Міністерство освіти на науки України

Львівський Національний Університет Природокористування

|  |  |  |
| --- | --- | --- |
| Факультет |  | Кафедра |
| механіки, енергетики та ІТ |  | Інформаційних технологій |

***КУРСОВА РОБОТА***

з дисципліни: ***Обʼєктно-орієнтоване програмування***

**на тему:** Розроблення додатку для інтерактивного завантаження музики

Студент гр. ІТ-31 Дмитров Ігор Ігорович \_\_\_\_\_\_« » 2023.

(ПІБ, підпис, дата)

Керівник:\_\_\_\_\_\_\_\_\_к.т.н.,доцент Татомир А.В. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наук. ступінь, ПІБ)

Львівський національний університет ПРИРОДОКОРИСТУВАННЯ

Кафедра

**Інформаційних технологій**

**“Затверджую”**

завідувач кафедри ІТ

д.т.н., професор \_\_\_\_\_\_\_\_\_\_\_\_\_А.М. Тригуба

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_202\_р.

**ЗАВДАННЯ**

на курсову роботу з дисципліни

***«Обʼєктно-орієнтоване програмування»***

Студенту групи ІТ-31\_\_\_\_\_\_\_\_\_\_\_Дмитров Ігор Ігорович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

**1. Тема роботи:** «Розроблення додатку для інтерактивного завантаження музики»

2. Термін здачі студентом закінченої роботи \_27 листопада 2023 р.\_

**3. Індивідуальне завдання:**

* Передбачити можливість поступового запиту для завантаження.
* Реалізувати механізм завантаження.
* Використати не менше трьох класів і механізм наслідування.
* Передбачити можливість прослуховування завантаженої музики.

**4. Перелік питань, які необхідно розробити в роботі:**

*Реферат.* Вступ.

1. АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ.

1.1. Аналіз предмету проектування.

1.2. Класифікація об’єктно-орієнтованих мов програмування.

1.3. Огляд та аналіз сучасних технологій та засобів проектування програмного забезпечення.

1.4. Універсальна мова проектування UML.

1.5. Уточнена постановка задачі на розробку програмного забезпечення.

2. РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ.

2.1. Аналіз функцій системи.

2.2. Розроблення структурної моделі системи.

3. РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ.

3.1. Декомпозиція системи.

3.2. Аналіз взаємозв’язків між об’єктами.

3.3. Розроблення інтерфейсів класів.

3.4. UML-діаграма класів.

4. РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ.

5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.

6. РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ.

Висновки.

Бібліографічний список

Додатки

**5. Консультанти з розділів курсової роботи: А. Татомир**

**6. Дата видачі завдання 15.02.23**

**7. Дні і години консультацій**

**8. Завдання на курсову роботу затверджено на засіданні кафедри інформаційних систем і технологій**

**Протокол №\_\_\_ від \_\_\_\_\_\_\_\_\_\_\_\_20\_\_\_р.**

**Керівник роботи: А. Татомир**

**Завдання прийняв до виконання (дата, підпис) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Зміст

[Реферат 1](#_Toc151537330)

[Вступ 2](#_Toc151537331)

[1. АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ 3](#_Toc151537332)

[1.1. Аналіз предмету проектування 4](#_Toc151537333)

[1.2. Класифікація об’єктно-орієнтованих мов програмування 5](#_Toc151537334)

[1.3. Огляд та аналіз сучасних технологій та засобів проектування програмного забезпечення 7](#_Toc151537335)

[1.4. Універсальна мова проектування UML 9](#_Toc151537336)

[1.5. Уточнена постановка задачі на розробку програмного забезпечення 10](#_Toc151537337)

[2. РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ 13](#_Toc151537338)

[2.1. Аналіз функцій системи 13](#_Toc151537339)

[2.2. Розроблення структурної моделі системи 14](#_Toc151537340)

[3. РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ 16](#_Toc151537341)

[3.1. Декомпозиція системи 16](#_Toc151537342)

[3.2. Аналіз взаємозв’язків між об’єктами 17](#_Toc151537343)

[3.3. Розроблення інтерфейсів класів 18](#_Toc151537344)

[3.4. UML-діаграма класів 19](#_Toc151537345)

[4. РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ 21](#_Toc151537346)

[5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 25](#_Toc151537347)

[6. РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ 27](#_Toc151537348)

[Висновки 29](#_Toc151537349)

[Бібліографічний список 31](#_Toc151537350)

[Додаток А 32](#_Toc151537351)

[Додаток Б 33](#_Toc151537352)

[Додаток В 35](#_Toc151537353)

# Реферат

До курсової роботи на тему

«Розроблення додатку для інтерактивного завантаження музики»

Курсова робота обсягом 33 аркушів складається із вступу, шести розділів, висновків, переліку посилань із 10 джерел та трьох додатків. Містить 7 рисунків та 1 діаграму.

Метою підготовки курсової роботи є систематизація та закріплення отриманих знань у процесі вивчення дисципліни "Обʼєктно-орієнтоване програмування". Головною метою є розробка функціонального додатку, який надасть користувачам зручний та ефективний інструмент для взаємодії з музичними файлами.

Курсова робота покликана надати комплексний підхід до розробки програмного продукту, об'єднуючи у собі концепції об'єктно-орієнтованого програмування, високоякісну архітектуру, тестування та документацію. Розроблений додаток спрямований на задоволення потреб користувачів у взаємодії з музичним контентом та забезпечення їм позитивного та ефективного досвіду використання.

**Ключові слова:**

Музика, музичний плеєр, завантаження музики, інтерфейс користувача, обʼєктно-орієнтоване програмування, тестування програмного забезпечення, документація.

# Вступ

Музика завжди була суттєвою частиною нашого життя. Вона вражає нас, надихає, робить наші емоції більш насиченими та пам'ятними. Із розвитком технологій і доступністю Інтернету музичний контент став більш доступним, ніж будь-коли раніше. Однак іноді ми бажаємо мати можливість інтерактивно обирати та завантажувати музику з власних джерел.

У цій роботі ми розглянемо процес розроблення додатку для інтерактивного завантаження музики, використовуючи мову програмування Python. Цей проект відкриє перед нами двері до унікального світу програмування та музики, де ми матимемо можливість створювати власний інструмент для завантаження, управління та насолоди музикою, яка найбільше нас цікавить.

Ми докладно розглянемо крок за кроком, як створити такий додаток, використовуючи потужні бібліотеки та інструменти Python. Ви дізнаєтеся, як здійснювати пошук та завантаження аудіофайлів з Інтернету, створювати інтерфейс для користувача та взаємодіяти з ним, а також як працювати з аудіоданими для створення різноманітних функцій та можливостей.

Весь цей процес допоможе вам не лише зрозуміти, як працюють додатки для музичного завантаження, але й надасть вам можливість втілити власні ідеї та вдосконалити цей проект на ваш розсуд. Готові вирушити в подорож в світ програмування та музики? Давайте рушимо вперед та розпочнемо наше дослідження.

# 1. АНАЛІЗ СУЧАСНОГО СТАНУ ПИТАННЯ ТА ОБҐРУНТУВАННЯ ЗАВДАННЯ НА РОБОТУ

Аналіз сучасного стану питання та обґрунтування завдання на роботу є важливим кроком у процесі розробки будь-якого проекту. В цьому розділі проводиться огляд існуючих рішень та розглядається актуальність проекту. Аналіз сучасного стану питання допомагає визначити прогалини та можливості в обраній галузі і забезпечує обґрунтування обраного завдання.

