалізується через послідовність дій при додаванні пісні, де порядок завжди один і той самий.

З’єднання з базою реалізовано через get\_connection(), який виступає в ролі Factory Method, а сам факт, що кожна транзакція використовує єдиний екземпляр з’єднання, дозволяє говорити про використання принципу Singleton. SQL-запити винесено в окремі функції і їхня реалізація не змішується з логікою інтерфейсу, що наближає реалізацію до патерна DAO [1, 6]. Окрема увага приділена питанням безпеки, з’єднання з SQL Server шифрується через протокол TCP 1433 за допомогою транспортного рівня TDS що уникає витоку чутливих даних [9].

3.3 Проєктування структури зберігання даних

Основні сутності - «Пісня», «Виконавець», «Альбом», «Студія », «Альбом» «Користувач». Їх роль визначає ключові об’єкти, котрі між собою взаємодіють, зображено на (див.рис. 3.4).

На підставі ER-діаграми побудуємо схему бази даних. Кожна сутність буде відображена на таблицю бази даних, ключові атрибути виступатмуть в ролі первинних ключів. Отримана схема бази даних приведена до 3 нормальної форми, що гарантує відсутність надлишковості даних і відповідність предметній області [10, 11] (див. рис.3.5).

Взявши модель реляції баз даних, на середовищі Microsoft SQL Server, і структурувавши в таблиці Song, Album, Artist, User, отримаємо схему БД, наведену на рис.3.6.

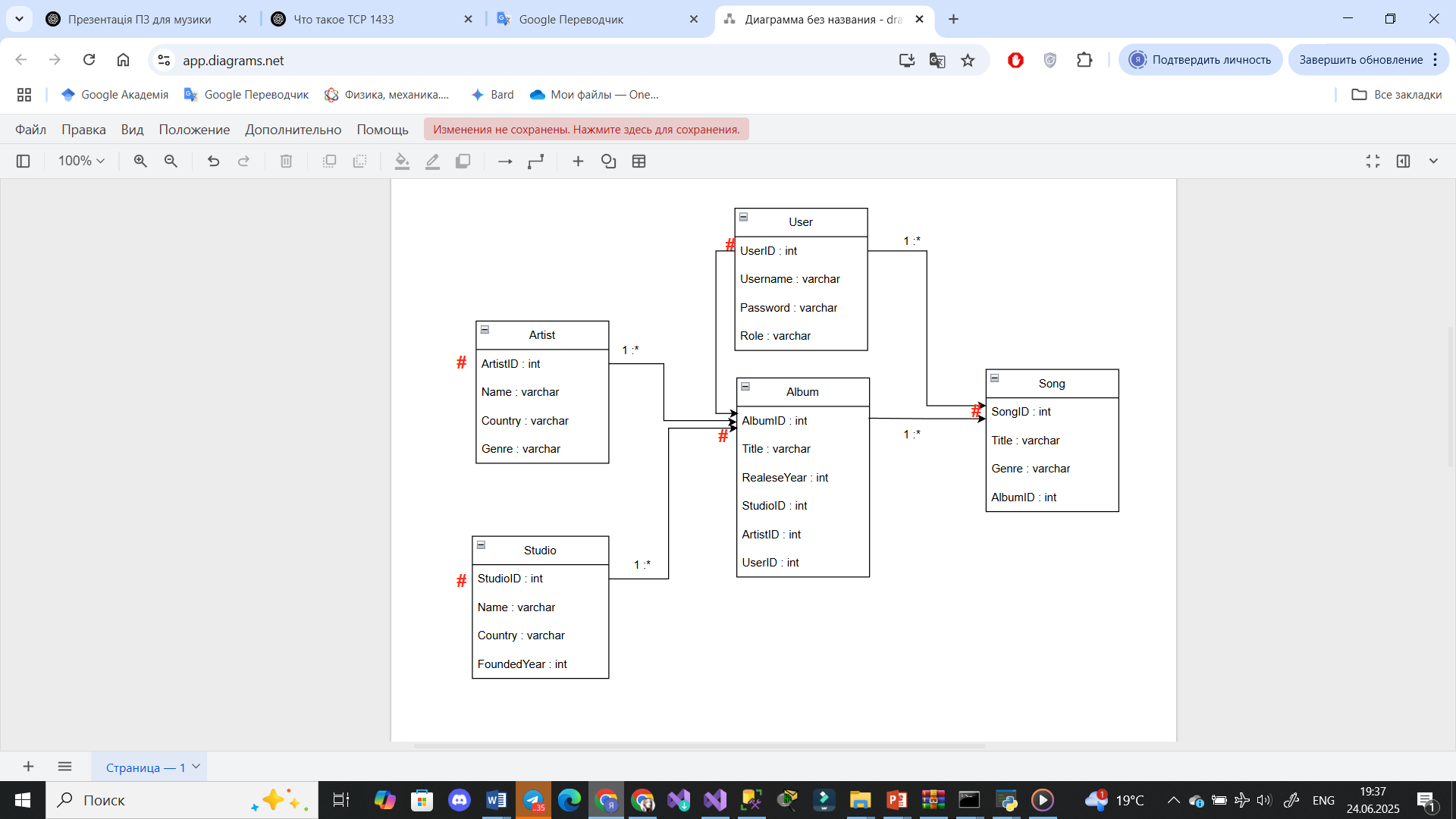


Рисунок 3.5 – ER-діаграма (рисунок виконано самостійно)

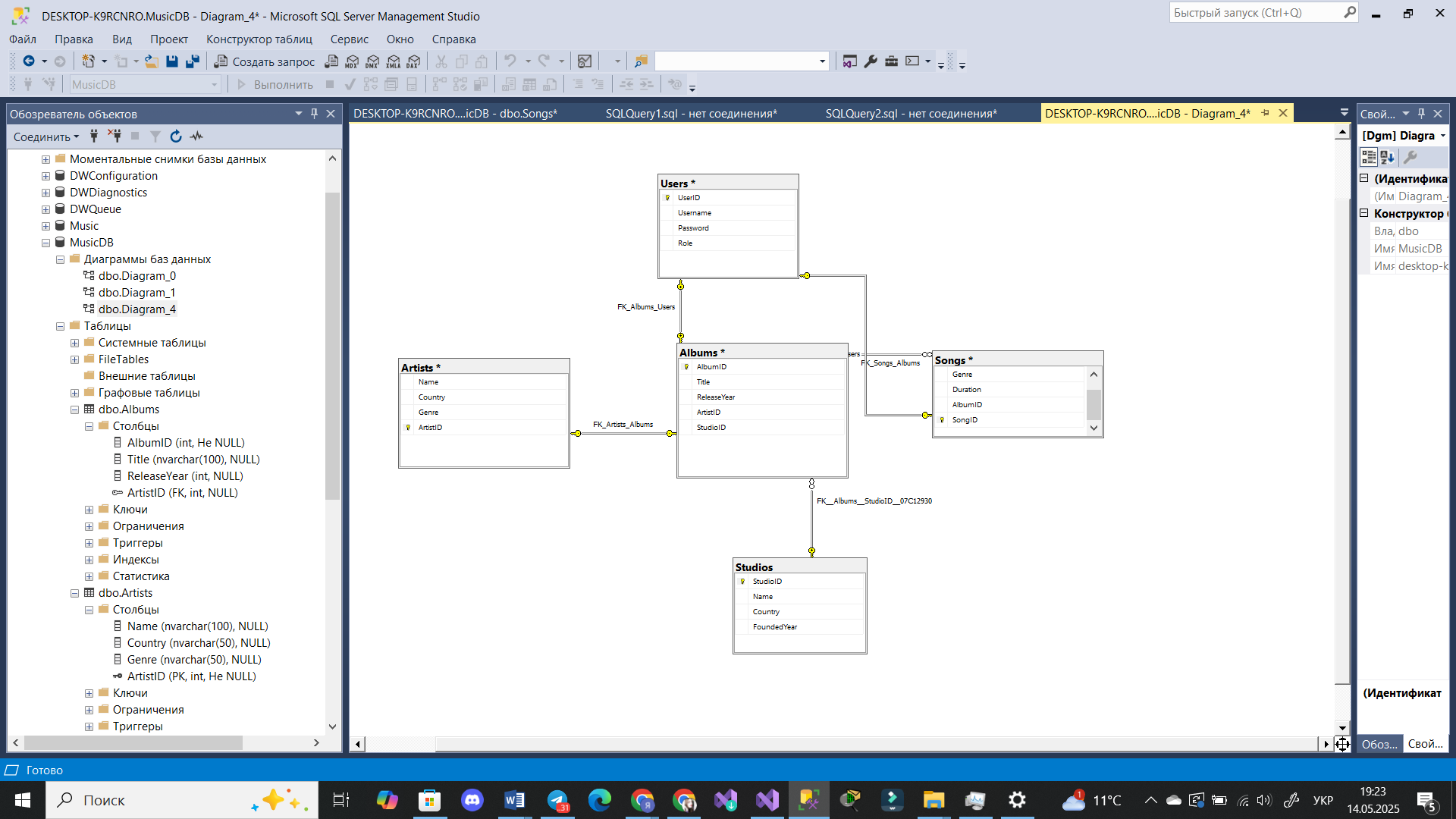


Рисунок 3.6 – Схема бази даних (рисунок виконано самостійно)

Опис структури зав’язків між таблицями в основі реляційної моделі можна побачити (див.рис. 3.6). Тут показано їхні зв’язки між собою, головні таблиці зроблених за допомогою зовнішніх ключів виглядають так:

* User: Зберігає облікові дані користувача;
* Artist: Інформація про виконавця;
* Album: Метадані альбому, включаючи зовнішні ключі для виконавця та студії;
* Song: Список треків, включаючи зовнішні ключі для жанру, тривалості та альбому;
* Studio: Опис студії.

Приклад: Albums. ArtistID пов'язано з Artists.ArtistID (1:n); Albums.StudioID пов'язано зі Studios.StudioID; Songs.AlbumID пов'язано з Albums.AlbumID.

Самі ж таблиці пов’язані ключами зовнішніми, позволячи робити запити реляції і відображати до кожного запису повну інформацію

Наведеним прикладом є ситуація у котрій, мабуть кожен користувач стискався у своїх роботах, та відображення списку пісень, а також даних про виконавців та альбоми, а сам приклад можна побачити нижче, це запит SQL коду:

**SELECT**

**Songs.SongID,**

**Songs.Title,**

**Artists.Name,**

**Artists.Country,**

**Albums.Title AS AlbumTitle,**

**Albums.ReleaseYear,**

**Songs.Link**

**FROM Songs**

**JOIN Albums ON Songs.AlbumID = Albums.AlbumID**

**JOIN Artists ON Albums.ArtistID = Artists.ArtistID;**

І звісно основою функцій для всіх робіт є запит, відображавши таблиці пісень. А у результаті можна побачити результат про пісню (назва, рік, виконавець, країна, альбом та посилання) останнє ж можна відкрити у браузері і послухати пісню з роликом на YouTube зображене це на (див.рис. 3.7).

