МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ТЕЛЕГРАМ БОТУ ДЛЯ ПОШУКУ МУЗИКИ ЗАСОБАМИ PYTHON ТА MONGODB»**

Виконав: студент 4 курсу, групи 6.1211-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Н.С. Мотрій
(ініціали та прізвище)
Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, д.т.н., професор Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Мотрію Назару Сергійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка телеграм боту для пошуку музики засобами Python та MongoDB

керівник роботи Лісняк Андрій Олександрович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд теоретичних властивостей та планування проєкту.
2. Проєктування та аналіз вимог до онлайн чату.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	15.01.2025	
2.	Збір вихідних даних.	10.02.2025	
3.	Обробка методичних та теоретичних	10.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	07.04.2025	
5.	Розробка третього розділу.	19.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2025	

Студент _____ Н.С. Мотрій
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка телеграм боту для пошуку музики засобами Python та MongoDB»: 49 с., 18 рис., 2 табл., 13 джерел.

БАЗА ДАНИХ, ПОШУК МУЗИКИ, ПРОГРАМНА ІНЖЕНЕРІЯ, РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, TELEGRAM-БОТ, API, MONGODB, PYTHON.

Об’єкт дослідження – процес автоматизації пошуку музичного контенту з використанням чат-ботів у месенджері Telegram.

Предмет дослідження – методи та засоби розробки програмного забезпечення для інтерактивного пошуку музики з використанням мови програмування Python та бази даних MongoDB.

Мета роботи – створити телеграм-бота, який надає користувачам зручний інтерфейс для пошуку музики за назвою, виконавцем або жанром, використовуючи зовнішні API та базу даних для зберігання запитів і результатів.

Методи дослідження – аналіз вимог до функціональності телеграм-бота, застосування принципів REST-архітектури для інтеграції з музичними сервісами, проєктування структури бази даних MongoDB, побудова діаграм UML та реалізація системи за допомогою Python.

У процесі реалізації проєкту було виконано аналіз предметної області та сформульовано функціональні вимоги. Розроблено архітектуру системи у вигляді UML-діаграм, спроєктовано структуру сховища даних у MongoDB. Бот реалізований на Python з використанням Telegram Bot API та інтеграцією зі сторонніми музичними API (наприклад, Last.fm або Deezer API).

В результаті розробки отримано готове до використання програмне забезпечення у вигляді телеграм-бота, який забезпечує ефективний пошук музичного контенту та зберігає інформацію про взаємодію з користувачами.

SUMMARY

Bachelor’s qualifying paper “Development of a Telegram Bot for Music Search Using Python and MongoDB”: 49 pages, 18 figures, 2 tables, 13 references.

API, DATABASE, MONGODB, MUSIC SEARCH, PYTHON, SOFTWARE DEVELOPMENT, SOFTWARE ENGINEERING, TELEGRAM BOT.

The object of the study is the process of automating music content search using chatbots in the Telegram messenger.

The subject of the research is the methods and tools for developing software for interactive music search using the Python programming language and MongoDB database.

The aim of the study is to create a Telegram bot that provides users with a convenient interface for searching music by title, artist, or genre, using external APIs and a database for storing queries and results.

The research methods include analyzing the functional requirements of the Telegram bot, applying REST architecture principles for integration with music services, designing the MongoDB database structure, building UML diagrams, and implementing the system using Python.

During the development process, an analysis of the subject area was conducted, and functional requirements were defined. The system architecture was developed in the form of UML diagrams, and the data storage structure in MongoDB was designed. The bot was implemented in Python using the Telegram Bot API and integrated with third-party music APIs (such as Last.fm or Deezer API).

As a result of the development, a fully functional Telegram bot was created that enables efficient music content search and stores information about user interactions.

ЗМІСТ

Завдання на кваліфікаційну роботу……...2
Реферат 4
Summary 5
Вступ..... 7
1 Огляд теоретичних властивостей та планування проєкту 9
1.1 Опис предметної області 9
1.2 Python та його можливості 10
1.3 MongoDB та його можливості 11
1.4 Функціональні та нефункціональні вимоги 13
1.5 Висновки до розділу 1 15
2 Проєктування та аналіз вимог до онлайн чату..... 17
2.1 Концептуальне проєктування інтерфейсу 17
2.2 Вимоги до інтеграції з музичними API..... 22
2.3 Налаштування серверної частини та запитів АПІ 25
2.4 Висновки до розділу 2 29
3 Реалізація та тестування 31
3.1 Розробка серверної частини з використанням Telegram Bot..... 31
3.2 Налаштування бази даних MongoDB..... 34
3.3 Підключення зовнішніх музичних API..... 36
3.4 Тестування функціоналу та обробка типових сценаріїв..... 39
3.5 Висновки до розділу 3 45
Висновки 46
Перелік посилань..... 48

ВСТУП

У сучасному цифровому світі стрімкий розвиток технологій відкриває нові можливості для споживання та пошуку музичного контенту. З огляду на популярність месенджерів, особливо Telegram, користувачі очікують отримувати потрібну інформацію швидко та зручно. Одним із перспективних рішень у цьому напрямку є створення Telegram-ботів, які надають можливість інтерактивного пошуку музики без необхідності використовувати окремі застосунки або вебплатформи.

Мета – створення Telegram-боту для пошуку музики, який дозволяє користувачам шукати композиції за назвою, виконавцем або жанром, використовуючи зручний інтерфейс у месенджері. Бот також може надавати рекомендації, історію запитів і популярні треки.

Для досягнення цієї мети було поставлено наступні завдання:

– проаналізувати потреби користувачів у контексті пошуку музики через месенджери;
– дослідити особливості Telegram Bot API та вибрати оптимальний спосіб реалізації функціоналу;
– спроєктувати структуру бази даних на MongoDB для зберігання інформації про треки, виконавців і історію запитів;
– реалізувати серверну частину на Python із використанням бібліотеки python-telegram-bot та pymongo;
– протестувати функціонал бота на типових сценаріях взаємодії користувачів.

Об’єкт дослідження – процес взаємодії користувача з Telegram-ботом для музичного пошуку.

Предмет дослідження – сукупність методів та засобів створення програмного забезпечення на Python із використанням Telegram API та бази даних MongoDB для реалізації функціоналу пошуку музики.

Методи дослідження – аналіз вимог, моделювання, проєктування, побудова діалогових сценаріїв, REST API та зберігання даних у NoSQL базі.

Для реалізації проєкту обрано стек технологій, який включає:

– Python – як основна мова програмування;
– MongoDB – як NoSQL база даних для зберігання інформації про музичний контент;
– Telegram Bot API – для взаємодії з користувачами через месенджер;
– Heroku або Railway – для розгортання бота у хмарі.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та переліку посилань. Загальний обсяг роботи становить 49 сторінок. У переліку посилань – 13 джерел.

У першому розділі наведено теоретичний огляд чат-ботів, історію їх розвитку, особливості Telegram-ботів, аналіз використовуваних технологій і планування етапів проєкту.

Другий розділ присвячено проєктуванню Telegram-бота, аналізу функціональних і нефункціональних вимог, розробці загальної архітектури, а також структурі бази даних.