Почнемо з аналізу сучасного стану. На сьогоднішній день існують численні додатки та сервіси для завантаження та відтворення музики. Деякі з них пропонують миттєвий доступ до безлічі треків через інтернет, в той час як інші надають можливість завантажувати музику для прослуховування в офлайн-режимі. Однак існують обмеження, такі як платна підписка на сервіси чи обмеження на обсяг завантаження.

Обґрунтування завдання на роботу полягає в тому, що створення власного додатку для інтерактивного завантаження музики на Python виправляє деякі з цих проблем. Мета цього проекту полягає в створенні зручного інструмента, який дозволить користувачам швидко та легко завантажувати та організовувати свою музику без обмежень, які часто пов'язані з існуючими рішеннями.

Обране завдання включає створення програми для пошуку, завантаження, збереження та відтворення аудіофайлів. Вона надасть користувачам можливість завантажувати улюблені пісні, створювати власні списки відтворення та насолоджуватися музикою за їхніми власними умовами.

Зважаючи на популярність музики та питому велику кількість користувачів, які шукають більше контролю над своєю музичною колекцією, обґрунтування завдання на роботу демонструє актуальність та цінність створення такого додатку.

## 1.1. Аналіз предмету проектування

Аналіз предмету проектування є важливим етапом в процесі розробки будь-якого проекту, оскільки дозволяє ретельно вивчити та зрозуміти всі аспекти, пов'язані із самим проектом. В цьому аналізі розглядаються всі ключові параметри, вимоги, особливості та обмеження, що стосуються предмету розробки.

Основні кроки аналізу предмету проектування включають:

* *Визначення мети проекту*: Це перший крок, в якому слід чітко сформулювати, що саме має бути досягнуто завдяки проекту. Наприклад, у нашому випадку мета проекту створення додатку для інтерактивного завантаження музики на Python.
* *Визначення користувачів та їх потреб*: Розгляньмо, для кого призначений проект, і які потреби він повинен задовольнити. Наші користувачі можуть бути ентузіастами музики, які шукають зручний спосіб завантажити та організувати свою музичну колекцію.
* *Вивчення конкурентів та існуючих рішень*: Проведемо аналіз існуючих додатків або сервісів, які вже пропонують схожі функції. Визначимо їх переваги та недоліки, щоб зрозуміти, як ми можемо відзначитися серед конкурентів.
* *Технічний аналіз*: Розгляньмо технічні аспекти проекту, включаючи вибір технологій, мов програмування, баз даних тощо. Цей аналіз допоможе визначити, які інструменти та ресурси будуть необхідні для реалізації проекту.
* *Визначення функціональних та нефункціональних вимог*: Визначимо, які функції має виконувати проект (функціональні вимоги) та які критерії якості чи безпеки мають бути враховані (нефункціональні вимоги).
* *Оцінка ризиків*: Визначимо можливі ризики, які можуть виникнути під час реалізації проекту, і розробимо план їх управління.
* *Визначення обсягу та графіка робіт*: Визначимо, які етапи роботи буде включати проект та складемо графік виконання завдань.

Аналіз предмету проектування допомагає зрозуміти всі аспекти проекту, визначити його цільову аудиторію, вимоги та обмеження. Ця інформація стане основою для подальшого проектування та розробки нашого додатку для інтерактивного завантаження музики на Python.

## 1.2. Класифікація об’єктно-орієнтованих мов програмування

Об'єктно-орієнтовані мови програмування (ООП) можна класифікувати на основі декількох основних критеріїв. Вот така класифікація мов програмування ООП [1]:

* *За рівнем об'єктності*:
  + Повністю об'єктно-орієнтовані мови: У таких мовах все є об'єктом, включаючи базові типи даних (наприклад, Smalltalk).
  + Частково об'єктно-орієнтовані мови: Вони дозволяють робити роботу з об'єктами, але також мають імперативні та функціональні елементи (наприклад, Java або C++).
  + Мови з об'єктно-орієнтованими можливостями: Це мови, які мають певні об'єктно-орієнтовані функції, але не є повністю ООП (наприклад, Python або Ruby).
* *За спадкоємністю класів*:
  + Однорівневі ООП мови: Мають один рівень класів, тобто всі класи є на одному рівні і не можуть успадковувати один одного.
  + Багаторівневі ООП мови: Мають різні рівні класів, де класи на більш низькому рівні можуть успадковувати класи на вищому рівні.
* *За типами спадкоємства*:
  + Одиночне спадкоємство: Клас успадковує властивості лише від одного базового класу (наприклад, в мові Java).
  + Багатоцільове спадкоємство: Клас може успадковувати властивості від кількох базових класів (наприклад, в мові C++).
* *За поліморфізмом*:
  + Динамічний поліморфізм: Властивість, що дозволяє об'єктам викликати однаково названі методи, але різні класи можуть виконувати їх по-різному (наприклад, використовуючи перевизначення методів).
  + Статичний поліморфізм: Властивість, яка дозволяє об'єктам викликати однаково названі методи в залежності від передачі параметрів (наприклад, перевантаження методів).
* *За підтримкою інкапсуляції*:
  + Мови зі строгою інкапсуляцією: Вони надають обмежений доступ до внутрішніх деталей об'єктів.
  + Мови з розширеною інкапсуляцією: Вони дозволяють більший рівень доступу до внутрішніх деталей об'єктів.

Ця класифікація допомагає розуміти різницю між об'єктно-орієнтованими мовами програмування та їх характеристиками, дозволяючи вибрати найкращий інструмент для конкретного завдання.

## 1.3. Огляд та аналіз сучасних технологій та засобів проектування програмного забезпечення

Цей аналіз допомагає розуміти, які рішення відповідають потребам проекту та забезпечують ефективність та якість в розробці програмного забезпечення. У цьому огляді розглядаються різні аспекти сучасних технологій та інструментів для програмного проектування:

* Мови програмування: Огляд і аналіз мов програмування, які можуть бути використані для реалізації проекту. Розглядаються основні характеристики, продуктивність, підтримка спільноти та додаткові бібліотеки.
* Фреймворки та бібліотеки: Аналіз фреймворків та бібліотек, які спрощують розробку програмного забезпечення, забезпечують готові рішення та допомагають розробникам прискорити роботу.
* Інструменти управління проектом: Розгляд інструментів, призначених для планування, відстеження завдань, спільної роботи та управління проектом (наприклад, Jira, Trello, Asana).
* Системи контролю версій: Огляд систем контролю версій, які дозволяють ефективно ведення та відстеження змін в програмному коді (наприклад, Git).
* Інструменти автоматизації тестування і CI/CD: Розгляд інструментів для автоматизації тестування, неперервної інтеграції та постачання (наприклад, Jenkins, Travis CI, CircleCI).
* Облачні сервіси та інфраструктура: Огляд можливостей обчислення в хмарних сервісах, які надають інфраструктуру для розгортання та масштабування додатків (наприклад, AWS, Azure, Google Cloud).
* Бази даних та системи керування базами даних: Розгляд різних типів баз даних і систем керування базами даних (наприклад, PostgreSQL, MySQL, MongoDB).
* Засоби безпеки та захисту даних: Аналіз інструментів та методів для забезпечення безпеки даних та захисту від потенційних загроз.
* Методології розробки програмного забезпечення: Розгляд сучасних методологій розробки, таких як Agile, Scrum, Kanban, Waterfall, і їх використання в конкретному проекті.
* Інші технологічні інновації: Огляд новітніх технологічних тенденцій, таких як штучний інтелект, блокчейн, Інтернет речей (IoT) та інші.