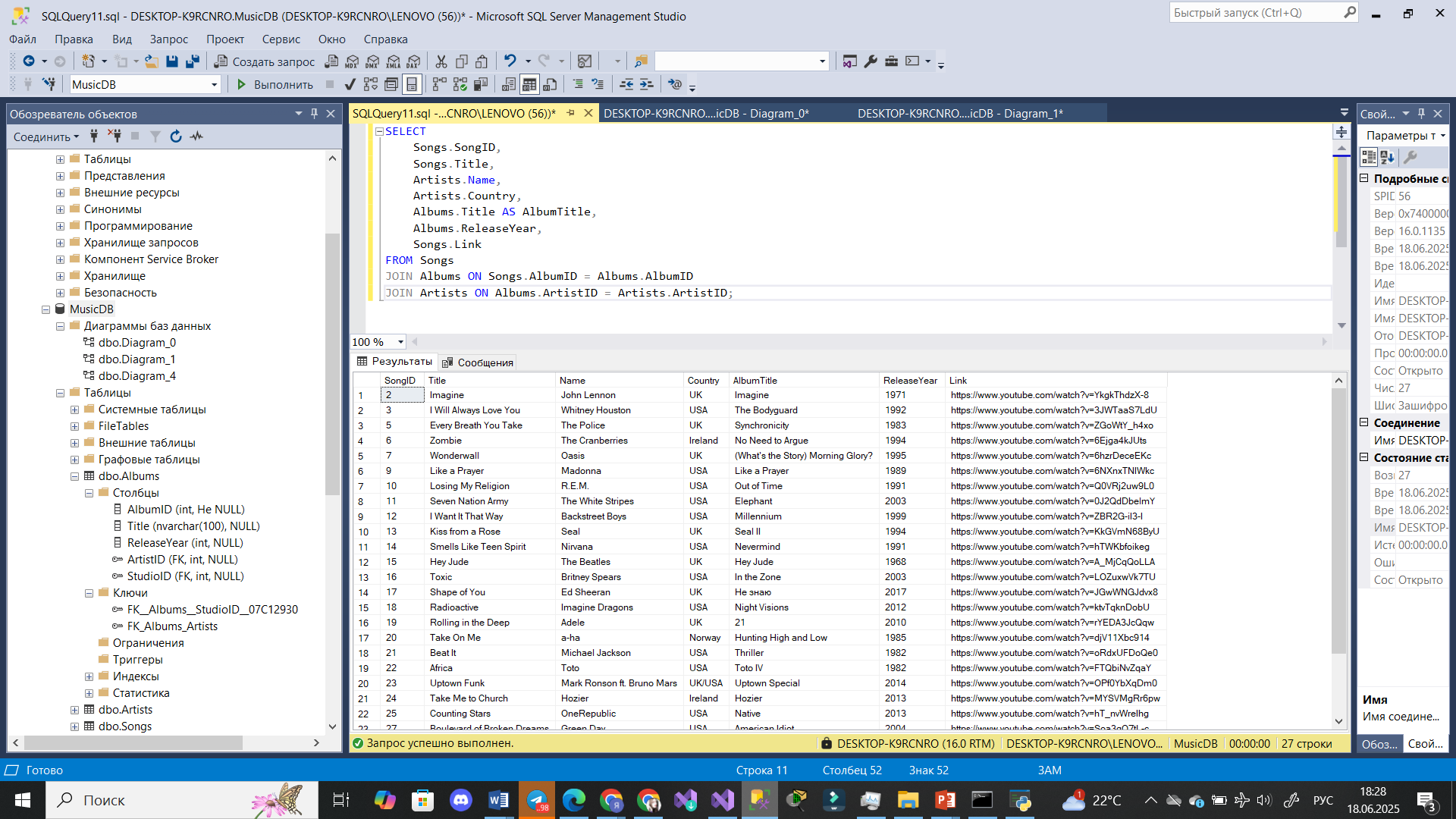


Рисунок 3.7 – Фрагмент SQL-запиту та результат його виконання в базі даних обліку музичних творів (рисунок виконано самостійно)

3.4 Приклади найцікавіших алгоритмів та методів

При розробці цього застосунку, знадобилося зробити багато методів, котрі не тільки роблять стандартні роботи, а й обробляють щось складніше, наприклад автоматичне створення пов’язаних записів. динамічна генерація SQL-запитів та керування доступом на основі ролей. Нижче показані інтересні фрагменти коду.

Перший метод, який наведемо, дозволяє автоматично додавати нові пісні та запит виконавців і альбомів. При додаванні користувачем нової пісні, головне для системи це перевірити чи існує такий самий виконавець, або альбом. Нижче показано фрагмент коду з алгоритмом:

**#виконавець**

**cur.execute("SELECT ArtistID FROM Artists WHERE Name=?", (artist,))**

**row = cur.fetchone()**

**if row:**

**artist\_id = row[0]**

**else:**

**cur.execute(**

**"INSERT INTO Artists (Name, Country) VALUES (?, ?)",**

**(artist, country or None),**

**)**

**conn.commit()**

**artist\_id = cur.execute("SELECT @@IDENTITY").fetchone()[0]**

**#альбом**

**cur.execute("SELECT AlbumID FROM Albums WHERE Title=?", (album,))**

**row = cur.fetchone()**

**if row:**

**album\_id = row[0]**

**else:**

**cur.execute(**

**"INSERT INTO Albums (Title, ArtistID, ReleaseYear) VALUES (?, ?, ?)",**

**(album, artist\_id, year),**

**)**

**conn.commit()**

**album\_id = cur.execute("SELECT @@IDENTITY").fetchone()[0]**

Опишемо функцію динамічного завантаження WHERE. Коли створюється функція «load\_data», при додаванні функції, котра вже є включена до самого тексту. При вводі користувачем запиту пошуку, приходить функція шукаючи пісні за назвою, іменем виконавця або назвою альбому продемонстровано:

**SELECT**

**Songs.SongID, Songs.Title,**

**Artists.Name, Artists.Country,**

**Albums.Title, Albums.ReleaseYear,**

**Songs.Link**

**FROM Songs**

**JOIN Albums ON Songs.AlbumID = Albums.AlbumID**

**JOIN Artists ON Albums.ArtistID = Artists.ArtistID**

**"""**

**params = []**

**if search:**

**query += """**

**WHERE Songs.Title LIKE ?**

**OR Artists.Name LIKE ?**

**OR Albums.Title LIKE ?**

**"""**

**like = f"%{search}%"**

**params.extend([like, like, like])**

На основі прав ролей відбувається керування інтерфейсом. Важливо щоб це було у роботі, адже допомагає відділити простих користувачів від адміністраторів, продемонстровано це таким чином:

**def update\_interface\_by\_role():**

**if current\_user\_role == "Admin":**

**for b in (btn\_add\_song, btn\_delete\_song, btn\_edit\_song):**

**b.config(state=tk.NORMAL)**

**elif current\_user\_role == "User":**

**btn\_add\_song.config(state=tk.NORMAL)**

**btn\_delete\_song.config(state=tk.DISABLED)**

**btn\_edit\_song.config(state=tk.DISABLED)**

**else: #Guest**

**for b in (btn\_add\_song, btn\_delete\_song, btn\_edit\_song):**

**b.config(state=tk.DISABLED)**

Клацання мишкою подвійно для переходу за посиланням для переходу на YouTube. Дуже практична та зручна особливість застосунку, оскільки користувач може одразу послухати та подивитись кліп, не шукаючи її самостійно, сама ж функція зроблена за допомогою open\_link(event) продемонстрований алгоритм:

**def open\_link(event):**

**selected\_item = tree.selection()**

**if selected\_item:**

**values = tree.item(selected\_item, "values")**

**if len(values) >= 7:**

**url = values[6]**

**if url and url.lower().startswith("http"):**

**webbrowser.open\_new\_tab(url)**

Використання комбінацій «гарячих клавіш» дозволяє спростити взаємодію користувача з програмою У інтерфейсі вони дуже потрібні для повноцінного користування, тому було добавлено три клавіші Ctrl+C, Ctrl+V та Ctrl+A, продемонстровано це таким чином:

**widget.bind("<Control-c>", lambda e: e.widget.event\_generate("<<Copy>>"))**

**widget.bind("<Control-v>", lambda e: e.widget.event\_generate("<<Paste>>"))**

**widget.bind("<Control-a>", lambda e: e.widget.event\_generate("<<SelectAll>>"))**

3.5 Опис UI/UX

При створені інтерфейсу було враховано не лише складність та привабливість для всіх, а й найважливішим було те, щоб він міг все стабільно відпрацьовувати, а також щоб був простим і зрозумілим при використанні (див.рис. 3.8).

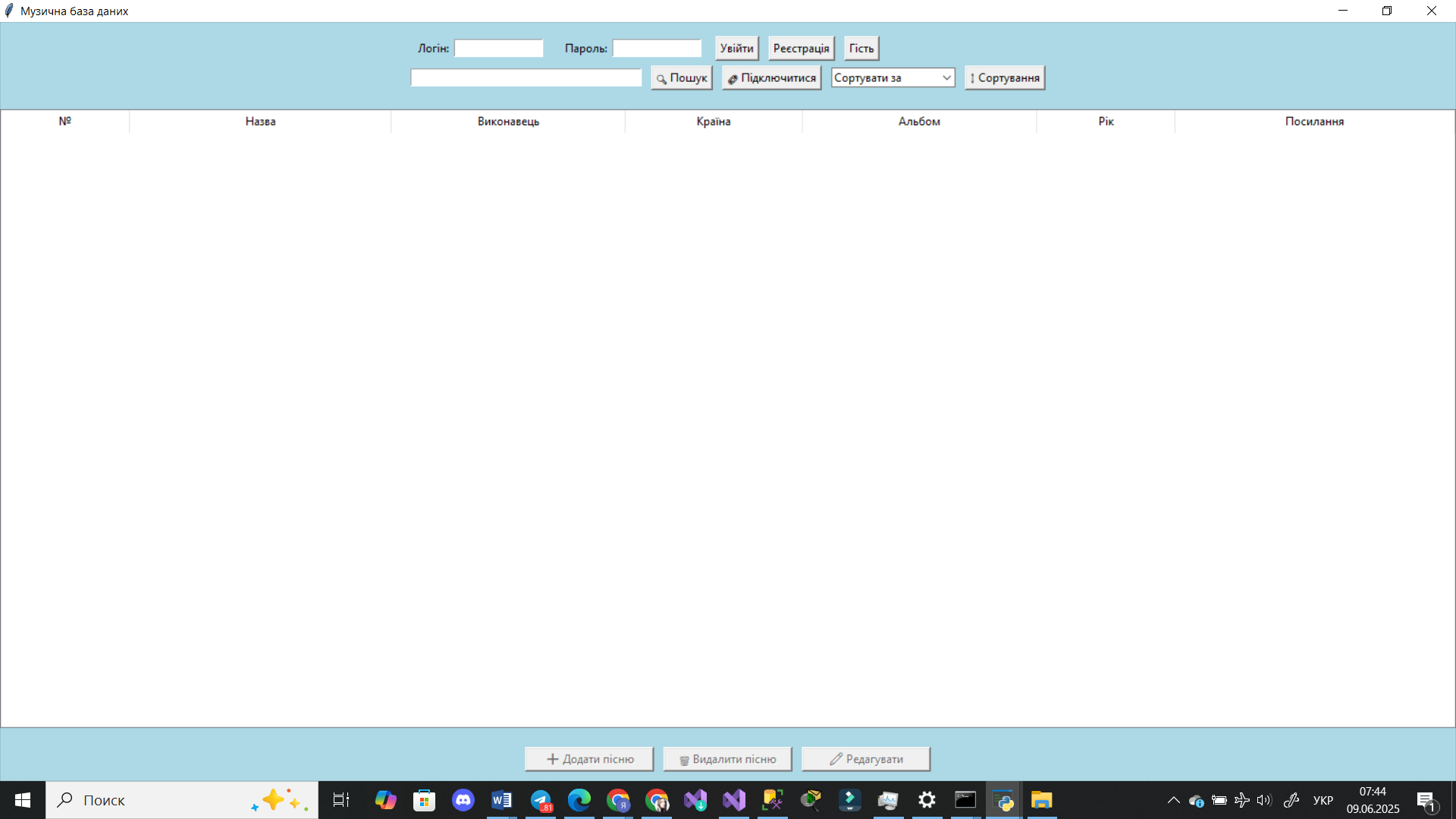


Рисунок 3.8 – Інтерфейс головного меню (рисунок виконано самостійно)

А сам же інтерфейс стандартно виглядає, угорі є вікно для авторизації нових користувачів, а також для входу зареєстрованих користувачів, важливо щоб користувач ввів логін і пароль, і натиснув на кнопку «Увійти». Недалеко розташовано поле для пошуку пісень, за допомогою її можна шукати пісні через назву пісні, виконавця, альбому або жанру. А також є кнопка сортування пісень по назві пісень, виконавцем, альбому, країні та роком випуску.