У третьому розділі описано процес реалізації Telegram-бота, організацію роботи з API Telegram, взаємодію з MongoDB, обробку запитів користувачів, а також проведення тестування функціоналу. Розглянуто сценарії використання та оцінено ефективність рішення.

1 ОГЛЯД ТЕОРЕТИЧНИХ ВЛАСТИВОСТЕЙ ТА ПЛАНУВАННЯ ПРОЄКТУ

1.1 Опис предметної області

У сучасному цифровому середовищі чат-боти виступають як ефективний інструмент взаємодії між користувачем і системою. Вони здатні автоматизувати численні рутинні завдання, забезпечувати підтримку в режимі 24/7, покращувати клієнтський досвід і зменшувати навантаження на персонал. Предметна область, що розглядається в цьому проєкті, стосується створення інтелектуального Telegram-бота, який виконує функції консультанта або помічника в конкретній сфері діяльності (наприклад, освітній, медичній, комерційній, сервісній тощо) [1].

Основною метою є забезпечення інтерактивної взаємодії з кінцевим користувачем через інтерфейс месенджера Telegram. Це дозволяє значно знизити бар’єр входу, оскільки не потребує окремого застосунку чи вебсайту, а використовується знайоме середовище спілкування. Такі боти можуть надавати інформаційні послуги, приймати заявки, проводити опитування, відповідати на поширені запитання, працювати з базами даних або навіть інтегруватися з іншими цифровими сервісами (CRM, ERP, платіжні системи, API сторонніх платформ) [2].

Особливістю предметної області є висока гнучкість і вимога до адаптивної логіки: бот повинен розуміти запити користувача, реагувати на події, обробляти дані, зберігати історію сесій і діяти в залежності від контексту розмови. Також важливим аспектом є реалізація безпечної роботи з персональними даними, відповідність вимогам приватності, а також стійкість до помилкових або неочікуваних дій з боку користувача [3].

У межах цього проєкту реалізується Telegram-бот, який взаємодіє з базою даних MongoDB (або PostgreSQL), має зручну структуру команд і здатний працювати як в режимі меню, так і через обробку текстових

повідомлень. Його логіка написана на мові програмування Python з використанням відповідних бібліотек для інтеграції з Telegram API. Бот може зберігати користувацькі дані, адаптувати поведінку залежно від ролі (наприклад, користувач або адміністратор), а також підтримувати багатомовний інтерфейс [4].

Розробка таких рішень є актуальною як для малого та середнього бізнесу, так і для некомерційних організацій, державних установ чи освітніх платформ. Автоматизація спілкування через ботів відкриває нові можливості для оптимізації процесів, підвищення доступності інформації та покращення загальної ефективності цифрових сервісів.

1.2 Python та його можливості

Python – це високорівнева мова програмування загального призначення, яка вирізняється простим синтаксисом, зрозумілою структурою коду та широким спектром застосувань. Вона підходить як для початківців, так і для досвідчених розробників, забезпечуючи зручність при створенні як простих, так і комплексних програмних рішень [13].

Python застосовується в багатьох сферах, зокрема у веброзробці, автоматизації, обробці даних, машинному навчанні, а також при створенні чат-ботів. Його гнучкість пояснюється підтримкою як об’єктно-орієнтованого, так і функціонального програмування, а також наявністю великої кількості готових бібліотек, які полегшують і прискорюють розробку [5].

Зокрема, у розробці Telegram-ботів Python є одним із найзручніших інструментів завдяки таким бібліотекам, як python-telegram-bot, aiogram чи telebot, що дозволяють взаємодіяти з Telegram API для обробки повідомлень, команд, клавіатур та інших функцій. Це робить мову зручною для реалізації ботів із доступом до зовнішніх джерел, баз даних або інших сервісів [6].

MongoDB, як документоорієнтована база даних, добре інтегрується з Python через бібліотеку PyMongo. Вона забезпечує зберігання та обробку структурованої інформації, що особливо зручно для зберігання даних користувачів, історії запитів або результатів пошуку [6].

Python також надає потужні можливості для обробки природної мови (NLP), що може бути корисним у ботах, пов’язаних з пошуком текстової інформації. Інструменти, такі як spaCy, TextBlob, nltk або моделі з бібліотеки Transformers, дозволяють здійснювати семантичний аналіз, класифікацію запитів та обробку текстових відповідей [7].

У контексті вебвзаємодії Python підтримує створення REST API через фреймворки Flask, FastAPI чи Django, що дозволяє реалізовувати інтеграцію з музичними сервісами або базами даних. Також підтримується асинхронне програмування через asyncio або aiohttp, що важливо для обробки великої кількості одночасних запитів [7].

Для перевірки коректності роботи застосунку Python надає інструменти тестування, зокрема unittest і pytest, які дозволяють валідувати логіку обробки запитів та взаємодію з базою даних.

Завдяки цим перевагам Python та MongoDB є доцільним вибором для розробки Telegram-бота, який виконує пошук музики, зберігає запити користувачів та працює з зовнішніми API [7].

1.3 MongoDB та його можливості

MongoDB – це документоорієнтована система керування базами даних, що належить до категорії NoSQL. Вона зберігає дані у вигляді документів у форматі BSON (розширення JSON), що забезпечує гнучкість у структурі та зручність масштабування. Такий формат зберігання є придатним для сучасних вебзастосунків, API-сервісів і систем з динамічно змінними даними [7].

Відсутність жорсткої схеми є однією з ключових особливостей MongoDB. Це дає змогу зберігати документи з різною структурою в межах

однієї колекції, що спрощує розробку застосунків із нерегламентованими або змінними даними, як-от Telegram-боти чи прототипи систем. Такий підхід зменшує кількість необхідних попередніх налаштувань і дозволяє швидко змінювати модель даних у процесі розробки.

MongoDB часто використовується для зберігання інформації про користувачів, повідомлення, історію запитів і налаштування. У випадку з Telegram-ботами, документи можуть містити user_id, стан взаємодії, параметри пошуку або тимчасові дані діалогу, що забезпечує ефективне управління контекстом [7].

MongoDB добре поєднується з Python через бібліотеку pymongo, а також підтримує використання об’єктно-документоорієнтованих маперів (ODM), наприклад mongoengine. Це дозволяє реалізовувати CRUD-операції, пошук, агрегації й оновлення даних без використання SQL, що може спростити реалізацію логіки у проєктах з обмеженим обсягом структурованих запитів.

Серед додаткових функцій MongoDB варто відзначити підтримку транзакцій, TTL-колекцій (для автоматичного видалення старих записів), повнотекстовий пошук, геозапити, індексацію даних, реплікацію, а також наявність хмарної платформи MongoDB Atlas. Це дає змогу масштабувати систему, керувати інфраструктурою у хмарному середовищі та підвищити доступність сервісу [8].

Механізм агрегацій у MongoDB дозволяє виконувати складні операції обробки даних, як-от фільтрація, сортування, групування або підрахунок – без потреби виносити логіку на рівень застосунку. Це може бути корисно для отримання статистики використання бота, аналізу частоти запитів або підготовки звітів.