## 1.4. Універсальна мова проектування UML

Універсальна мова проектування (UML) є стандартизованою мовою для моделювання та проектування програмного забезпечення. UML була створена з метою спростити і полегшити процес розробки програм та систем і забезпечити спільну мову для комунікації між розробниками, аналітиками та іншими учасниками проекту. Ось декілька ключових аспектів UML [2]:

* *Стандартизація*: UML розроблена Інститутом інженерів програмного забезпечення (SEI) та є міжнародним стандартом, який підтримується Об'єднаною міжнародною асоціацією з інформатики та обчислювальної техніки (ACM) та Інститутом інженерів електротехніки та електроніки (IEEE).
* *Графічне моделювання*: UML використовує графічні елементи для створення діаграм, які візуалізують різні аспекти програми або системи. До типових діаграм UML включають діаграми класів, діаграми послідовності, діаграми станів та інші.
* *Мультипарадигмальність*: UML підтримує різні аспекти розробки програмного забезпечення, включаючи об'єктно-орієнтоване, структурне та функціональне програмування. Це дозволяє розробникам вибирати підходи, які найкраще відповідають конкретному проекту.
* *Комунікація і документація*: UML дозволяє розробникам легко спілкуватися і документувати проекти. Вона надає засоби для зміцнення зрозуміння вимог клієнтів та команди розробників.
* *Спрощення проектування*: UML спрощує процес проектування, дозволяючи моделювати систему на високому рівні та виділяти ключові компоненти.
* *Реінжиніринг і аналіз*: UML може використовуватися не лише для розробки нових проектів, але й для аналізу та реінжинірингу існуючого програмного забезпечення.

UML став популярним інструментом для розробників програмного забезпечення та системних інженерів, оскільки він дозволяє створювати зрозумілі та ефективні моделі програм та систем.

## 1.5. Уточнена постановка задачі на розробку програмного забезпечення

Уточнена постановка задачі є важливим кроком у розробці програмного забезпечення, оскільки вона визначає вимоги та очікування щодо проекту. Такий документ допомагає забезпечити успішну реалізацію та впровадження програми.

**Назва проекту**: Розроблення інтерактивного додатку для завантаження музики на Python.

**Опис проблеми або потреби**: Сучасні користувачі музичних сервісів бажають мати можливість інтерактивного завантаження та управління своєю музикою зручним та простим способом. Даний проект спрямований на створення програмного забезпечення, яке надає користувачам можливість завантажувати та відтворювати музику на платформі Python.

**Мета проекту**:

* Розробити додаток, який дозволить користувачам завантажувати музику з їхнього пристрою на платформу.
* Забезпечити інтерактивний та зручний інтерфейс для вибору, завантаження та відтворення музичних файлів.
* Забезпечити можливість керування списками відтворення та організації музичної колекції.

**Вимоги до функціональності**:

* Можливість завантажувати музичні файли.
* Можливість пошуку пісень.
* Відтворення музики та управління відтворенням
* Графічний інтерфейс для користувача.

**Нефункціональні вимоги**:

* Коректна робота додатку на операційних системах, що підтримуються.
* Зручний та інтуїтивний інтерфейс користувача.
* Висока продуктивність та мінімальне споживання ресурсів, щоб забезпечити плавне відтворення музики.
* Забезпечення безпеки та конфіденційності для користувачів та їхніх музичних файлів.

**Список обмежень**:

* Обмежений бюджет для розробки проекту.
* Терміни виконання проекту відповідно до графіку.

**Очікувані результати**:

* Готовий програмний додаток, який дозволяє користувачам інтерактивно завантажувати та відтворювати музику.
* Документація проекту, яка включає інструкції щодо використання та налаштування додатку.
* Виправлення помилок та підтримка випуску поновлень для покращення функціональності та безпеки додатку.

# 2. РОЗРОБКА ЗАГАЛЬНОЇ СТРУКТУРИ ПРОГРАМИ

Розробка загальної структури програми - це важливий етап у створенні програмного забезпечення. Вона включає в себе визначення структурних елементів, модулів та основних компонентів програми, які допомагають організувати код та забезпечити його логічну структуру.

## 2.1. Аналіз функцій системи

Аналіз функцій системи важливий крок у розробці програмного додатку для інтерактивного завантаження музики на Python. Він допомагає визначити, які функціональні можливості та операції повинні бути реалізовані в системі для задоволення потреб користувачів. Ось деякі функції, які можуть бути важливими для додатку:

1. **Завантаження музики**: Користувачі повинні мати можливість завантажити свої музичні файли з комп'ютера на додаток.
2. **Відтворення музики**: Додаток повинен надавати можливість відтворювати завантажену музику.
3. **Управління відтворенням**: Користувачі повинні мати зручний спосіб управління відтворенням музики, включаючи паузу, перемотування та гучність.
4. **Відображення метаданих**: Додаток повинен відображати інформацію про виконавця, назву пісні та інші метадані для користувача під час відтворення.
5. **Апаратна сумісність**: Впевненість, що додаток працює на різних операційних системах та апаратних платформах.

Це лише загальний перелік функцій, і важливо враховувати конкретні потреби і вимоги користувачів для додатку. Після визначення функцій можна розробляти детальніший опис кожної функції, включаючи їхні взаємозв'язки та послідовність виконання операцій. Цей аналіз допоможе створити чіткий план розробки та забезпечити успішну реалізацію вашого додатку.

## 2.2. Розроблення структурної моделі системи

Розроблення структурної моделі системи - це процес визначення компонентів, модулів, їхніх взаємозв'язків та організації системи для досягнення певних функціональних цілей. Для додатку для інтерактивного завантаження музики на Python, структурна модель може виглядати наступним чином:

1. **Головний файл програми (Main.py)**: Це початкова точка входу в програму. Головний файл відповідає за ініціалізацію та запуск програми. Він містить основний цикл програми та взаємодіє з іншими модулями.
2. **Модуль інтерфейсу користувача (UI.py)**: Цей модуль містить код для графічного інтерфейсу користувача, включаючи вікна, кнопки, поля введення та інші елементи управління. Він взаємодіє з користувачем та передає команди до інших модулів.
3. **Модуль завантаження музики (MusicLoader.py)**: В цьому модулі реалізована логіка завантаження музичних файлів з комп'ютера користувача. Він отримує файли від користувача та передає їх для подальшої обробки.
4. **Модуль управління відтворенням (MusicPlayer.py)**: Цей модуль відповідає за відтворення музики та управління відтворенням, включаючи функції паузи, перемотування та гучність.
5. **Модуль роботи з аудіофайлами (AudioProcessor.py)**: В цьому модулі реалізована обробка та робота з аудіофайлами, включаючи визначення метаданих, які включають в себе інформацію про виконавця, назву пісні та інші дані.
6. **Тестові модулі (Test.py)**: Модуль тестів для перевірки функціональності та відладки програми.

Структурна модель системи вказує, як ці компоненти взаємодіють між собою для досягнення функціональних цілей додатку. Кожен модуль має визначені інтерфейси та зобов'язання, і це допомагає забезпечити чітку організацію та розділеність коду, полегшуючи розробку та підтримку програми.

# 3. РОЗРОБЛЕННЯ ІЄРАРХІЇ КЛАСІВ

Розробка ієрархії класів - це важливий аспект при розробці програмного забезпечення, оскільки вона допомагає організувати та структурувати код, полегшуючи розширення функціональності та підтримку програми.

## 3.1. Декомпозиція системи

Декомпозиція системи - це процес розбиття складної системи на менші, більш керовані та прості компоненти, які можуть бути розроблені, випробувані та підтримані окремо. Цей підхід допомагає полегшити розробку та керування великими проектами, зменшує складність та ризики, а також сприяє більшій повторній використовуваності коду.