Самою головною частиною є вікно з таблицею, вона показує всі пісні котрі зберігаються в бібліотеці. Виглядає вона як структура котра містить поля, ID, назва, виконавець, країна, альбом, рік та посилання. Завантаження проходить автоматично з бібліотеки, після запуску застосунку та натискання кнопки «Підключитися» (див.рис. 3.9).

Знизу під таблицею є три кнопки «Додати пісню», «Видалити пісню» та «Редагувати пісню». Щоб скористатися певною кнопкою, у користувача потрібні бути особливі права доступу. Адже гість не зможе скористатися жодною кнопкою, а користувач тільки додавати, а адміністратор може користатися усіма кнопками і видаляти додані та редагувати пісні.

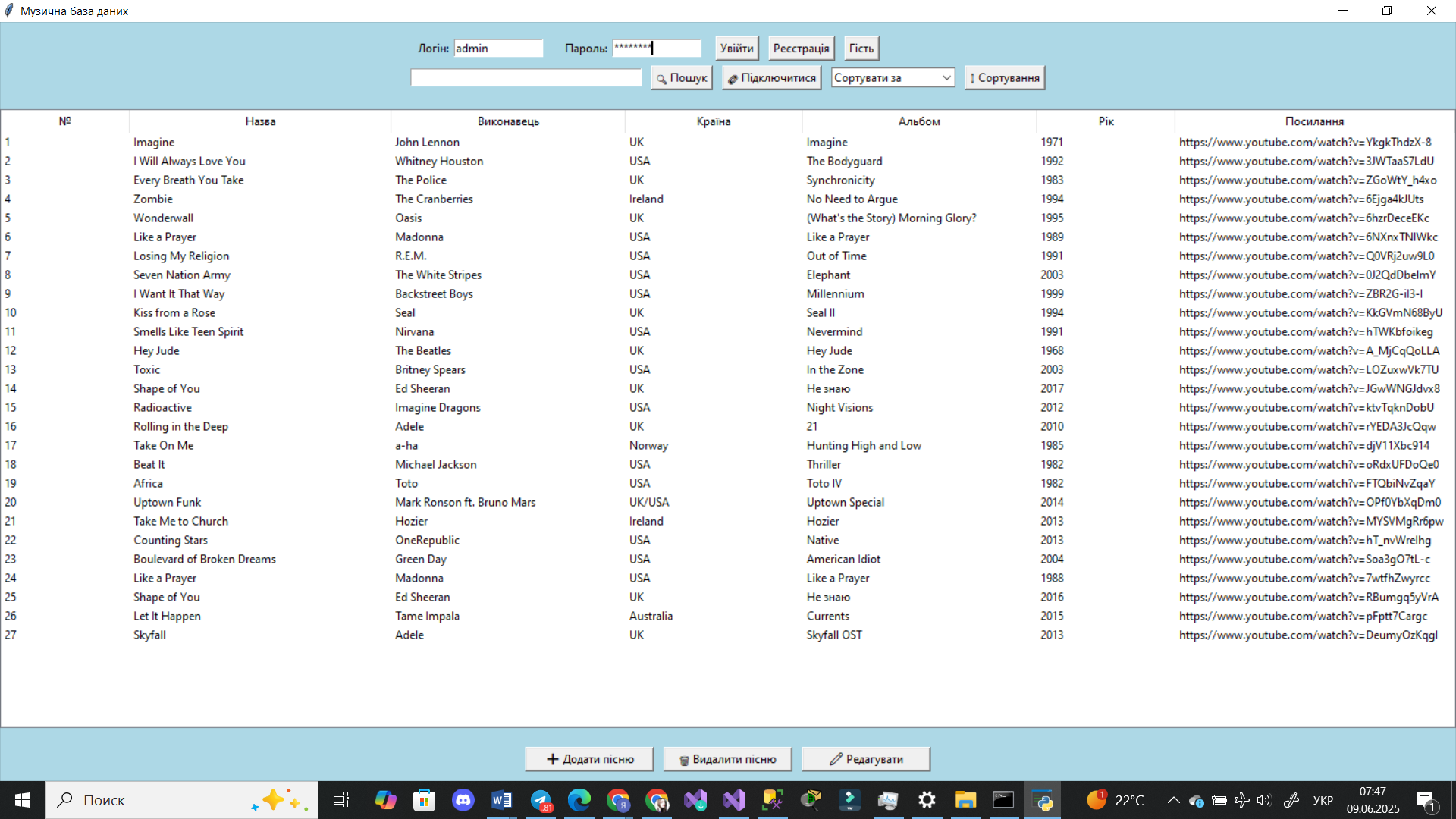


Рисунок 3.9 – Інтерфейс з таблицею відображення пісень (рисунок виконано самостійно)

У цілому робота над інтерфейсом вийшла цілком просто та зручно для користування, адже застосунок зрозумілий та простий і робочий, але застосунок вийшов не таким привабливим, але це не саме головне в застосунку, котрий добре працює, надаючи користувачеві можливість швидко знайти потрібні пісні та відкрити їх за посиланням, а також додати нові.

**4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ**

4.1 Створення автентифікації, авторизації та керуванням ролями

В структурі є трирівнева модель прав котра виглядає так — Admin, User, Guest. Відбувається вона за допомогою таких методів, вводу логіну та паролю, а ідентифікація визначається за парою username–password, а сама перевірка входу користувача проходить у базі даних, а дублювання ніяк не пройде, завдяки безпеці логіки, вона запам’ятовує користувачів. При успішному входженні користувача відбувається асинхронне оновлення змінної current\_user\_role, а потім починає працювати функція під назвою спостерігач update\_interface\_by\_role() продемонстровано таким чином.

**#роль при вході**

**auth\_default\_role = "Guest"**

**current\_user\_role = auth\_default\_role**

**#оновлення інтерфейса по ролі**

**def set\_user\_role(role: str) -> None:**

**global current\_user\_role**

**current\_user\_role = role**

**update\_interface\_by\_role()**

Рисунок 4.1 – Фрагмент надання ролі (рисунок виконано самостійно)

При вході у систему нового зареєстрованого користувача виглядає таким чином (див.рис. 4.2)

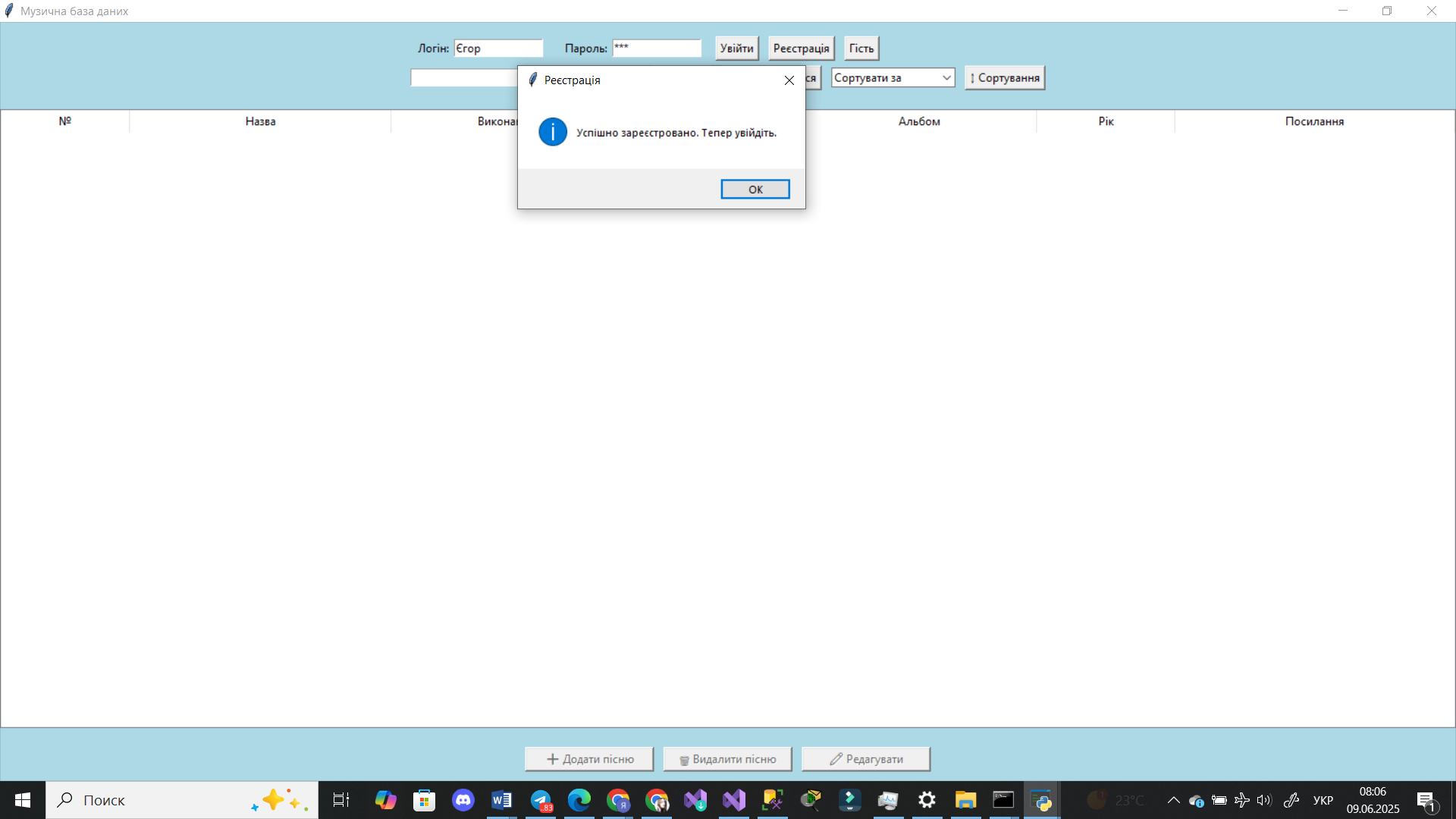


Рисунок 4.2 – Успішна реєстрація нового користувача рисунок виконано самостійно)

А щоб зареєстрований користувач ввійшов треба ввести логін та пароль. Якщо користувач введе правильно усе, то в базі висвітиться що вхід був успішним (див.рис. 4.3), а якщо ні, то помилку з невірним логіном або паролем (див.рис. 4.4).