Індексування, включаючи текстові індекси, дозволяє оптимізувати швидкість пошуку. Це особливо актуально в проєктах, де реалізується функція пошуку за ключовими словами, як-от у випадку Telegram-бота для пошуку музики [8].

MongoDB також може інтегруватися з іншими компонентами архітектури, зокрема системами кешування (Redis), чергами повідомлень

(RabbitMQ), брокерами подій (Kafka), що забезпечує можливість створення асинхронних та масштабованих рішень. Це важливо в системах, де обробка запитів відбувається паралельно або розділена між кількома сервісами.

У поєднанні з Python MongoDB забезпечує ефективну обробку подій Telegram API та дозволяє гнучко зберігати й обробляти динамічні дані, що виникають у процесі взаємодії з користувачем.

Загалом, MongoDB є придатним інструментом для зберігання та обробки інформації в реальному часі, особливо у випадках, коли потрібна висока гнучкість структури даних. У межах розробки Telegram-бота для пошуку музики MongoDB дозволяє ефективно зберігати параметри пошуку, результати запитів і взаємодії користувачів із системою [8].

1.4 Функціональні та нефункціональні вимоги

Функціональні вимоги описують основні завдання, які Telegram-бот повинен виконувати у межах предметної області. Вони визначають поведінку системи під час взаємодії з користувачем, адміністратором і зовнішніми сервісами, забезпечуючи реалізацію ключових сценаріїв використання. Саме на основі функціональних вимог формується структура роботи бота, логіка обробки запитів, формування відповідей, доступ до баз даних та адміністративних інструментів [8].

У таблиці 1.1 наведено основні функціональні можливості Telegram-бота, які визначають його поведінку під час взаємодії з користувачами, адміністратором і зовнішніми сервісами. Ці функції охоплюють широкий спектр завдань, зокрема автоматичну ідентифікацію та збереження користувачів, підтримку структурованого меню й текстових команд, динамічну генерацію відповідей залежно від запиту, логування дій користувачів, а також доступ до адміністративного функціоналу. Крім того, важливою є інтеграція з базами даних для збереження інформації, підтримка

мультимедійного контенту, обробка помилок і можливість масштабування системи.

Таблиця 1.1 – Функціональні вимоги

№	Функція	Опис
1	Збереження та ідентифікація користувачів	Автоматична реєстрація ID, імені, username і дати першого звернення користувача
2	Структуроване меню та текстові команди	Підтримка команд типу /start, /help, /menu та інших для навігації
3	Динамічна генерація відповідей	Відповіді формуються на основі запиту користувача та даних з бази
4	Логування взаємодій	Збереження історії дій користувача для аналітики або відновлення сесії
5	Адміністративний функціонал	Обмежений доступ до керування контентом, розсилки, статистики тощо
6	Мультимовний інтерфейс (опційно)	Можливість вибору мови (наприклад, українська, англійська) через меню налаштувань
7	Інтеграція з базою даних	Підключення до MongoDB, PostgreSQL або іншої СКБД для зберігання інформації
8	Надсилання мультимедійного контенту	Підтримка зображень, відео, документів, повідомлень з форматуванням
9	Обробка помилок та недійсних запитів	Оброблення винятків, інформування користувача у разі помилкових або незрозумілих дій
10	Масштабованість і розширюваність	Легка інтеграція нових функцій без перебудови всієї системи або зупинки сервісу

Нефункціональні вимоги відіграють важливу роль у процесі розробки програмного забезпечення, оскільки вони визначають, як саме повинна функціонувати система, забезпечуючи стабільність, зручність та ефективність її використання. На відміну від функціональних вимог, які описують, що система має робити, нефункціональні вимоги зосереджені на характеристиках якості, що впливають на користувацький досвід та надійність роботи системи в цілому. Вони не менш важливі, ніж функціональні, оскільки часто саме від них залежить, наскільки система буде зручною, безпечною та масштабованою.

У таблиці 1.2 наведено основні нефункціональні вимоги до Telegram-бота, які слугують орієнтирами для забезпечення його стабільної та якісної роботи [10].

Таблиця 1.2 – Опис нефункціональних вимог

Вимога	**Опис**
Надійність	Безперебійна робота 24/7. У разі помилок система автоматично обробляє винятки та відновлює роботу без втручання адміністратора.
Масштабованість	Підтримка зростаючого навантаження та додавання нових функцій без зупинки сервісу. Архітектура має бути гнучкою та адаптивною.
Продуктивність	Швидка відповідь на запити користувачів – до 1-2 секунд навіть під високим навантаженням.
Безпека даних	Захищене зберігання інформації з обмеженим доступом. Конфіденційні дані зберігаються у зашифрованому вигляді, наприклад, через AWS Secrets Manager.
Зручність	Інтерфейс простий, логічний і зрозумілий навіть для користувачів без технічного досвіду.
Сумісність	Повна відповідність Telegram Bot API. Бот працює на Android, iOS та десктопних версіях Telegram.
Розширюваність	Можливість додавання нових мов, функцій і повідомлень без значних змін у коді або структурі бази даних.

Завдяки ретельному дотриманню як функціональних, так і нефункціональних вимог, Telegram-бот зможе повною мірою виконувати поставлені завдання, гарантуючи не лише правильність і логічність своєї роботи, а й високий рівень зручності, ефективності та надійності. Такий підхід дозволяє забезпечити стабільну взаємодію з кінцевими користувачами та адміністратором, мінімізуючи можливість збоїв, затримок або помилок. Крім того, дотримання вимог щодо безпеки та масштабованості робить систему стійкою до навантажень і зовнішніх загроз, а також придатною до подальшого розвитку та розширення функціональності відповідно до зростаючих потреб або змін у середовищі використання [8].

1.5 Висновки до розділу 1

У першому розділі було розглянуто історію розвитку чат-ботів та їх важливу роль у сучасних цифрових сервісах. Чат-боти значно спростили взаємодію користувачів із різноманітними онлайн-системами, підвищили

ефективність обслуговування та відкрили нові можливості для автоматизації бізнес-процесів.

Далі були досліджені принципи роботи Telegram-ботів, зокрема способи їх інтеграції з API, що дозволяє створювати гнучкі та масштабовані системи взаємодії з користувачами. Використання Telegram Bot API забезпечує широкі можливості для розробки інтерфейсів і інтерактивних функцій.

Особливу увагу було приділено мові програмування Python, яка завдяки своїй простоті, багатству бібліотек та підтримці асинхронності є оптимальним вибором для реалізації серверної частини чат-ботів. Python дозволяє швидко створювати, тестувати та підтримувати складні системи.

Також розглянуто базу даних MongoDB і її можливості, що робить її зручною для зберігання даних у форматі документів, що гнучко масштабуються. Підтримка кешування, індексації і високої швидкодії робить MongoDB ідеальним вибором для обробки запитів чат-ботів із великою кількістю одночасних користувачів.

Таким чином, у першому розділі сформовано базове розуміння ключових технологій і концепцій, необхідних для розробки сучасного Telegram-бота з підтримкою музичних сервісів, що закладає основу для подальшої практичної реалізації.