Для додатку завантаження музики декомпозиція системи може бути наступною:

* + **Модуль для завантаження музики**: Відповідає за завантаження музики з комп'ютера користувача.
  + **Модуль для відтворення музики**: Відповідає за відтворення та управління відтворенням.
  + **Модуль для графічного інтерфейсу**: Відповідає за створення та взаємодію з графічним інтерфейсом.
  + **Модуль для роботи з аудіофайлами**: Відповідає за обробку та аналіз аудіофайлів.

Цей підхід допоможе зробити код більш організованим, легше керованим та розширюваним. Кожен модуль чи клас відповідає за конкретну частину функціональності та може бути розроблений, випробуваний та підтриманий окремо.

## 3.2. Аналіз взаємозв’язків між об’єктами

Аналіз взаємозв'язків між об'єктами - це важливий аспект при розробці програмного забезпечення, оскільки він допомагає зрозуміти, як об'єкти взаємодіють один з одним, передають інформацію та виконують функції. Для додатку завантаження музики аналіз взаємозв'язків між об'єктами може включати наступні аспекти:

* + Клас *MusicLoader* може використовувати клас *AudioFile* для обробки аудіофайлів та витягнення метаданих.
  + Клас *MusicPlayer* взаємодіє з класом *AudioFile* для відтворення аудіо та відображення метаданих.
  + Клас *UserInterface* може використовувати класи *MusicLoader* та *MusicPlayer* для взаємодії з користувачем.

Цей аналіз допомагає визначити, як інформація та управління перетинаються між різними частинами програми та як об'єкти взаємодіють для досягнення функціональних цілей. Це також допомагає зрозуміти, як можна оптимізувати та поліпшити взаємодію між об'єктами для досягнення бажаних результатів.

## 3.3. Розроблення інтерфейсів класів

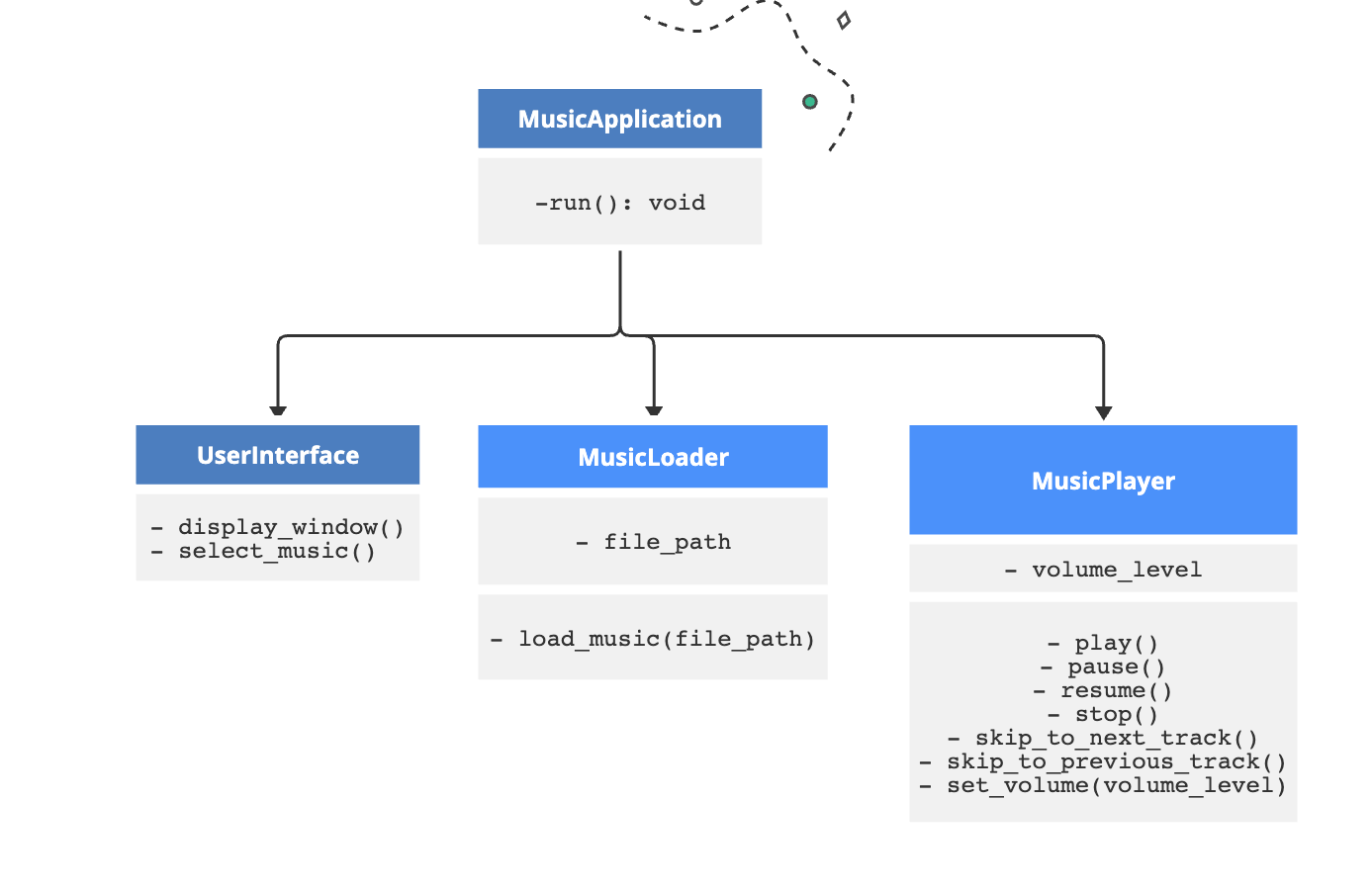
Розроблення інтерфейсів класів - це визначення публічних методів та властивостей класу, які доступні для взаємодії з іншими частинами програми. Інтерфейси визначають, як об'єкти класу можуть використовуватися та які дії можна виконувати з цими об'єктами. Для вашого додатку для інтерактивного завантаження музики на Python, розроблення інтерфейсів класів може включати наступні аспекти:

1. **Інтерфейс класу *MusicLoader***:
   * Метод *load\_music(file\_path)*: Цей метод приймає шлях до музичного файлу та виконує завантаження цього файлу в програму.
   * Метод *get\_metadata():* Цей метод повертає метадані аудіофайлу, такі як назва пісні, виконавець тощо.
2. **Інтерфейс класу *MusicPlayer***:
   * Метод *play()*: Цей метод відтворює музику або призупиняє відтворення музики.
   * Метод *next\_track():* Цей метод переходить до наступного треку у списку відтворення.
   * Метод *previous\_track():* Цей метод переходить до попереднього треку у списку відтворення.
   * Метод *set\_volume(volume\_level):* Цей метод встановлює гучність відтворення музики.
3. **Інтерфейс класу *UserInterface*:**
   * Методи та події, які відповідають за відображення графічного інтерфейсу, включаючи кнопки, вікна та інші елементи.

За допомогою визначення цих інтерфейсів ви створюєте чітку специфікацію того, як класи можуть взаємодіяти один з одним та як користувачі можуть взаємодіяти з додатком. Це спрощує розробку та забезпечує послідовність взаємодій між об'єктами. Кожен клас може реалізувати власний інтерфейс та методи, які відповідають за конкретну функціональність цього класу.