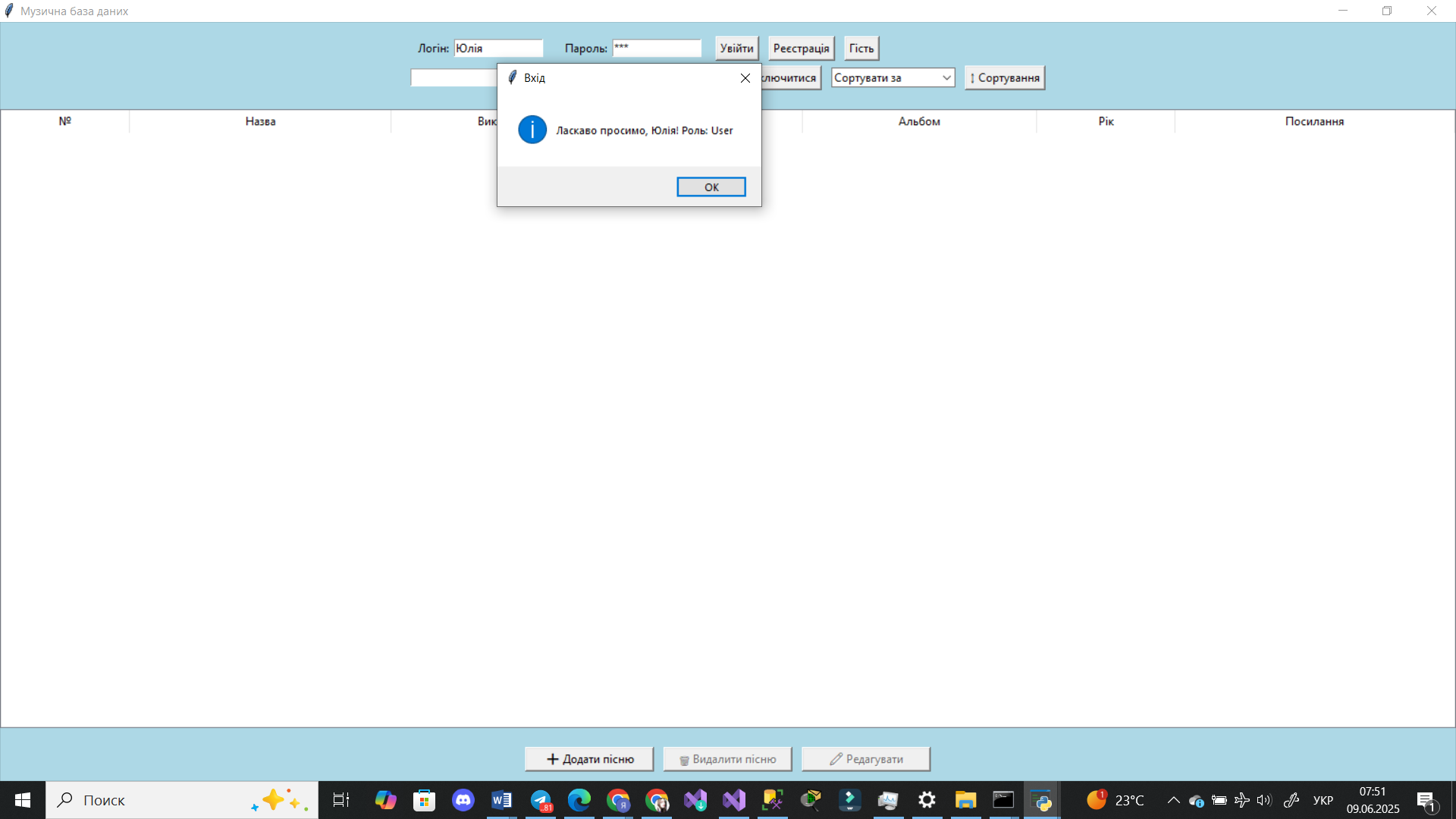


Рисунок 4.3 – Введення логіну та паролю і успішний вхід користувачем (рисунок виконано самостійно)

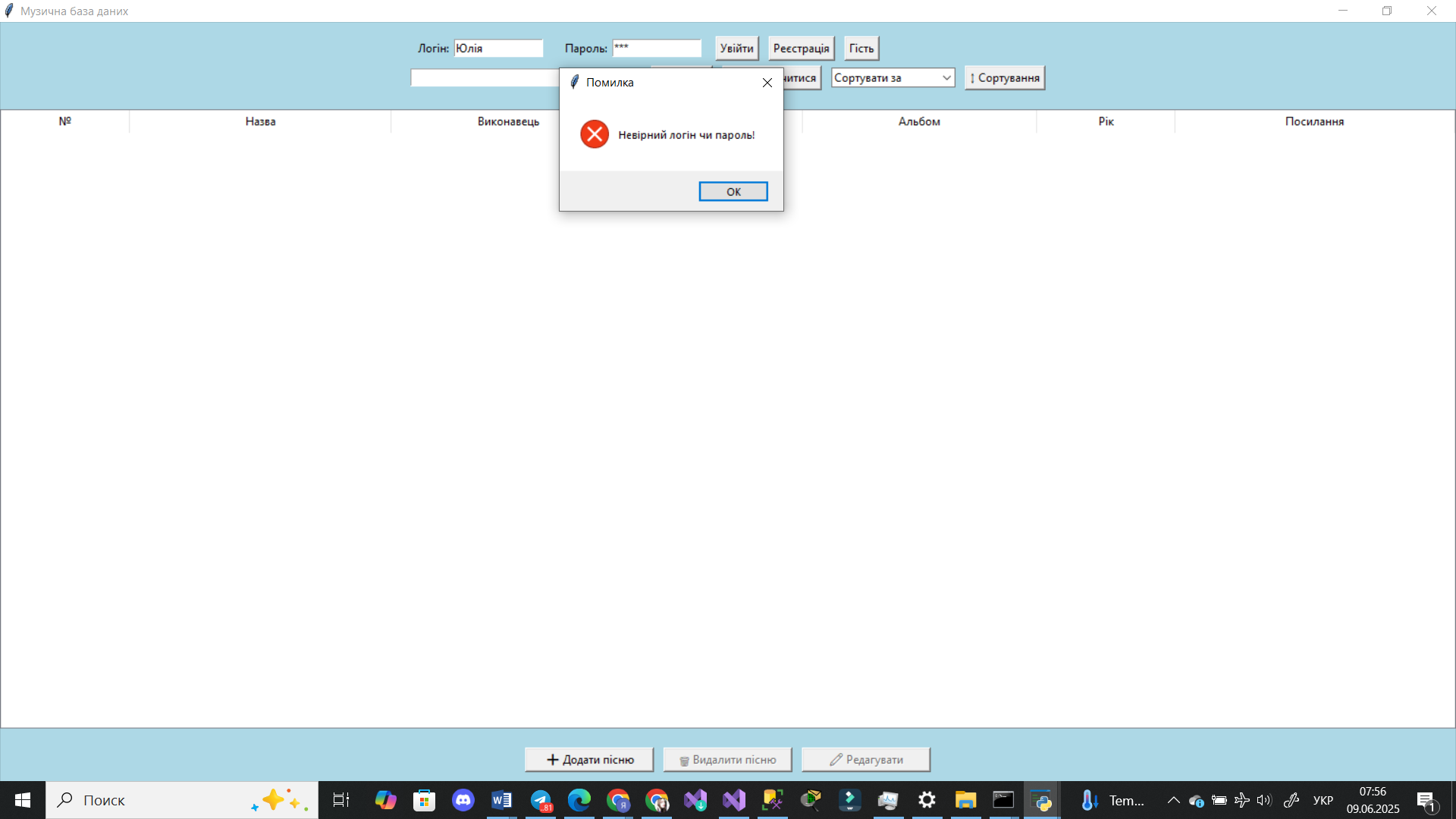


Рисунок 4.4 – Помилки при вході (рисунок виконано самостійно)

А кінцева функція повністю робить блокування усіх нижніх кнопок, тільки при тому випадку для груп із нижчими повноваженнями, як приклад у гостьовому режимі будь-які дії за для модифікації заблоковані повністю, а також допомагає зменшити ризики неавторизованих дій

Також зауважу ще раз, що при вході на роль користувача розширяються можливості, такі як додавання пісень. А сама роль адміністратора взагалі надає змогу користуватися усіма трьома кнопками такими як додавання, видалення та редагування.(див.рис. 4.5)

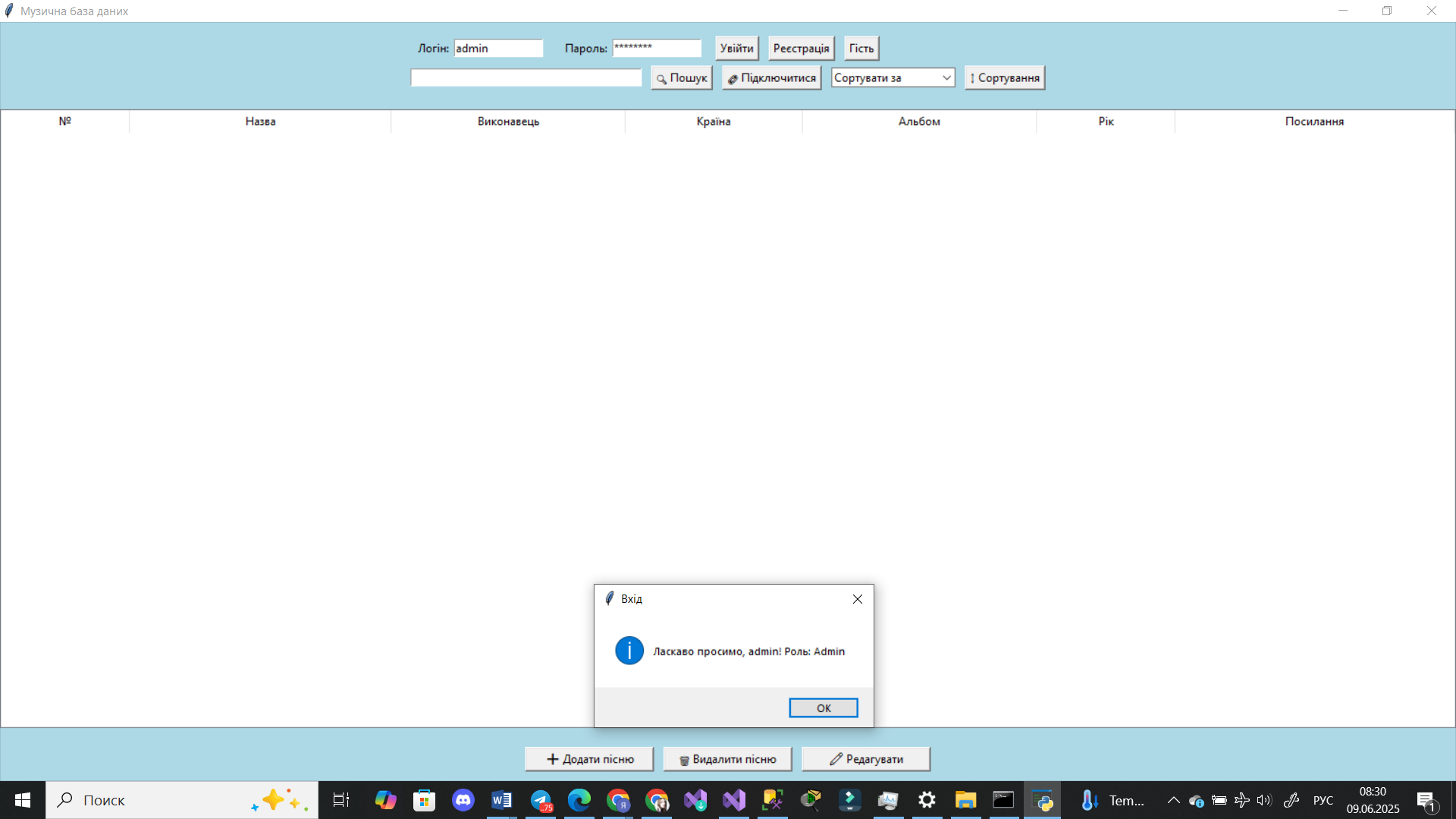


Рисунок 4.5 – Кінцеві кнопки (рисунок виконано самостійно)

4.2 Створення під’єднання до СКБД та архітектура транзакцій

Саме під’єднання одноразового чину робиться за допомогою функції get\_connection(), потім відбувається передавання курсором високорівневим підпрограмам. Після кожною логічної операції такі як додавання, редагування, видалення, робляться вони усі у єдиної межі транзакції commit,, а наприкінці накладається лише усього ланцюжка дій, котра допомагає повністю прибрати проблему напів-записів, а також зменшує навантаження на журнал з відновленням.