2 ПРОЄКТУВАННЯ ТА АНАЛІЗ ВИМОГ ДО ОНЛАЙН ЧАТУ

2.1 Концептуальне проєктування інтерфейсу

Концептуальне проєктування є важливим етапом розробки телеграм-боту для пошуку музики, що дозволяє визначити основні принципи взаємодії користувачів із системою. На цьому етапі формується загальне уявлення про функціональність сервісу, що суттєво знижує ризик створення рішення, яке не відповідає очікуванням кінцевих користувачів, і задає чіткий напрям для подальшої реалізації.

На рисунку 2.1 представлено діаграму варіантів використання (прецедентів) телеграм-боту для пошуку музики, яка ілюструє ключові функціональні можливості системи, а також взаємодію між основними типами користувачів, зокрема звичайними користувачами, адміністраторами та музичними редакторами.



Рисунок 2.1 – Діаграма прецедентів

Представлена діаграма прецедентів відображає ключові варіанти використання системи «Телеграм-бот для пошуку музики» та взаємодію трьох основних типів користувачів: звичайного користувача, музичного редактора та адміністратора. Кожна з ролей має доступ до визначеного набору функціональних можливостей бота відповідно до своїх прав і завдань.

Користувач – це кінцевий користувач телеграм-бота, який взаємодіє з системою з метою пошуку музичних композицій. Основні варіанти використання для цієї ролі включають пошук музики за назвою або виконавцем, прослуховування доступного фрагменту треку, додавання треків до вибраного, перегляд історії запитів та можливість залишити відгук або оцінку.

Музичний редактор – користувач, який наповнює базу даних контентом. Йому доступні функції створення нових записів у базі (додавання треків із зазначенням назви, виконавця, жанру, посилання на аудіо), редагування існуючих треків, а також видалення застарілої або невідповідної інформації. Крім того, редактор може реагувати на зворотний зв’язок користувачів щодо якості або актуальності музичних даних.

Адміністратор відповідає за загальне управління ботом. Його прецеденти охоплюють перегляд статистики використання системи (наприклад, кількість пошукових запитів, популярні треки), керування обліковими записами редакторів, призначення ролей і блокування порушників. Також адміністратор має можливість модерувати відгуки, керувати глобальними налаштуваннями бота, зокрема обмеженнями на кількість запитів або доступністю функцій.

Таким чином, діаграма прецедентів забезпечує цілісне уявлення про взаємодію користувачів із телеграм-ботом та функціональні можливості системи, що є основою для подальшого технічного проєктування та розробки архітектури.

При проєктуванні програмного забезпечення критичну роль відіграє прототипування основних сценаріїв використання, що дозволяє на ранніх

етапах візуалізувати логіку роботи бота, оптимізувати взаємодію з користувачем і запобігти помилкам під час реалізації функціоналу.

Для розробки телеграм-бота для пошуку музики було створено кілька ключових прототипів інтерфейсу взаємодії з користувачем – початкове привітальне повідомлення, меню з основними командами, інтерфейс пошуку музики за ключовими словами, а також блок з результатами пошуку, що включає кнопку для прослуховування треку та можливість додати його до вибраного.

На рисунку 2.2 представлено макет стартового вікна бота, яке є центральною точкою входу в систему. Воно містить навігаційне меню з командами (наприклад, /search, /favorites, /history), коротку інструкцію з використання, а також кнопки швидкого доступу до основних функцій для зареєстрованих користувачів.

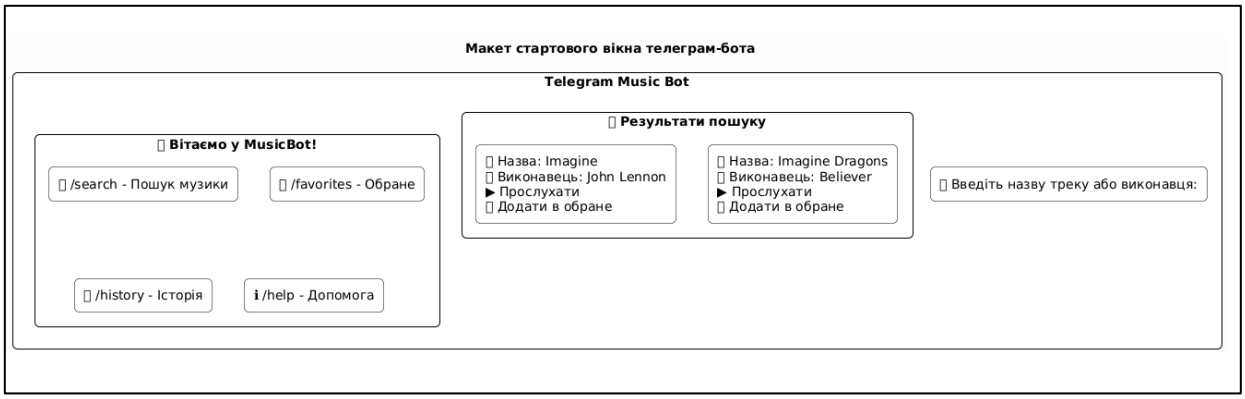


Рисунок 2.2 – Макет стартового вікна

Відповідно до наведеного макету, користувача зустрічає привітальне повідомлення та основні інформаційні блоки. У верхній частині інтерфейсу телеграм-бота розташоване меню з основними командами: «Пошук музики», «Обране», «Історія», «Допомога». Після цього користувач отримує інструкцію щодо початку пошуку та поле для введення запиту (назва треку або виконавець). Нижче розміщуються блоки з результатами пошуку, які містять назву пісні, виконавця, кнопку для прослуховування та можливість додати композицію до обраного.

Такий підхід забезпечує зручну та швидку взаємодію з ботом, а також дозволяє користувачеві легко знаходити та зберігати улюблену музику. Основним функціональним компонентом бота є система пошуку треків, макет якої наведено на рисунку 2.3.