## 3.4. UML-діаграма класів

UML (Unified Modeling Language) - це мова для візуального моделювання програмного забезпечення, включаючи створення діаграм класів для подання структури та взаємозв'язків між класами в програмі. Приклад UML-діаграми класів для додатку для інтерактивного завантаження музики на Python.



3.1. Спрощена UML-діаграма

У цій UML-діаграмі класів:

* + *MusicApplication* - головний клас програми, який містить посилання на інші класи та метод run(), який ініціалізує та запускає програму.
  + *UserInterface* - клас, відповідальний за графічний інтерфейс користувача. Він має методи для відображення вікон та взаємодії з користувачем, такі як display\_window() та select\_music().
  + *MusicLoader* - клас для завантаження музики. Він має метод load\_music(file\_path), який відповідає за завантаження музичних файлів.
  + *MusicPlayer* - клас для відтворення музики та управління відтворенням. Містить методи для відтворення, паузи, перемотування та інші функції.

Ця UML-діаграма допомагає візуалізувати структуру програми та взаємозв'язки між класами. Вона може бути використана як вихідна точка для подальшого розвитку та розробки програми, допомагаючи зрозуміти, як класи взаємодіють між собою.

# 4. РОЗРОБКА СИСТЕМИ ВВОДУ-ВИВОДУ ДАНИХ

Отож, розробка системи вводу-виводу даних для додатку інтерактивного завантаження музики є важливою складовою даної роботи. Це дозволяє реалізувати зручну та інтуїтивну взаємодію користувача з програмою.

При проектуванні системи вводу-виводу було враховано сучасні тенденції розробки інтерфейсів [3], а саме:

* + Використання різних платформ для взаємодії: командний рядок для швидкого тестування, графічний інтерфейс для зручної роботи на ПК, веб-інтерфейс для кросплатформеності, Telegram бот для мобільності.
  + Застосування інтуїтивних елементів керування: текстові поля для введення запитів, кнопки для вибору треків та керування програванням, повзунки для регулювання гучності.
  + Використання зручних способів відображення даних: текстові повідомлення для виводу результатів пошуку та статусів, графічні елементи для відображення обкладинок та інформації про треки.
  + Інтеграція мультимедіа: відтворення аудіо, відображення зображень обкладинок.
  + Асинхронна обробка даних та оновлення інтерфейсу для плавної роботи.

Для реалізації взаємодії з користувачем використовуються можливості таких бібліотек:

* Tkinter - для створення графічного інтерфейсу з віджетами, такими як мітки, кнопки, текстові поля, списки, тощо. Це дозволяє швидко та якісно побудувати GUI. [4]
* Telegram Bot API - для інтеграції з месенджером Telegram за допомогою бота. Використовуються методи відправлення повідомлень, медіафайлів, інлайн-кнопок.
* Flask - для створення веб-інтерфейсу на основі Python фреймворку. Використовуються декоратори для обробки HTTP-запитів і шаблонізатор Jinja2 для генерації HTML.

Додатково застосовуються можливості мови Python - робота з файловою системою, мультимедіа, модулі для обробки метаданих аудіо.

Результат готової програми можемо побачити на скріншотах нище.

Основні методи програми описані в «[*Додаток А*](#_Додаток_А)».

А повний код знаходиться в «[*Додаток В*](#_Додаток_В)».

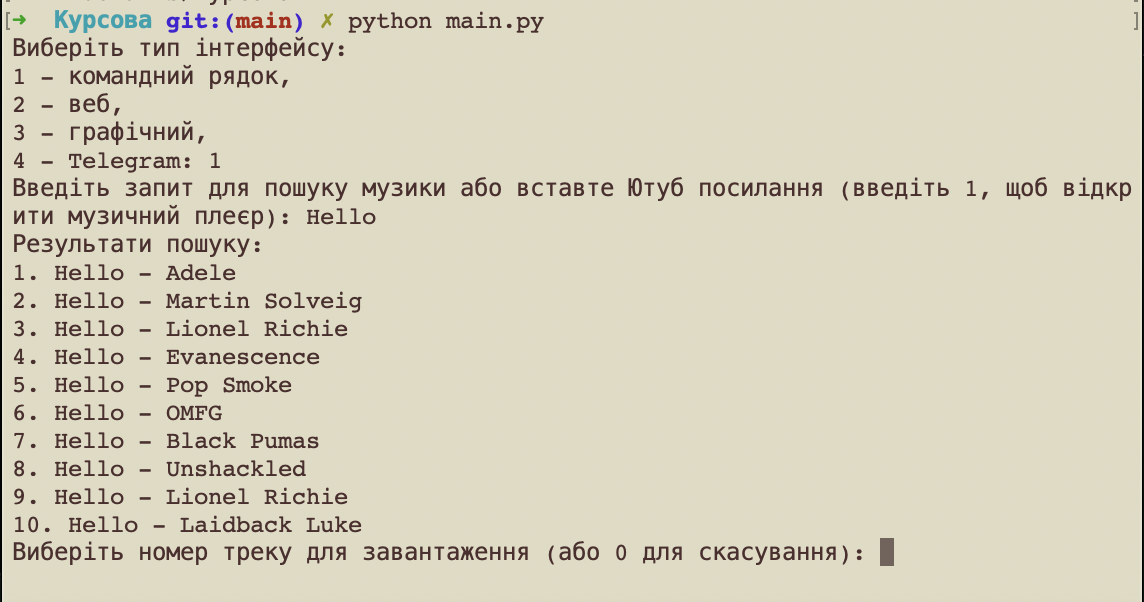


Рис. 4.1 Робота програми через інтерфейс командного рядка.