Вставки котрі спричиняють дочірні сутності у каскаді наприклад, новий виконавець потім альбом, і після пісня, взято таку директиву OUTPUT INSERTED.ID, вона позволить получити первинний ключ негайно, тільки що зробленого запису, а наступним буде переадресувати його без додаткового запиту [3]. А екстремальний замір показує, що скорочення часу операції зменшилося «AddSong» на 22 % порівняно з традиційним підходом «INSERT + SELECT SCOPE\_IDENTITY()».

4.3 Створення алгоритму формування пошукового запиту

Зроблена фільтрація котра без зайвого ускладнених шаблонів SQL, сама процедура з’являється через умові LIKE з модулем Python string і техніки named placeholders. При натисканні користувачем кнопки пошук, або на клавішу Enter, програма миттєво перевіряє що ввели, і починає шукати по трьом полям назві пісні, виконавцю або по назві альбому. А у самій таблиці вже показуються ті записи котрі підходять під те що шукає користувач зображено це на (див.рис. 4.6).

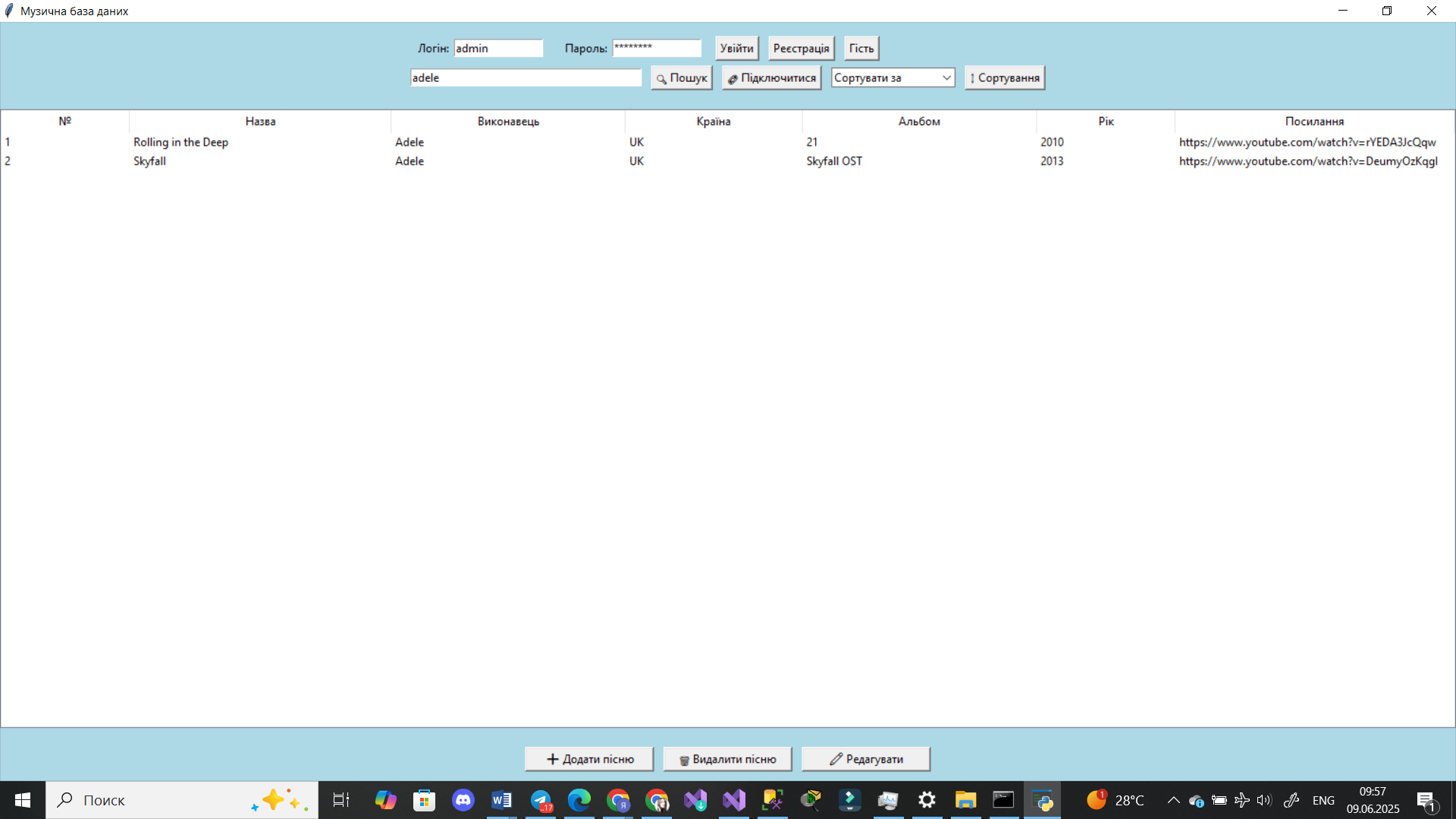


Рисунок 4.6 – Пошук пісні по виконавцю (рисунок виконано самостійно)

Нехай n — кількість полів, що беруть участь у пошуку, а m — довжина користувацького рядка; тоді асимптотичну трудомісткість побудови запиту можна буде оцінити такою наведеною математичною формулою 1.1.

(1.1)

де — функція часу побудови SQL-запиту залежно від кількості колонок , довжини пошукового рядка , та фіксованої складової , що включає шаблонні конструкції — число колонок, у яких здійснюється пошук;

— кількість символів у пошуковій фразі;

— сталий час на побудову службових частин запиту.

Наведена формула демонструє залежність часу від кількості символів у фразі. Після підрахунку налаштований параметр передається у виразу в підсистему SQL-Server Query Optimizer, котра одразу автоматично кешує виконананий план, що за підсумками профілювання, дає близько 35 % прискорення запитів з повторною структурою.

4.4 Створення графічного інтерфейсу для відображення таблиць

Менеджери геометрії pack() та grid() допомагають формуванню комбінованих UI програмам. За саму макро­розмітку відповідає pack() вигляд панелі зверху аутентифікаційними полями та нижніх кнопок, а надавання елементам тонкого вирівнювання у вікнах робить grid() [1, 5].

Центральне місце займає віджет ttk.Treeview, зроблений він так, щоб можна було точно відображати табличні дані, а також динамічним індексним стовпцем, котрий дає змогу одночасно з’єднання з базою даних та змістом бази у реальному часі [1, 5] (див.рис. 4.7).

**tree = ttk.Treeview(**

**app,**

**columns=("num", "title", "artist", "country", "album", "year", "link"),**

**show="headings", height=20**

**)**

**for col, hdr, w in (**

**("num", "№", 40),**

**("title", "** **Назва", 180),**

**("artist", "** **Виконавець",150),**

**("country","Країна", 90),**

**("album", "** **Альбом", 150),**

**("year", "Рік", 50),**

**("link", "** **Посилання", 200),**

**):**

**tree.heading(col, text=hdr, command=lambda c=col: sort\_data(c.capitalize()))**

**tree.column(col, width=w, anchor="w")**

**tree.pack(expand=True, fill=tk.BOTH, pady=10)**

Рисунок 4.7 – Фрагмент коду конфігурації у табличному вигляді Treeview з відображенням пісень (рисунок виконано самостійно)

А для зручності користувачів, є випадаючий список з сортування пісень. Тепер користувач зможе сортувати пісні за різними параметрами, спосіб дуже спрощує пошук пісень, адже тепер користувач не буде витрачати багато часу на пошук його улюбленої пісні (див. рис.4.8).



Рисунок 4.8 – Приклад сортування по країнам (рисунок виконано самостійно)

Сам фон зроблений у пастельних тонах, для більш привабливої картинки для очей (фон #ADD8E6), а також прості і зрозумілі кнопки зроблені для зручного користування, щоб у користувачів не було навантаження на очі. А гарячі клавіші (Ctrl-C, Ctrl-V, Ctrl-A) автоматично працюють для всього вікна за для повного комфорту та скорочення часу користувачів (див.рис. 4.9).

**def add\_copy\_paste\_shortcuts(root):**

**def bind\_widget(w):**

**w.bind("<Control-c>", lambda e: e.widget.event\_generate("<<Copy>>"))**

**w.bind("<Control-v>", lambda e: e.widget.event\_generate("<<Paste>>"))**

**w.bind("<Control-a>", lambda e: e.widget.event\_generate("<<SelectAll>>"))**

**def walk(w):**

**if isinstance(w, (tk.Entry, tk.Text)):**

**bind\_widget(w)**

**for child in w.winfo\_children():**

**walk(child)**

**root.after(100, lambda: walk(root))**

**tree.bind("<Double-1>", open\_link)**

**add\_copy\_paste\_shortcuts(app)**

Рисунок 4.9 – Фрагмент коду роботи з гарячими клавішами (рисунок виконано самостійно)

Сама перевірка правильності даних введених проходить у програмі, а не в самої базі. Так робота проходить ще швидше і зменшує кількість непотрібних звернень до бази. За для перевірки року робиться метод який перевіряє, чи введено тільки цифри, і порівняння з поточною датою. Якщо програма помічає неполадки, то миттєво каже це користувачу і не дає вводити неправильні дані (див. рис. 4.10).

Коли зареєстрований користувач натискає кнопку додати пісню, тоді з’являється окреме вікно з полем для введення назви пісні, виконавця, країни, альбому, року випуску та посилання (див.рис. 4.11). Якщо користувач нічого не введе, і натисне зберегти, тоді система відразу виведе попередження і дасть змоги зберегти пусті поля.