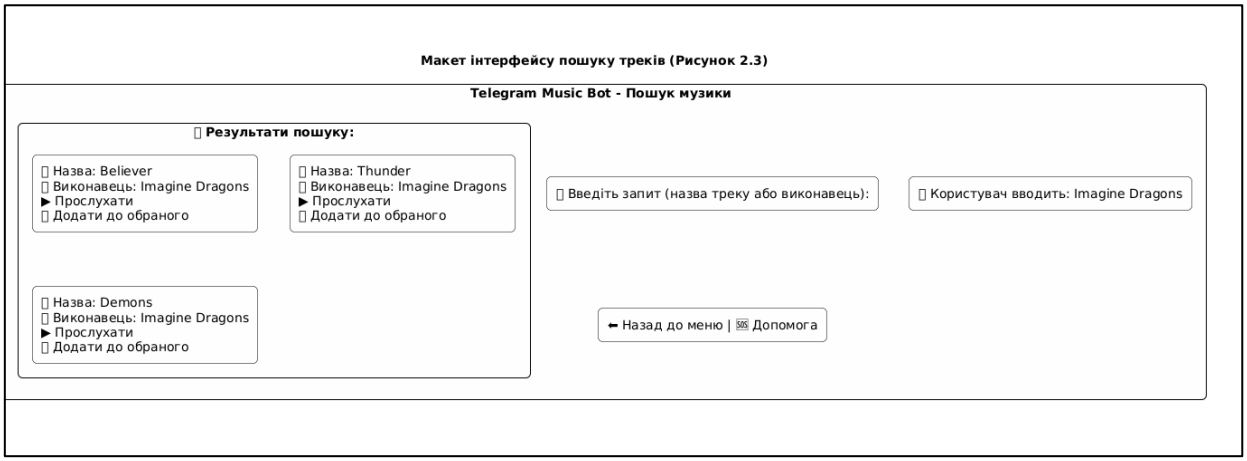


Рисунок 2.3 – Макет інтерфейсу пошуку

Компонентами інтерфейсу екрану реєстрації в телеграм-боті для пошуку музики є логотип або назва бота, які виводяться у верхній частині чату з метою впізнаваності та ідентифікації сервісу. Основну частину інтерфейсу займає покрокова форма для створення нового облікового запису. Користувачеві послідовно пропонується ввести ім’я, прізвище, електронну пошту або Telegram-нікнейм, а також придумати пароль.

Після заповнення усіх даних бот відображає кнопку підтвердження «Зареєструватися», яка активує процес створення нового облікового запису в базі даних MongoDB. Для зручності реалізоване посилання-команда на вхід (наприклад, /login) для тих користувачів, які вже зареєстровані.

Інтерфейс реалізовано у лаконічному текстовому стилі, з чіткими інструкціями та підказками на кожному етапі. Повідомлення про помилки або некоректне введення (наприклад, неправильний формат email) виводяться одразу після відповіді користувача. Ключові елементи, такі як кнопки або запити, виділяються за допомогою емодзі або форматування, що підвищує зручність взаємодії.

На рисунку 2.4 зображено діаграму розгортання телеграм-бота, розробленого для пошуку музики засобами Python та MongoDB. Діаграма ілюструє основні апаратно-програмні компоненти системи, включно з Telegram-сервером, серверною частиною бота, базою даних MongoDB та взаємозв’язками між ними в середовищі розгортання.

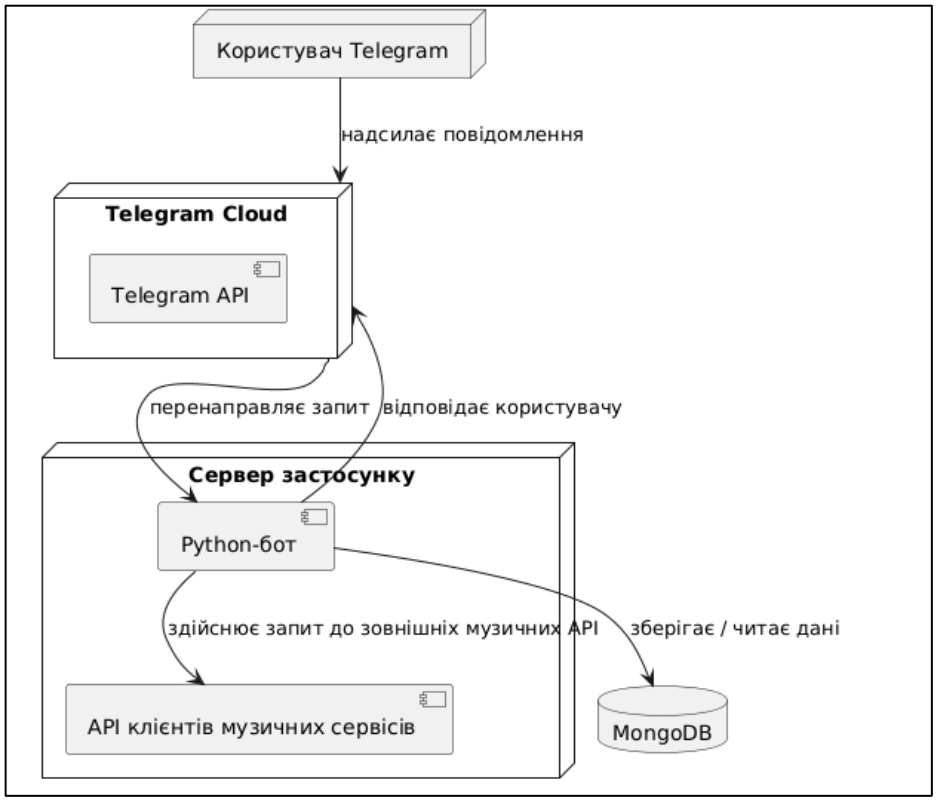


Рисунок 2.4 – Діаграма розгортання [9]

Система базується на сучасному технологічному стеку, до якого входять Python для розробки логіки телеграм-бота, бібліотека aiogram для взаємодії з Telegram API та MongoDB як нереляційна база даних для зберігання інформації про користувачів, музичні запити та історію пошуку.

Бот працює на сервері, обробляючи повідомлення користувачів у Telegram. Він відповідає за прийом запитів на пошук музики, взаємодію з зовнішніми музичними API для отримання результатів, а також збереження й оновлення даних у базі MongoDB.

Користувачі взаємодіють із системою через мобільний або десктопний клієнт Telegram, надсилаючи повідомлення та отримуючи відповіді в чаті з

ботом. Вся взаємодія відбувається через захищене з’єднання Telegram, що забезпечує безпеку переданих даних.

2.2 Вимоги до інтеграції з музичними API

Інтеграція Telegram-бота з музичними API є важливою складовою реалізації його функціональності. Цей компонент дає змогу отримувати інформацію про композиції, альбоми, виконавців, рейтинги або навіть уривки пісень для відтворення, що значно розширює можливості бота та робить його корисним для користувача.

Перш за все, API, які планується використовувати (наприклад, Spotify API, Last.fm API, Deezer API чи інші), повинні мати відкриту документацію, підтримувати REST-запити, мати можливість авторизації через OAuth 2.0 та обмеження запитів у межах допустимих лімітів. Це дозволяє забезпечити стабільну та безпечну взаємодію між Telegram-ботом і зовнішнім сервісом.

Система має підтримувати отримання інформації за такими параметрами, як назва треку, ім’я виконавця, ID пісні, жанр або ключове слово. У запиті користувача ці параметри повинні коректно розпізнаватися та передаватися до API, а у відповіді – відображатися у зручному форматі.

Telegram-бот повинен обробляти API-відповіді в форматі JSON, аналізувати отриману інформацію та виводити користувачеві стислий і релевантний результат. Наприклад, бот може надсилати короткий опис композиції, посилання на прослуховування, обкладинку альбому та короткий фрагмент треку, якщо це дозволяє API.

Окрему увагу слід приділити обробці помилок. Якщо музичний сервіс не відповідає або користувач ввів некоректні дані (наприклад, трек не знайдено), бот має надати чітке повідомлення з пропозицією повторити запит або уточнити його. Такий підхід підвищує зручність і надійність взаємодії. Також важливо забезпечити кешування результатів запитів до музичного API,

щоб уникнути перевищення ліміту та прискорити відповіді бота при повторному зверненні до однієї й тієї ж пісні чи виконавця [13].