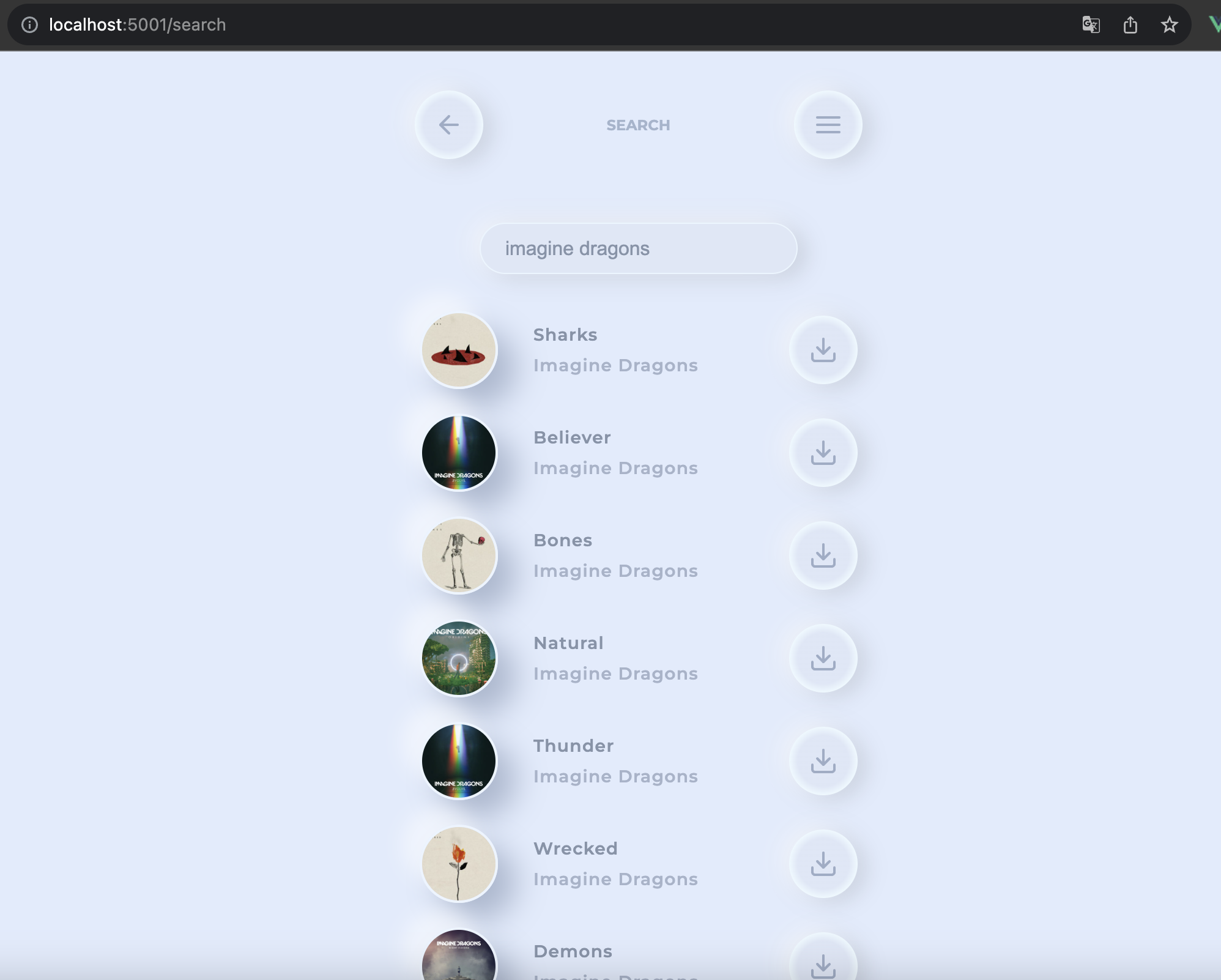


Рис. 4.2 Робота програми через веб інтерфейс

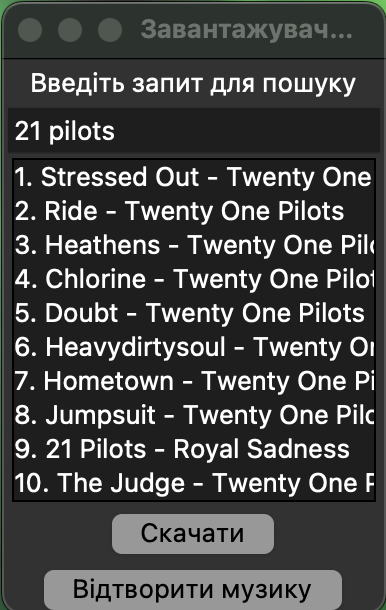


Рис. 4.3 Робота програми через графічний інтерфейс

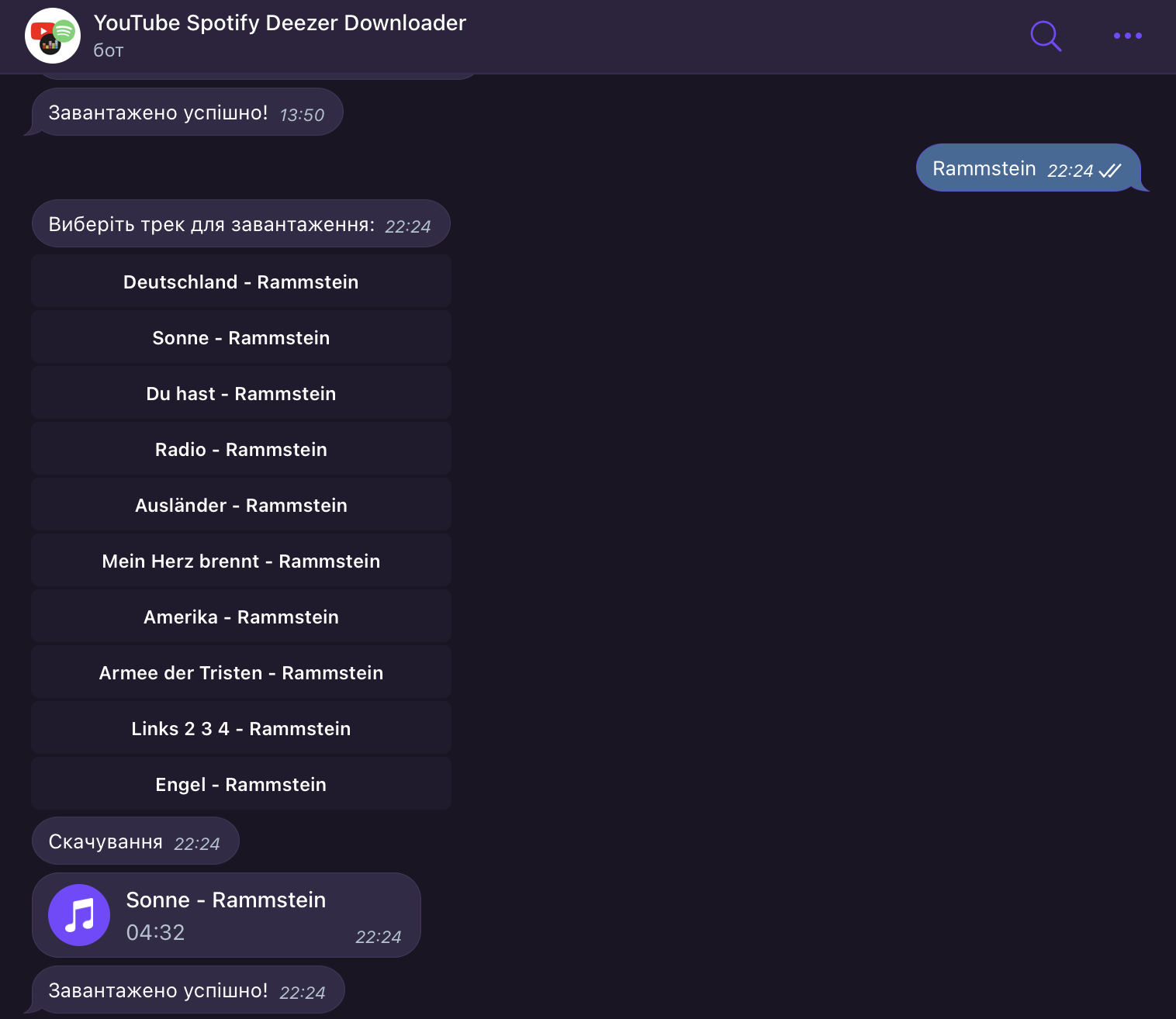


Рис. 4.4 Робота програми через Telegram-бота

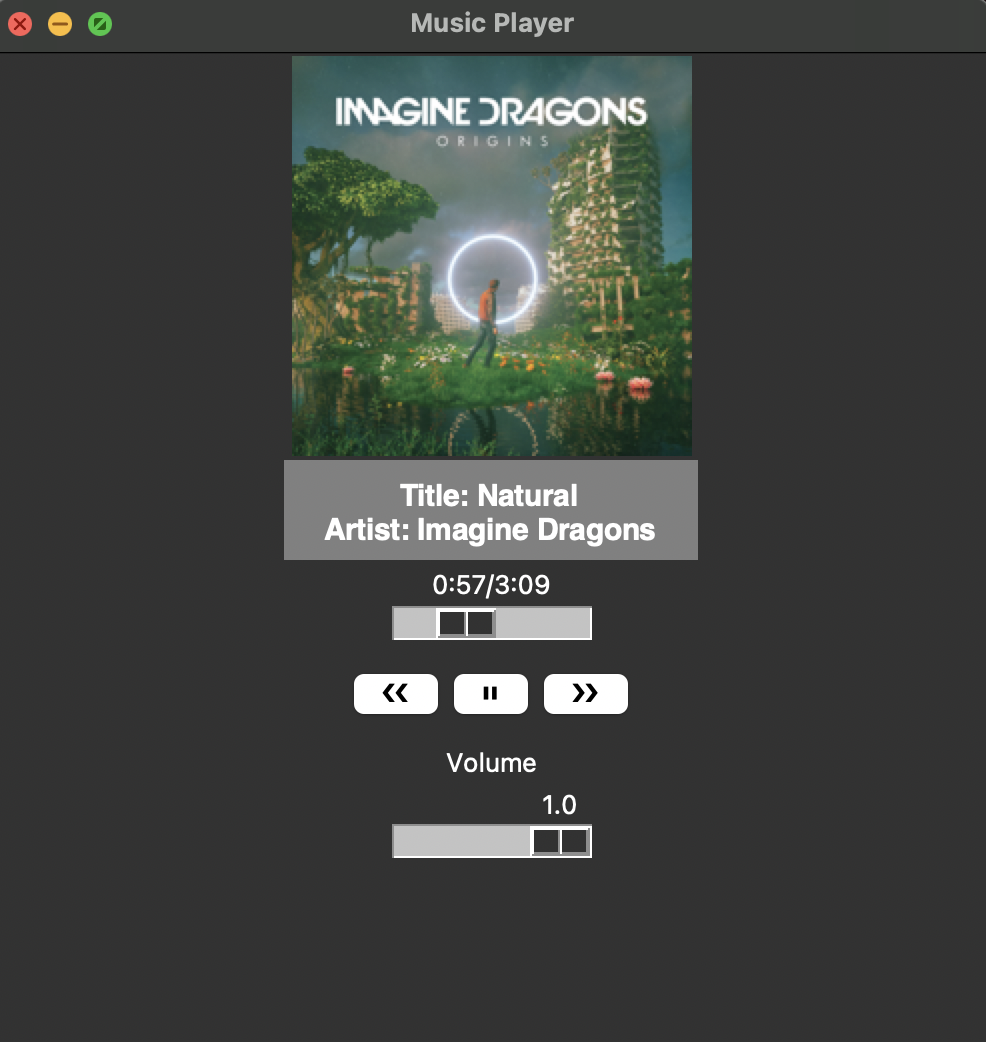


Рис 4.5 Робота музичного плеєра для відтворення завантажених пісень

Така система вводу-виводу дозволяє досягти максимальної зручності використання програми, що є важливим фактором при розробці сучасних додатків. Гнучкість інтерфейсів дає можливість легко розширювати функціонал у майбутньому.

# 5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Тестування програми є важливим етапом розробки, оскільки дозволяє виявити помилки та перевірити працездатність системи. Для тестування даного додатку було використано наступні види тестування:

* + Модульне тестування - перевірка роботи окремих компонентів програми, таких як класи, функції, методи. Це дозволяє забезпечити коректну роботу базової функціональності.
  + Інтеграційне тестування - перевірка взаємодії між модулями програми. Наприклад, тестування пошуку музики, завантаження треку, відтворення.
  + Тестування інтерфейсу - перевірка інтерфейсів користувача на предмет зручності, інтуїтивності, коректного відображення даних.
  + Тестування на різних платформах - перевірка роботи програми в різних операційних системах та середовищах (Windows, Linux, MacOS, Telegram, браузери).
  + Навантажувальне тестування - перевірка працездатності системи під високим навантаженням, наприклад, багато одночасних запитів до API.
  + Тестування безпеки - пошук можливих вразливостей та перевірка захисту від злому.