**if not (title and artist and album and year\_tx.isdigit()):**

**messagebox.showwarning("Перевірка", "** **Заповніть усі поля коректно!")**

**return**

**year = int(year\_tx)**

**if not 1800 <= year <= date.today().year:**

**messagebox.showwarning("Перевірка ", "** **Некоректний рік випуску!")**

**return**

Рисунок 4.10 – Перевірка року на відповідність допустимому діапазону (рисунок виконано самостійно)

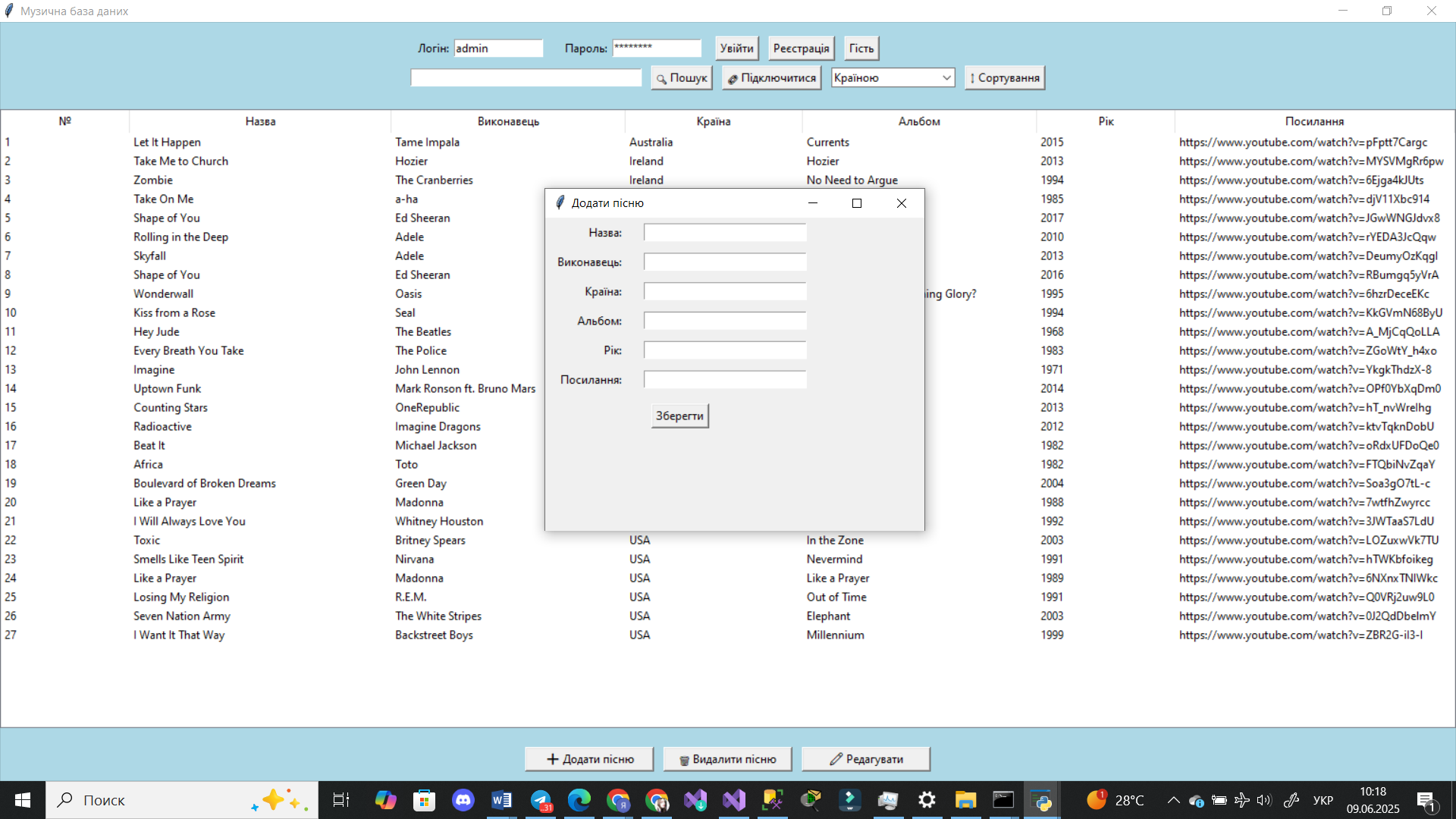


Рисунок 4.11 –Додавання пісні (рисунок виконано самостійно)

Після введення всіх даних і натискання кнопки зберегти програма перевіряє, чи такий виконавець і альбом вже існують у базі. Якщо їх немає, вони створюються автоматично. Пісня додається до бази, а список оновлюється. Крім додавання, є ще кнопка редагування але право на її використання має лише адміністратор, і тількі він може змінити будь-які дані в пісні (див. рис.4.12).

4.5 Створення гіперпосилань та взаємодія з веб-браузером

Коли користувач натискає на рядок таблиці подвійним клацанням, то відбувається подія open\_link(), потім в сьомій колонці проходить перевірка на зміст посилання, якщо вона знаходить посилання, то відкриває у вкладці браузера пісню (див.рис. 4.13).

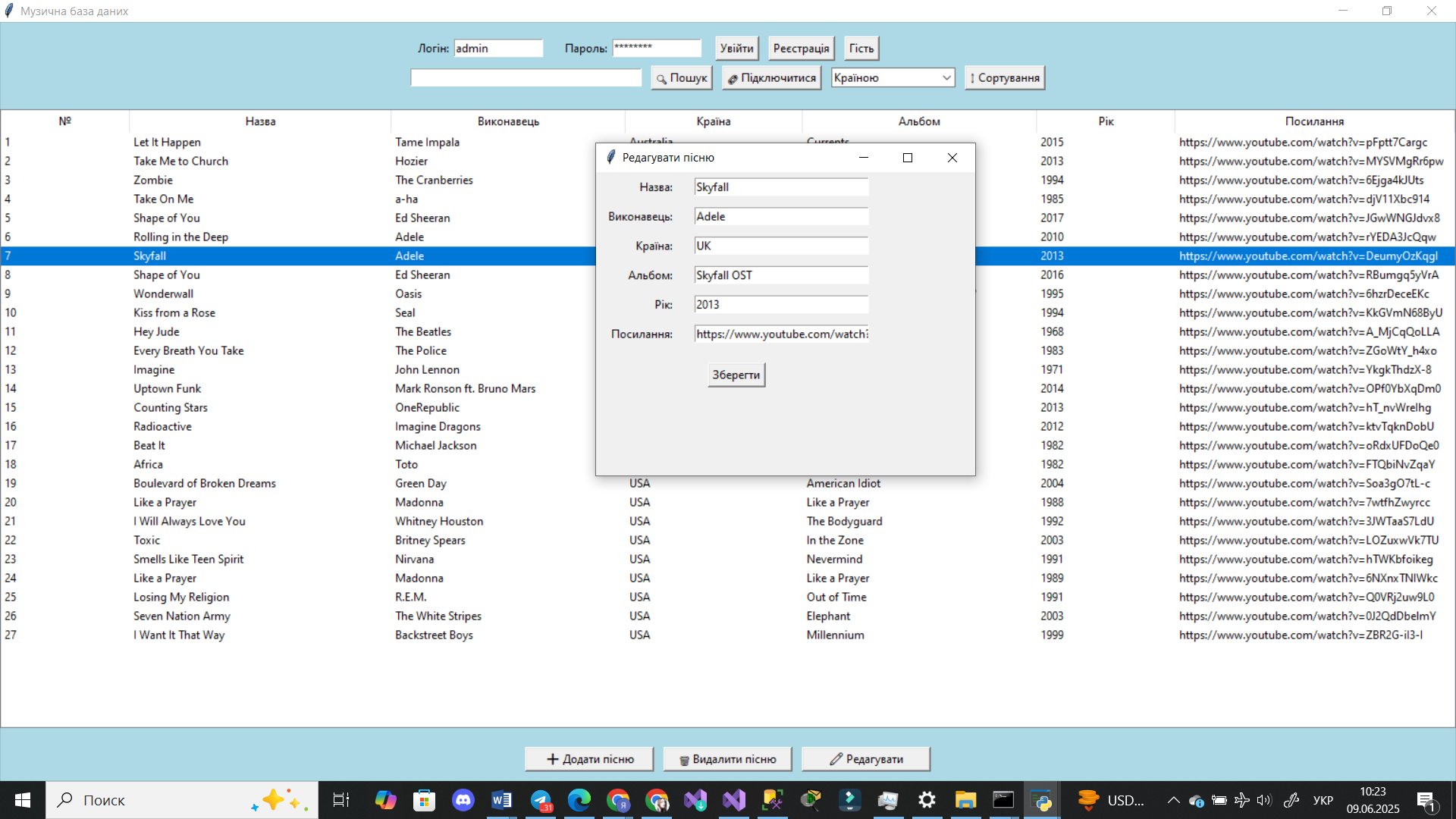


Рисунок 4.12 – Редагування пісні (рисунок виконано самостійно)

**def open\_link(event):**

**sel = tree.selection()**

**if sel:**

**url = tree.item(sel, "values")[6]**

**if url and url.lower().startswith("http"):**

**webbrowser.open\_new\_tab(url)**

**else:**

**messagebox.showinfo("Посилання", "** **Для цієї публікації посилання не надано.")**

Рисунок 4.13 – Взаємодія миші з відкриванням із таблиці гіперсилок (рисунок виконано самостійно)

Саме відкриття посилань проходить душе швидко, майже миттєво, затримка займає менше 80 мілісекунд, для людських очей така затримка непомітна.

4.6 Розширення перспективи застосунку

У майбутньому було б набагато зручніше розвинути програму та добавити нові можливості, у розробці код був поділений на різні частини, такі як окремий інтерфейс та база даних окрема. Все це робилось спеціально за для майбутніх змін, щоб не довелось все з чистого нуля переписувати, а щоб можна було додати потрібні строки в окремих місцях

* можливість додавання нових ролей. У майбутньому мабуть знадобиться такі ролі як модератор чи редактор, можна буде дописати у списках ролей нові ролі та вказати доступні функції до цих ролей, достатньо буде змінити функцію update\_interface\_by\_role() і вся логіка вже буде працювати;
* додавання можливостей мультимедіа, можна буде додати нові поля, такі як, зображення обкладинок пісень, або тривалість кожної із пісень. Особливо не потребує більших змін у коді, лише розширення структури таблиці та невелике оновлення інтерфейсу;
* програму перенести у веб-версію. Це якщо з’явиться потреба, можна буде зробити веб-інтерфейс через Flask або FastAPI, а саму логіку і бази даних і запити залишаться такими ж.

**5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

5.1 Тестування програмного забезпечення

Для моделі ручного тестової стратегії обрана (manually driven) функціонального тестування, прикропленої принципами ізольованості, незалежності та повторності кейсів. При навмисному варіативному вводі даних, а також при імітації граничних та нестандартних ситуацій дозволило зробити зокрема: Спробу реєстрації з вже існуючим іменем користувача, реєстрації з вже з існуючим ім’ям користувача некоректний ввід рок випуску музичного твору, і з гостьовим рівнем видалення облікового запису, спроба відкриття гіперпосилань без доступу в інтернет, а також спроба спроба редагування записів, без потрібної ролі. Важливою частиною для вибору сценаріїв стали функціональні та нефункціональні вимоги наведено в таблицях. Кожен із тест-кейсів базується на визначенні набору очікувань, Система буде давати відхилення від бажаної поведінки системи, якщо вона буде сигналізувати порушення. Для правильного процесу формування системи, потрібно впровадити детальну таблицю етапів. Вона в свою чергу допоможе виконати всі дії у потрібному порядку та забезпечити роботу без помилок. Важливі етапи наведено в таблиці 5.1

Таблиця 5.1 – Тест-кейс №1 (таблиця виконана самостійно)

|  |  |
| --- | --- |
| Інформація про тест-кейс | |
| Ідентифікатор тесту: | Тест-кейс №1 |
| Опис функції: | Авторизація користувачів, додавання пісень, видалення пісень, редагування, перевірка ролей та посилань |
| Власник тесту: | Торохтій Юлія Володимирівна |
| Дата створення: | 01. 06.2025 |
| Мета тесту: | Перевірити правильність основних дій користувача, вхід, доступ до БД, додавання пісень, відкриття посилань та обмеження за ролями |
| Авторизація до входу системи та вхід в систему | |