Інтеграція має бути розроблена таким чином, щоб у майбутньому можна було легко замінити або доповнити музичний API без суттєвих змін у логіці бота. Для цього рекомендується реалізувати окремий модуль або сервіс у структурі програми, який буде відповідати за взаємодію з музичними платформами [13].

На рисунку 2.5 зображено діаграму послідовності, що демонструє процес взаємодії користувача з Telegram-ботом під час пошуку музичної композиції за допомогою зовнішнього музичного API.

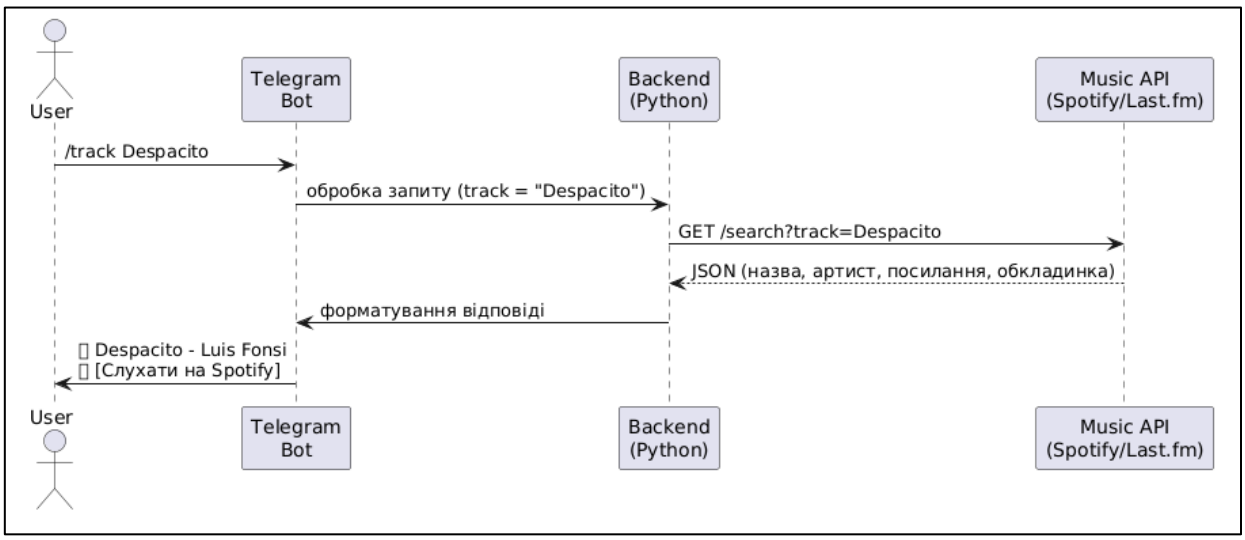


Рисунок 2.5 – Діаграма послідовності [9]

Користувач надсилає команду у вигляді текстового запиту (наприклад, /track Despacito) до Telegram-бота. Бот приймає запит і передає його на серверну частину системи для обробки. Сервер, у свою чергу, формує HTTP-запит до API музичного сервісу з відповідними параметрами пошуку.

Після успішного виконання запиту музичний API повертає відповідь у форматі JSON, яка містить релевантну інформацію: назву треку, виконавця, обкладинку альбому, посилання на прослуховування тощо. Отримані дані обробляються сервером, після чого сформована відповідь передається назад до Telegram-бота, який відправляє її користувачеві у зручному та зрозумілому

форматі. Такий підхід дозволяє забезпечити ефективну інтеграцію музичного функціоналу та високу якість обслуговування запитів користувачів [12].

На рисунку 2.6 зображено архітектуру системи Telegram-бота з підтримкою музичних API, яка демонструє основні компоненти та взаємозв'язки між ними.

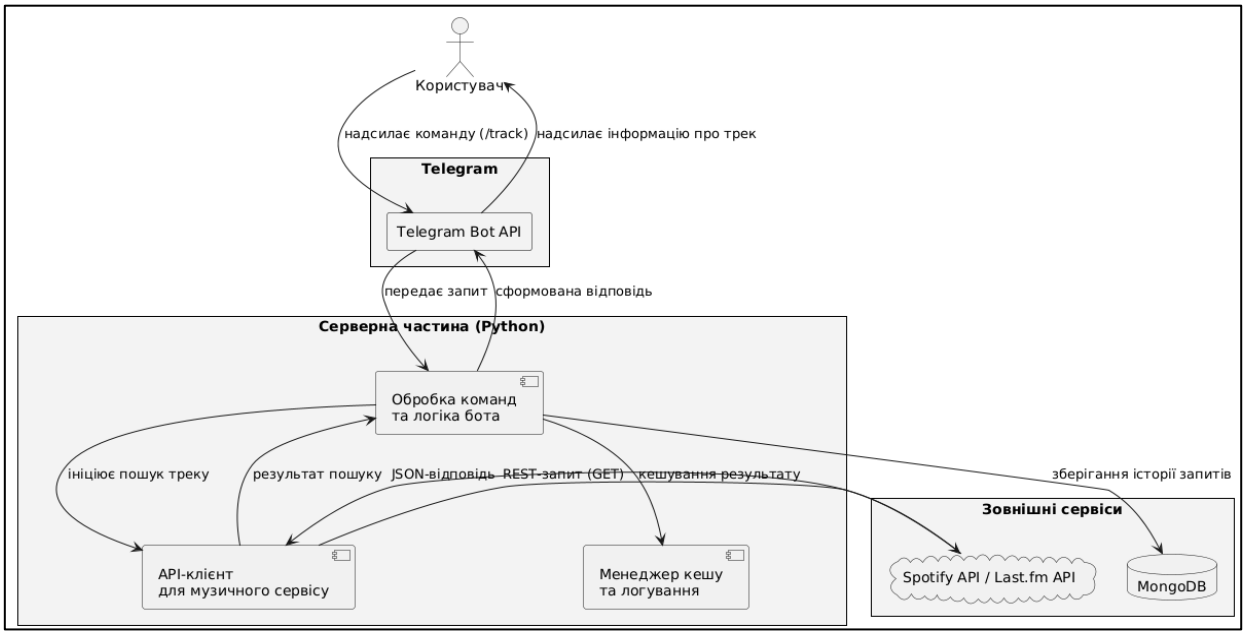


Рисунок 2.6 – Архітектура системи Telegram-бота [9]

Користувач взаємодіє з ботом через Telegram Bot API, надсилаючи текстову команду, наприклад, для пошуку музичного треку. Отримане повідомлення пересилається серверній частині, реалізованій на Python, яка містить модуль обробки логіки та команд користувача.

Далі сервер використовує API-клієнт для надсилання HTTP-запиту до зовнішнього музичного сервісу, такого як Spotify або Last.fm. Отримані дані у форматі JSON обробляються, після чого інформація кешується (для оптимізації наступних запитів) та частково зберігається в базі даних MongoDB – наприклад, для ведення історії пошуків або статистики.