Нижче наведений код юніт-тестів для перевірки роботи модуля пошуку та завантаження музики.



Рис 5.1 Юніт-тести для перевірки модуля пошуку та завантаження музики

За результатами тестування були виявлені та виправлені деякі дефекти, а також оптимізована продуктивність та стабільність роботи додатку [7]. Тестування дозволило підвищити якість програмного продукту.

Отже, комплексне тестування є важливим етапом розробки, що дозволяє виявити та виправити недоліки системи до її виходу в експлуатацію.

# 6. РОЗРОБКА ДОКУМЕНТАЦІЇ ДЛЯ СУПРОВОДЖЕННЯ ПРОГРАМНОГО ПРОДУКТУ

Документація є важливою складовою програмного забезпечення, оскільки полегшує його подальше використання, модифікацію та підтримку.

Для даного додатку було розроблено таку документацію:

* Технічне завдання - документ, що визначає мету, основні вимоги та функціонал системи.
* Посібник користувача - містить інструкції щодо встановлення, налаштування та використання програми кінцевими користувачами.
* Посібник розробника - детально описує архітектуру системи, структуру коду, особливості налаштування для подальшої розробки та підтримки.
* README файли - містять короткий опис призначення та способу використання окремих модулів чи пакетів програми.
* Коментарі в коді - пояснення окремих фрагментів коду, що полегшує розуміння логіки програми іншими розробниками.
* Діаграми - графічне подання структури та взаємозв'язків в системі.
* Документація API – містить інструкції, які запити доступні, їх призначення, параметри та формат відповіді. Приклади запитів і відповідей – показує, як правильно використовувати API на практиці.

Нижче подано документацію API сформовану сервісом Redocly завдяки файлу *openapi.yaml,* який можна побачити в «[*Додатку Б*](#_Додаток_Б)»