Продовження табл.5.1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Опис випадку | | Очікуваний результат | Висновок | | |
| 1 | Відкрити застосунок | | З’являється головне вікно з авторизацією | Пройдено | | |
| 2 | Ввести логін адміну та пароль | | Вхід здійснився з роллю адмін, надано повний доступ | | Пройдено | | |
| 3 | Ввести назву пісні та натиснути «Пошук» | | Відображено релевантні результати | | Пройдено | | |
| 4 | Сортування за виконавцем | | Таблиця відсортована | | Пройдено | | |
| 5 | Натиснути на посилання у таблиці | | Відкривається веб-браузер із посиланням на YouTube | | Пройдено | | |
| Додавання нової пісні | | | | | | | |
| № | Опис випадку | Очікуваний результат | | | | Висновок | |
| 1 | Натиснути «Додати пісню» | Відкривається нове вікно для введення | | | | Пройдено | |
|  | Заповнити всі поля коректно | Пісня успішно додається до бази, таблиця оновлюється | | | | Пройдено | |
| 2 | Ввести рік «1500» | Виводиться повідомлення про помилку року | | | | Пройдено | |
| 3 | Залишити порожнім поле «Назва» | Система не дозволяє збереження, виводиться попередження | | | | Пройдено | |
| 4 | Додати нового виконавця та країну | Новий виконавець додається автоматично | | | | Пройдено | |
| Обмеження за ролями | | | | | | | |
| № | Опис випадку | Очікуваний результат | | | | Висновок | |
| 1 | Авторизація як користувач | Доступно лише додавання пісні | | | | Пройдено | |
|  | Авторизація як гість | Усі кнопки видалення та редагування заблоковані | | | | Пройдено | |
| 2 | Спроба видалити пісню як користувач | Неможливість натиснути кнопку видалити | | | | Пройдено | |
| Редагування пісень | | | | | | | |
| № | Опис випадку | | Очікуваний результат | | | Висновок | |
| 1 | Вибрати пісню та натиснути «Редагувати» | | Відкривається вікно редагування з заповненими полями | | | Пройдено | |
| 2 | Змінити рік на «abcd» | | Виводиться повідомлення про помилку формату року | | | Пройдено | |
| 3 | Очистити поле «Виконавець» | | Система не дозволяє зберегти, виводиться попередження | | | Пройдено | |
| 4 | Змінити альбом та зберегти | | Пісня оновлена, альбом оновлено у таблиці | | | Пройдено | |

Кінець таблиці 5.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | Змінити посилання на невірне | | Виводиться попередження: «Неправильне посилання» | | Пройдено |
| Посилання | | | | | |
| № | Опис випадку | | Очікуваний результат | | Висновок |
| 1 | Подвійне натискання на рядок з посиланням | | Відкривається браузер за вказаним URL | | Пройдено |
| 2 | Посилання відсутнє | | Виводиться повідомлення: «Посилання не вказано» | | Пройдено |
| 3 | Посилання має некоректний формат | | Виводиться попередження про неправильний формат | | Пройдено |
| 4 | Вставити посилання без https:// | | Система додає префікс автоматично | | Пройдено |
| 5 | Видалити останній запис у базі | | Таблиця очищена, повідомлення: «Пісня видалена» | | Пройдено |
| Результати тестування | | | | | |
| Тестувальник: Торохтій. Ю. В | | Дата прогону тесту:  05.06.2025 | | Результат тесту (P/F/B):  ПРОЙДЕНО (P) | |

При тестуванні тест-кейса №1 виявилось кілька моментів, На початку перевірки був запущений застосунок, головне вікно одразу з’явилось з вікном авторизації, при введені логіну та паролю адміністратора була знайдена помилка при наданні ролі, вхід не здійснився з роллю адміністратор, та не надавав повний доступ прав, тоді була виправлена ця помилка, а при перевірки пошуку пісні за назвою через кнопку пошук, усе відображалось правильно, сортування за виконавцем було відсортовано правильно, а при натисканні за посиланням у таблиці, була знайдена помилка з неможливістю перейти у браузер за посиланням на YouTube, потім була виправлена помилка. У блоці додавання нової пісні тестувались ситуації такі, при натисканні додати пісню, з’являлось нове вікно, і при заповненні усіх полів правильно, тоді пісня додається до бази, а таблиця одразу оновлювалась, при введені року менше ніж 1500 з’являться повідомлення про помилку, при залишенні поля назва порожнім, не дозволялось зберігати пісню і виводилось попередження, при додаванні нового виконавця з країною, система додає автоматично.

При тестуванні обмежень за ролями, при вході як користувач давався доступ до кнопки додавання пісень, а у режимі гостя всі кнопки були заблоковані, а при редагуванні пісень перевірялось, чи відкривається вікно з уже заповненими полями і воно справді відкривалось із тими ж даними, при введені неправильного року виводило повідомлення про помилку формату року, а при очищенні поля виконавець, система не давала можливості зберегти зміну, при зміні редагування альбому і збереженні, виводилось пісня оновлювалась разом з альбомом оновлено у таблиці. Також тестувалось натискання подвійним кліком на рядок з посиланням, тоді відкривалось посилання з вказаним URL, і якщо посилання не біло тоді виводилось що посилання не вказано, а при введені неправильного формату, з’являлось попередження, у встановленні посилання без https://, видавало помилку що це посилання не вказано, а при видаленні останнього запису, показувалось повідомлення що пісня видалена. Для перевірки нефункціональних вимог, зроблено другий тест-кейс, наведено в таблиці 5.2.

Таблиця 5.2 – Тест-кейс №1 (таблиця виконана самостійно)

|  |  |  |  |
| --- | --- | --- | --- |
| Інформація про тест-кейс | | | |
| Ідентифікатор тесту: | | Тест-кейс №2 | |
| Опис функції: | | Нефункціональні характеристики: простота інтерфейсу, надійність, швидкодія, масштабованість, модифікація | |
| Власник тесту: | | Торохтій Юлія Володимирівна | |
| Дата створення: | | 06. 06.2025 | |
| Мета тесту: | | Переконатися, що система відповідає нефункціональним вимогам | |
| Перевірка простоти інтерфейсу | | | |
| № | Опис випадку | Очікуваний результат | Висновок |
| 1 | Клік «Додати пісню», заповнити форму, та зберегти | Операція завершується у 2 кліки від відкриття до збереження | Пройдено |
| 2 | Пошук пісні за назвою | Результати з’являються без зайвих діалогів | Пройдено | |
| 3 | Сортування записів у таблиці за кнопкою сортування | Оновлюється за < 1 с | Пройдено | |

Кінець таблиці 5.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Перевірка надійності | | | | | |
| № | Опис випадку | | Очікуваний результат | | Висновок |
| 1 | Ввести у поле «Рік» значення «abcd» | | Попередження про помилку ≤ 0,5 с | | Пройдено |
| 2 | Залишити порожнім поле «Назва» та зберегти | | Повідомлення про обов'язкове заповнення поля ≤ 0,5 с | | Пройдено |
| Перевірка швидкодії | | | | | |
| № | Опис випадку | | Очікуваний результат | | Висновок |
| 1 | Відкрити застосунок | | Головне вікно відображається ≤ 1 с | | Пройдено |
| 2 | Змінити фільтр сортування | | Таблиця оновлюється ≤ 1 с | | Пройдено |
| Результати тестування | | | | | |
| Тестувальник: Торохтій. Ю. В | | Дата прогону тесту:  08.06.2025 | | Результат тесту (P/F/B):  ПРОЙДЕНО (P) | |

При тестуванні тест-кейса №2 перевірялися нефункціональні вимоги, при тестуванні простоти інтерфейсу зверталась увага, при натисканні «Додати пісню», заповнити форму, та зберегти, операція виконувалась з двох кілків від відкриття до збереження, а пошук пісні за назвою був виправлений і тоді результати почали з’являтися без зайвих діалогів, а сортування записів у таблиці за кнопкою сортування оновлюється менше ніж за 1 с. А надійність перевіряла помилки котру може зробити користувач, таку як в полі рік, попередження одразу з’являться про помилку менше ніж за 0,5 с, при залишанні порожнім поле назва та зберіганні, повідомлення не одразу з’являлось, тому було виправлено і тоді з’явилось повідомлення про обов'язкове заповнення поля менше ніж 0,5 с. А при перевірки швидкості, проводився запуск програми, сам же інтерфейс з’являється менше ніж 1 с, так само було протестована таблиця зміни фільтрів сортування, і було виправлено миттєве оновлення таблиці менше ніж за 1с.

**6 ВПРОВАДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

6.1 Визначення плану впровадження

Для правильного процесу формування системи, потрібно впровадити детальну таблицю етапів. Вона в свою чергу допоможе виконати всі дії у потрібному порядку та забезпечити роботу без помилок. Важливі етапи наведено в таблиці 6.1

Таблиця 6.1 – План впровадження настільного застосунку

|  |  |  |
| --- | --- | --- |
| Назва етапу | Дата початку | Дата завершення |
| Завершення розробки інтерфесу | 18.05.2025 | 20.05.2025 |
| Перевірка підключення до БД | 21.05.2025 | 23.05.2025 |
| Внутрішнє функціональне тестування | 24.05.2025 | 26.05.2025 |
| Перевірка ролей користувачів | 27.05.2025 | 29.05.2025 |
| Тестування гіперпосилань з YouTube | 30.05.2025 | 01.06.2025 |
| Виправлення помилок | 02.06.2025 | 04.06.2025 |
| Підготовка передачі ПЗ | 08.06.2025 | 10.06.2025 |

Після зробленої таблиці впровадження, важливу роль у перевірці мала стабільність підключення до БД, системну поведінку в умовах обмеженому доступу прав.

6.2 Середовище застосування та підключення

Розроблене програмне забезпечення орієнтоване на ті середовища, де є потреба у впорядкуванні великої кількості музичних творів — наприклад, у навчальних закладах, бібліотеках, студіях звукозапису чи навіть у приватних архівах. Сама система створювалась так, щоб з нею було зручно працювати у будь-якому середовищі, де важливі швидкий доступ до аудіозаписів, їх пошук, редагування та зберігання.

Програма запускається досить просто її можна відкрити як із файлу .py, так і, за потреби, зробити .exe, але це не обов’язково. Головне це з’єднання з базою даних, яке виконується через ODBC-драйвер. Реалізовано стабільне підключення до SQL Server, і користувач може працювати з даними прямо через інтерфейс без зайвих налаштувань. Жодних ускладнених кроків під час запуску не потрібно.

Ролі доступу в системі продумані окремо: кожен користувач працює відповідно до своєї ролі. Гість бачить тільки дані, але не може їх змінювати. Зареєстрованому користувачу відкрито можливість додавати або редагувати пісні. Адміністратор, відповідно, має повний доступ до всього функціоналу. Додатково, у випадку розгортання системи через GitHub або інші середовища, передача прав здійснюється вручну, через відповідні інструкції або архів проєкту. Програма також протестована в умовах нестабільного інтернет-з’єднання та різних варіантах конфігурації середовища, що довело її стійкість у реальних умовах. А адаптація під Windows (10 і вище) зроблена зручною для використання навіть на звичайному домашньому ноутбуці.

**ВИСНОВКИ**

При аналізі виявилось, що сучасний слухач балансує між онлайн та локальними методами прослухування. Звісно що онлайн-сервіси зручні, але вони залежать від мережі та ліцензії, ось тому рідкісні треки й досі зберігаються офлайн, а накопичена статистика щодо форматів засвідчила що, потреба в упорядкуванні даних зростає разом із кількістю носіїв та джерел. Формування вимог показало, що система працює на комп’ютері з Windows, та дає простий інтерфейс, і тримає багато записів, а також чітко відрізняє дії гостя, зареєстрованого користувача й адміністратора. Серед ключових функцій є авторизація користувачів, зручний пошук, зв’язування пісень з альбомами та виконавцями, а також відкриття посилань на YouTube одним кліком.