Після повної обробки запиту сформована відповідь повертається через Telegram Bot API назад до користувача у зручному вигляді. Така архітектура дозволяє підтримувати масштабованість, модульність і швидку взаємодію з зовнішніми сервісами.

На рисунку 2.7 зображено приклад інтерфейсу Telegram-бота з боку користувача під час виконання запиту на пошук музичного треку.

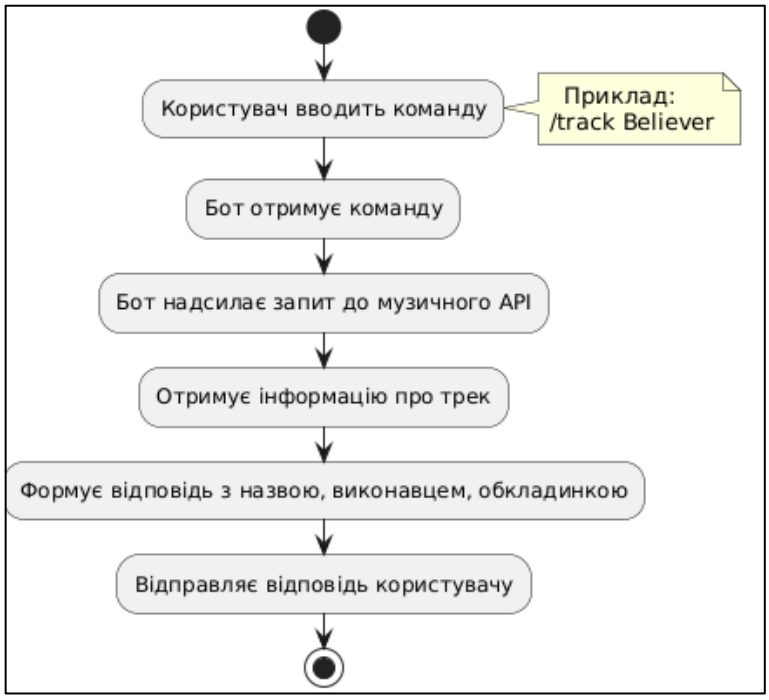


Рисунок 2.7 – Архітектура системи Telegram-бота [9]

Користувач надсилає команду /track Believer, після чого бот виконує запит до зовнішнього музичного API, отримує інформацію про композицію та повертає її у форматі, зручному для перегляду.

У відповіді бота відображається назва пісні, виконавець, обкладинка альбому (у вигляді прев'ю) та активне гіперпосилання на прослуховування треку у відповідному сервісі (наприклад, Spotify або YouTube Music). Такий інтерфейс є мінімалістичним, зручним і дозволяє навіть недосвідченому користувачеві швидко отримати бажану інформацію.

2.3 Налаштування серверної частини та запитів АПІ

Серверна частина Telegram-бота розробляється мовою програмування Python із використанням популярного мікрофреймворку Flask, який забезпечує швидке створення RESTful API для обробки запитів від Telegram

та взаємодії з зовнішніми музичними сервісами. Сервер приймає запити через HTTPS, що забезпечується або локальним проксі-сервером (наприклад, ngrok під час тестування), або розгортанням на хмарному хості з підтримкою SSL, як-от Heroku, Render, або AWS EC2.

Після надсилання повідомлення користувачем, Telegram перенаправляє його до вказаного webhook-URL. Flask-програма слухає цей маршрут і викликає відповідну функцію-обробник, яка аналізує текст команди (наприклад, /track Believer) і виконує необхідну логіку.

Для доступу до музичних сервісів використовується зовнішній API, такий як Spotify Web API або Last.fm API. Перед виконанням запиту до API необхідно пройти автентифікацію – зазвичай за допомогою токену доступу, який передається в заголовку Authorization або як параметр запиту.

На рисунку 2.8 зображено діаграму класів, яка демонструє архітектурну структуру основних компонентів серверної частини Telegram-бота, реалізованого мовою Python [10].

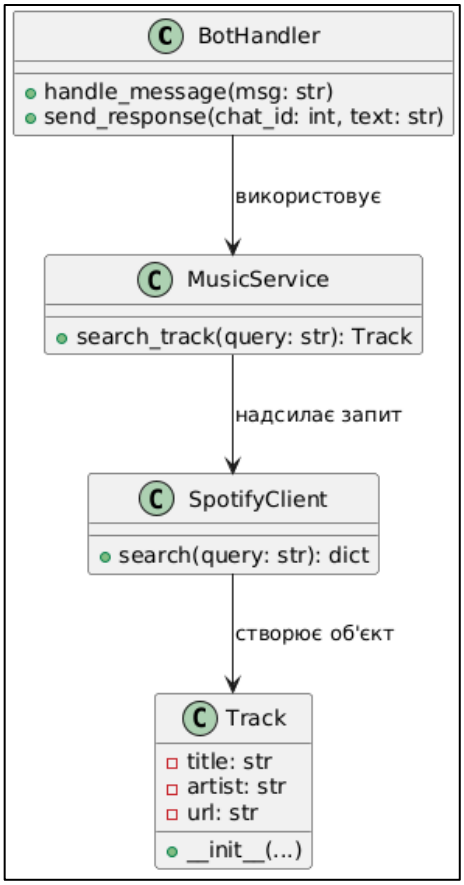


Рисунок 2.8 – Діаграма класів [9]

Основну роль у цій структурі відіграє клас BotHandler, який приймає вхідні повідомлення від Telegram через webhook та ініціює обробку запиту. Даний клас взаємодіє з MusicService – службовим класом, відповідальним за логіку пошуку треків на основі запиту користувача.

Клас MusicService у свою чергу використовує SpotifyClient, який реалізує взаємодію з зовнішнім API (у даному випадку – Spotify). Цей клієнт формує HTTP-запити до API та обробляє JSON-відповіді. Результатом такої обробки є створення об’єкта класу Track, що містить уніфіковані дані про знайдену пісню: назву, виконавця, посилання на обкладинку чи URL треку.

Такий розподіл відповідальностей забезпечує розділення логіки обробки повідомлень, роботи з API та зберігання/представлення музичної інформації, що підвищує масштабованість і підтримуваність серверної частини проєкту.

На рисунку 2.9 зображено діаграму компонентів, яка відображає взаємозв’язок між основними частинами серверної архітектури Telegram-бота. Основний функціонал системи реалізується за допомогою таких модулів: BotHandler.py, MusicService.py, SpotifyClient.py і MongoDB Storage.