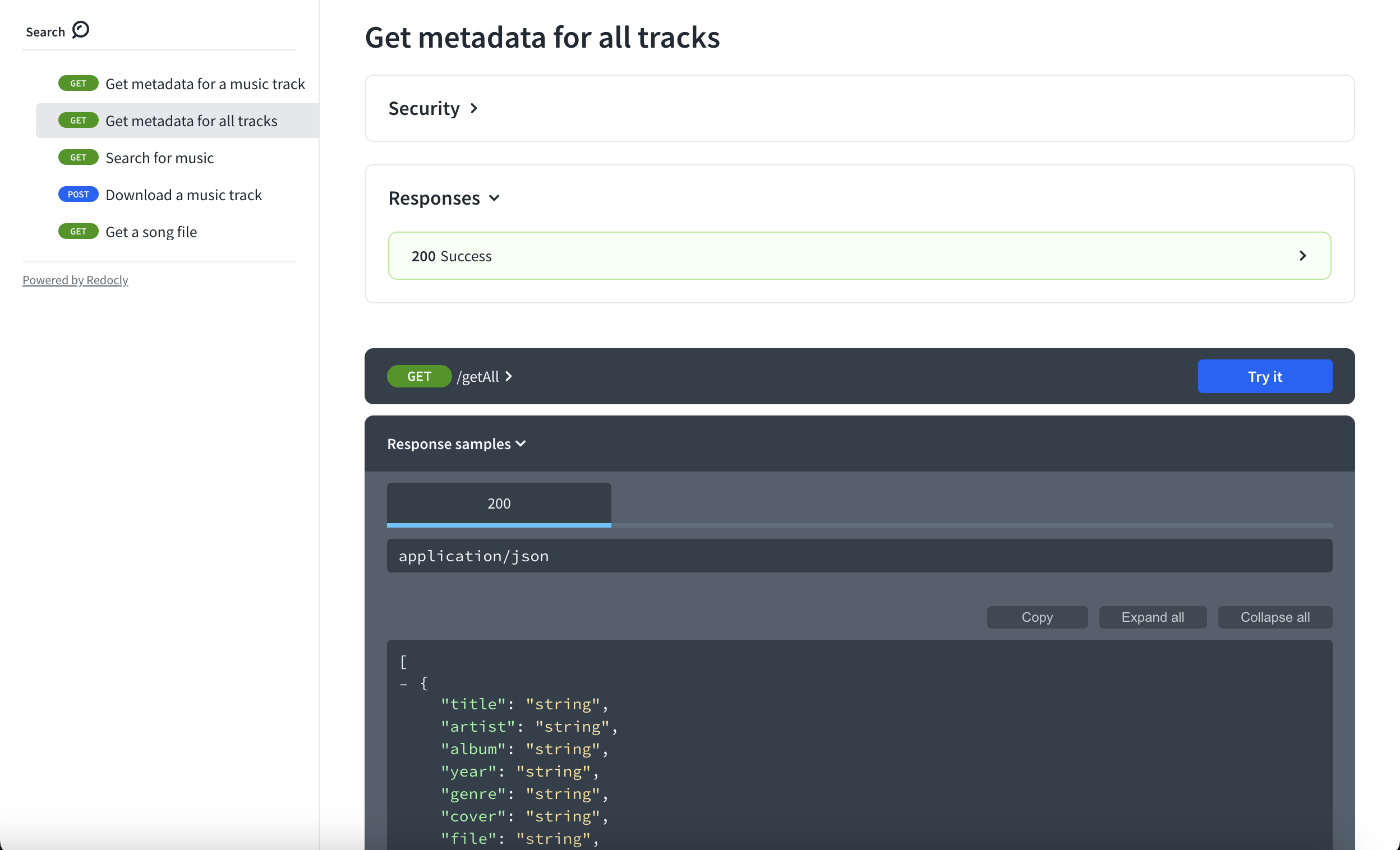


Рис. 6.1 Документація API створена завдяки сервісу Redocly.

Така документація дозволяє новим розробникам швидко розібратися в програмному коді та архітектурі системи. А користувачі можуть легко навчитися працювати з додатком за допомогою посібника.

# Висновки

Під час розробки цього додатку було проведено аналіз сучасного стану питання, визначено завдання на роботу та проведено докладний огляд сучасних технологій та інструментів для проектування програмного забезпечення. Була вивчена універсальна мова проектування UML та ії застосування для створення моделей системи. В рамках проекту розроблена загальна структура програми, проведений аналіз функцій системи та створена ієрархія класів.

Також була проведена декомпозиція системи та аналіз взаємозв'язків між об'єктами. Були розроблені інтерфейси класів та створена UML-діаграма класів для кращого розуміння структури системи.

Одним із ключових аспектів було використання паттернів проектування, які дозволили ефективно організувати структуру програми та забезпечити легкість управління функціональністю. Паттерн стратегії (Strategy) визначив родину алгоритмів для пошуку та завантаження музики, паттерн фабричного методу (Factory Method) використовувався для динамічного створення об'єктів інтерфейсу користувача, паттерн компонувальника (Composite) допоміг відобразити ієрархію інтерфейсів, а паттерн команди (Command) дозволив інкапсулювати та обробляти запити користувача.

Отже, в коді широко застосовуються базові паттерни ООП для розділення логіки програми на незалежні частини з чіткими інтерфейсами та слабкозв'язаною архітектурою.

Після цього була уточнена постановка задачі на розробку програмного забезпечення та написані тести для перевірки модуля на завантаження музики. Завершальним етапом була розробка документації для супроводження програмного продукту, яка включала в себе технічну документацію, інструкції користувача.

Всі ці етапи спрямовані на створення високоякісного програмного продукту, який задовольнить потреби користувачів та буде легко супроводжуватися та розвиватися в майбутньому. Розробка програмного забезпечення є процесом, що вимагає уваги до деталей, досліджень та системного підходу, і цей проект не є винятком.

Нехай цей додаток стане корисним та піднесе радість людям, які будуть ним користуватися.

# Бібліографічний список

1. Мірцева О. В. "Основи об’єктно-орієнтованого програмування в Python" - Київ: БІНОМ. Лабораторія знань, 2018. - 256 с.
2. Шведов Д. В. "Універсальна мова проектування UML. Посібник для практичних занять" - Київ: Видавництво Київського університету, 2020. - 288 с.
3. Умаров Р. Ш., Васильев А. В. "Розробка сучасних додатків на Python" - Київ: Пітер, 2018. - 416 с.
4. Гаммель Г. Г. "Python 3 і PyQt 5. Розробка додатків" - Київ: КВІНТА-ДАНА, 2020. - 400 с.
5. Сметана Л. "Мова програмування Python" - Київ: Діалектика, 2019. - 328 с.
6. Буч Г. "Об'єктно-орієнтоване програмування. Сучасні методи проектування" - Київ: Діалектика, 2017. - 704 с.
7. Fowler M., Beck K., Brant J., Opdyke W., Roberts D. "Рефакторинг. Покращення структури і архітектури коду" - Київ: Діалектика, 2019. - 480 с.
8. Хант Э., Томас Дж. "Персональне програмування на Python" - Київ: Видавництво Манн, Іванов і Фербер, 2021. - 432 с.
9. Орелли Ш. "Професійна розробка на Python. Як стати першокласним розробником" - Київ: Пітер, 2017. - 256 с.
10. Бек Кент. "Екстремальне програмування. Визначені і постійні правила в розробці програмного забезпечення" - Київ: Діалектика, 2018. - 352 с.

# Додаток А

Основні методи програми

*Main.py*

from UI import selectUI

def main() -> None:

# Створюємо екземпляр класу CommandLineUI

ui = selectUI()

# Запускаємо інтерфейс

ui.run()

if \_\_name\_\_ == "\_\_main\_\_":

main()

*UI.py*

*…*

def selectUI():

while True:

try:

choice = int(input("Виберіть тип інтерфейсу:\n1 - командний рядок,\n2 - веб,\n3 - графічний,\n4 - Telegram: "))

if choice == 1:

return CommandLineUI()

elif choice == 2:

return WebUI()

elif choice == 3:

return GUI()

elif choice == 4:

return TelegramUI()

else:

print("Недійсний вибір. Спробуйте ще раз.")

except ValueError:

print("Недійсний вибір. Введіть номер інтерфейсу.")

# Додаток Б

Документація API за стандартом OpenAPI 3.0

*openapi.yaml*

openapi: 3.0.0

info:

title: Music Downloader API

description: API for downloading and listening to music

version: 1.0.0

servers:

- url: http://localhost:5001

security:

- apiKey: []

paths:

/getMusicData:

get:

operationId: getMusicMetadata

security:

- apiKey: []

summary: Get metadata for a music track

parameters:

- name: pos

in: query

description: The position of the track

required: false

schema:

type: integer

default: 0

responses:

200:

description: Success

content:

application/json:

schema:

type: object

properties:

title:

type: string

artist:

type: string

album:

type: string

year:

type: string

genre:

type: string

cover:

type: string

file:

type: string

max:

type: integer

/getAll:

get:

operationId: getAllMusicMetadata

security:

- apiKey: []

summary: Get metadata for all tracks

responses:

200:

description: Success

content:

application/json:

schema:

type: array

items:

type: object

properties:

title:

type: string

artist:

type: string

album:

type: string

year:

type: string

genre:

type: string

cover:

type: string

file:

type: string

max:

type: integer

# Додаток В

Повний код програми

Повний код програми можна отримати за посиланням: <https://github.com/igordmitrow/MusicDownloader>