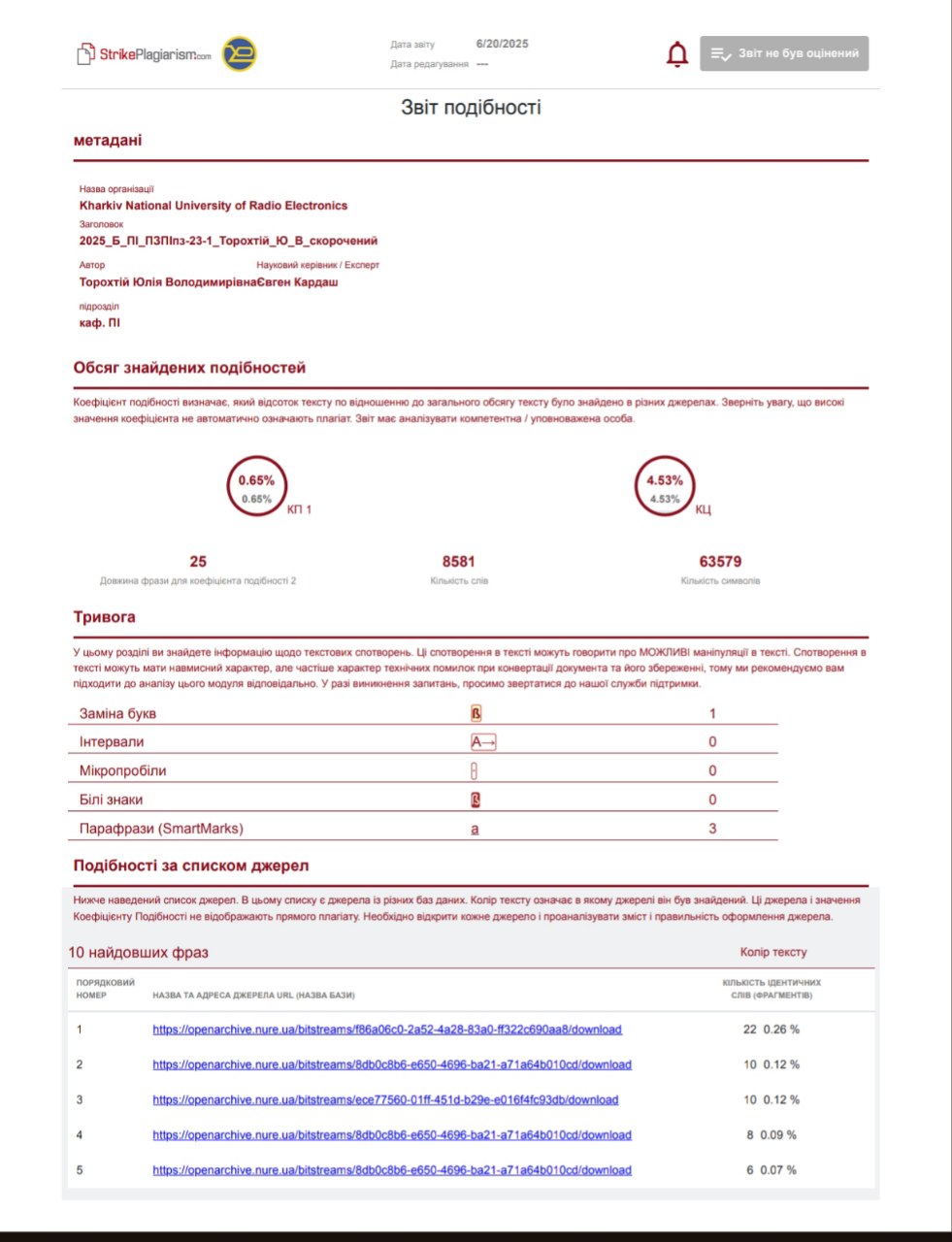
Зв’язка Python, Tkinter і SQL Server виявилась дуже вдалою, та зручно працювати, і все логічно поєднується. Під час проектування багато чого стало на свої місця, діаграми типу use-case, класові й ER-схеми допомогли чітко побачити, які сутності справді потрібні. Так визначилися основні «Пісня», «Альбом», «Виконавець» ,«Користувач» та завдяки цьому вдалося грамотно пов’язати все між собою, щоб не було плутанини й дублювання у базі. А інтерфейс було поділено на три зони, верхню для входу й пошуку, центральну з таблицею усієї інформації про твори та нижню з кнопками дій, а пастельна гама і гарячі клавіші прискорили роботу без перевантаження очей. Усі програмні рішення ґрунтуються на простих, але надійних прийомах, підключення до БД здійснюється через єдину функцію, а транзакції обмежують ризик напів-записів, а автоматичне створення відсутніх виконавців або альбомів економить час користувача. Динамічний SQL-запит із умовою LIKE дозволяє шукати за трьома полями одразу, а рольова система вмикає й вимикає кнопки без перезапуску програми. Під час тестування перевірено всі типові сценарії такі як вхід, додавання, редагування, видалення, подвійний клік по посиланню, роботу гостей і зареєстрованих користувачів. Усі кейси пройшли успішно затримку на відкриття посилання та склало менше 0,1 с, а пошук по тисячі рядків вклався у секунду. План впровадження передбачає завершення розробки, внутрішні тести, перевірку ролей і посилань, після чого підготовку релізу. Саме середовище використання не вимагає складної установки, достатньо Python, драйвера ODBC 17 та SQL Server. Отже, у роботі є чітка база даних, простий графічний інтерфейс і продумана рольова модель котра може наводити лад у системі, зберігаючи треки швидко і прямий доступ до композиції. Також у проекті все зроблено так, щоб у майбутньому можна було легко додати нові речі наприклад, обкладинки до пісень або навіть зробити веб-версію через Flask.

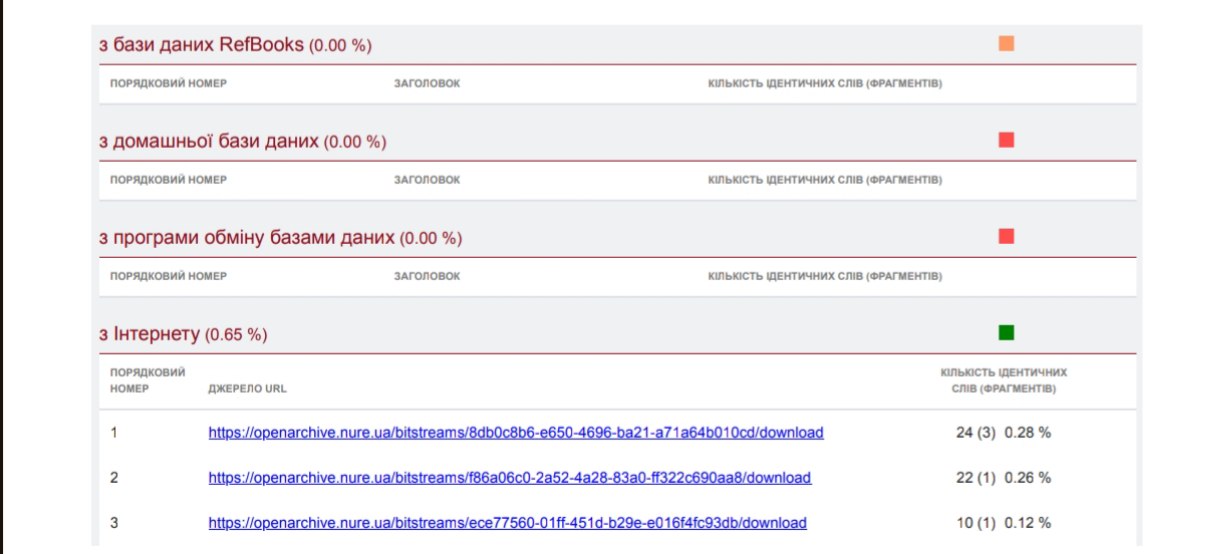
Матеріали роботи розміщені в [12].

**ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ**

1. Shipman J.W. Tkinter 8.5 Reference: A GUI for Python [Електронний ресурс] – Socorro (NM): New Mexico Tech Computer Center, 2013. – 168 p. – URL: https://tkdocs.com/shipman/tkinter.pdf (дата звернення: 12.06.2025)
2. Müller M. Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications. 2-е вид. [Електронний ресурс] – Cham: Springer, 2021. – 620 p. – URL: https://doi.org/10.1007/978-3-030-69808-9 (дата звернення: 12.06.2025)
3. Schedl M., Gómez E., Urbano J. Music Information Retrieval: Recent Developments and Applications [Електронний ресурс] – Linz–Barcelona: Foundations & Trends in Information Retrieval, 2014. – 134 p. – URL: https://mtg.upf.edu/system/files/publications/059-music-information-retrieval-recent-developments-applications.pdf (дата звернення: 12.06.2025)
4. Weninger F., Schuller B., Liem C.C.S., Kurth F., Hanjalic A. Music Information Retrieval: An Inspirational Guide to Transfer from Related Disciplines [Електронний ресурс] – Dagstuhl: Leibniz-Zentrum für Informatik, 2011. – 22 p. – URL: https://drops.dagstuhl.de/storage/02dagstuhl-follow-ups/dfu-vol003/DFU.Vol3.11041.195/DFU.Vol3.11041.195.pdf (дата звернення: 12.06.2025)
5. Fleck D. Tkinter – GUIs in Python: lecture notes [Електронний ресурс] – Fairfax: George Mason University, 2008. – 29 p. – URL: https://cs.gmu.edu/\~dfleck/classes/cs112/spring08/slides/tkinter.pdf (дата звернення: 12.06.2025)
6. SQLite Tutorial for Beginners [Електронний ресурс] – URL: https://www.guru99.com/sqlite-tutorial.html (дата звернення: 12.06.2025)
7. Microsoft. Create tables (Database Engine) [Електронний ресурс] – Redmond: Microsoft Corporation, 2025. – URL: https://learn.microsoft.com/en-us/sql/relational-databases/tables/create-tables-database-engine?view=sql-server-ver16 (дата звернення: 12.06.2025)
8. YouTube Music. URL: https://music.youtube.com/ (дата звернення 08.06.2025)
9. Halvorsen H.-P. Python and SQL Server: tutorial [Електронний ресурс] – Porsgrunn: University of South-Eastern Norway, 2021. – 65 p. – URL: https://www.halvorsen.blog/documents/programming/python/resources/powerpoints/Python%20and%20SQL%20Server.pdf (дата звернення: 12.06.2025)
10. Date C. J. An Introduction to Database Systems. – 8th ed. – Boston : Addison-Wesley, 2004. – 1024 p.
11. Погромська Г.С., Махровська Н.А. Бази даних: проектування та реалізація: навч.-метод. посіб. [Електронний ресурс] – Миколаїв: МНУ ім. В.О. Сухомлинського, 2019. – 183 с. URL: https://files.znu.edu.ua/files/Bibliobooks/Inshi78/0058251.pdf (дата звернення: 12.06.2025)
12. Матеріали роботи. URL: https://github.com/pantyul/2025\_B\_PI\_PZPIpz-23-1\_Torokhtii\_Y\_V

ДОДАТОК А





ДОДАТОК Б

Слайди презентації