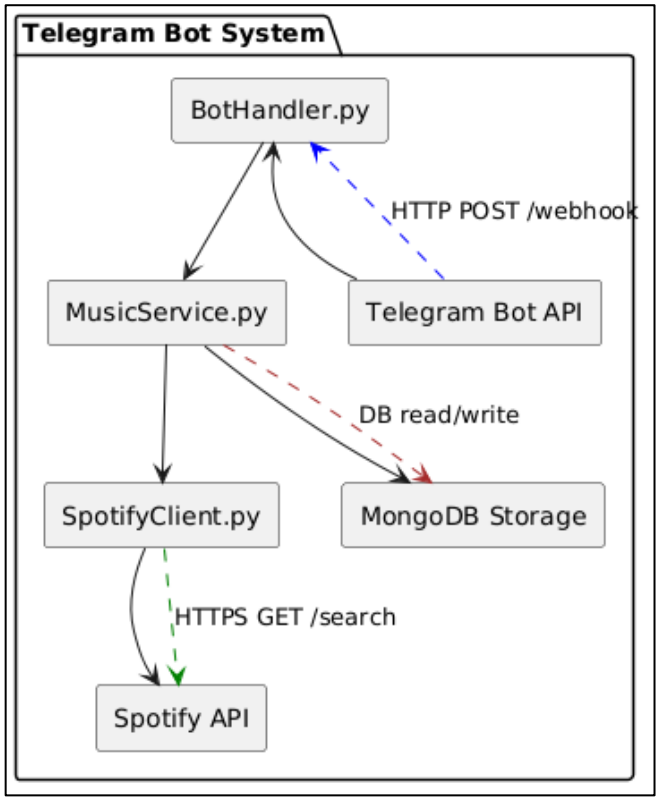


Рисунок 2.9 – Діаграма компонентів серверної частини [10]

Компонент BotHandler.py відповідає за прийом вхідних повідомлень через webhook, отриманих від Telegram Bot API. Він аналізує вміст повідомлень та ініціює виклики до відповідних службових модулів.

MusicService.py є логічним ядром системи – тут реалізована обробка команд користувача, пошук треків, форматування результатів і додаткова обробка. Цей компонент викликає SpotifyClient.py, який формує запити до зовнішнього API Spotify, обробляє відповіді та повертає результат у зручному для системи форматі. MongoDB Storage виступає в ролі внутрішнього сховища даних – використовується для кешування результатів пошуку або тимчасового збереження користувацьких даних (наприклад, історії пошуку).

Зовнішні компоненти включають Telegram Bot API (ініціює взаємодію користувача з ботом) та Spotify API, який є джерелом музичного контенту. Дана архітектура забезпечує чітке розділення відповідальностей між модулями, спрощує супровід і дозволяє масштабувати систему без суттєвих змін у загальній структурі.

На рисунку 2.10 зображено діаграму запитів, що відбуваються всередині серверної частини Telegram-бота під час обробки пошукового запиту користувача.

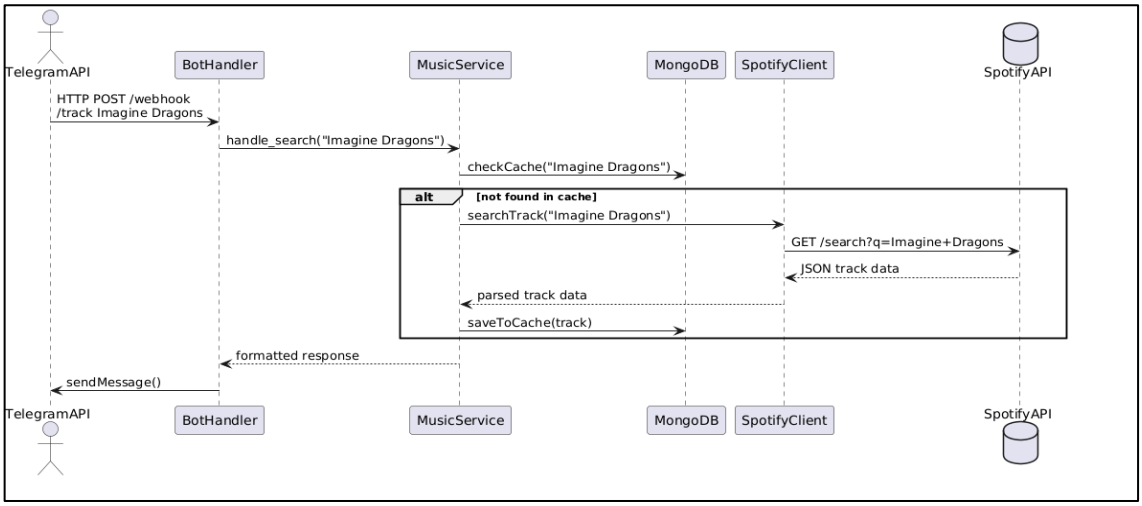


Рисунок 2.10 – Діаграма запитів серверної частини [10]

Діаграма демонструє послідовність взаємодії між внутрішніми модулями системи – від прийому запиту через webhook до отримання

результату від зовнішнього музичного API (наприклад, Spotify) і формування відповіді.

Після того як повідомлення користувача потрапляє до webhook, воно обробляється модулем BotHandler, який визначає тип запиту та ініціює виклик методу пошуку в модулі MusicService. MusicService, проаналізувавши запит, перевіряє локальне кешування через MongoDB. Якщо відповідь відсутня, викликається SpotifyClient, який формує HTTP-запит до API Spotify, надсилає його і отримує JSON-відповідь.

Після цього отримані дані форматуються в MusicService, результат зберігається в кеші MongoDB для оптимізації подальших запитів, і надсилається назад у BotHandler, де формується повідомлення у відповідь до Telegram.

Ця діаграма демонструє чітку внутрішню логіку взаємодії компонентів, орієнтовану на повторне використання, продуктивність і модульність. Такий підхід дозволяє легко масштабувати або змінювати окремі модулі без впливу на всю систему [11].

2.4 Висновки до розділу 2

У другому розділі було детально описано предметну область створення Telegram-бота для пошуку та відтворення музики з інтеграцією до сторонніх музичних API. Було визначено основні функціональні та нефункціональні вимоги, які забезпечують якісну, надійну та безпечну роботу системи.

Особлива увага була приділена технічним аспектам інтеграції з музичними сервісами, а також архітектурі серверної частини, що передбачає масштабованість, високу продуктивність і використання кешування для зниження навантаження на зовнішні API.

Було проілюстровано основні компоненти системи та взаємодію між ними за допомогою діаграм, що дозволяє краще зрозуміти структуру та логіку

роботи Telegram-бота. Налаштування серверної частини і опис запитів API дали змогу побачити, як забезпечується ефективний обмін даними між користувачем, сервером і зовнішніми сервісами.

Таким чином, у другому розділі сформовано чітке уявлення про технічні вимоги і архітектурні рішення, які ляжуть в основу практичної реалізації проєкту, забезпечуючи його стабільність, масштабованість і зручність у використанні.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Розробка серверної частини з використанням Telegram Bot

Для реалізації серверної частини Telegram-бота було обрано сучасний та продуктивний вебфреймворк FastAPI, який базується на Python. FastAPI забезпечує високу швидкість розробки, підтримку асинхронних операцій та автоматичну генерацію документації API, що особливо важливо для розробки масштабованих вебсервісів.

Однією з головних переваг FastAPI є підтримка асинхронного програмування на основі async/await, що дозволяє ефективно обробляти одночасні запити, мінімізуючи затримки при зверненнях до зовнішніх API або бази даних.

На рисунку 3.1 зображено структуру директорій та файлів серверної частини Telegram-бота. Головна директорія проєкту має назву telegram-bot-server і містить підкаталог src/, у якому розміщено основну логіку застосунку.

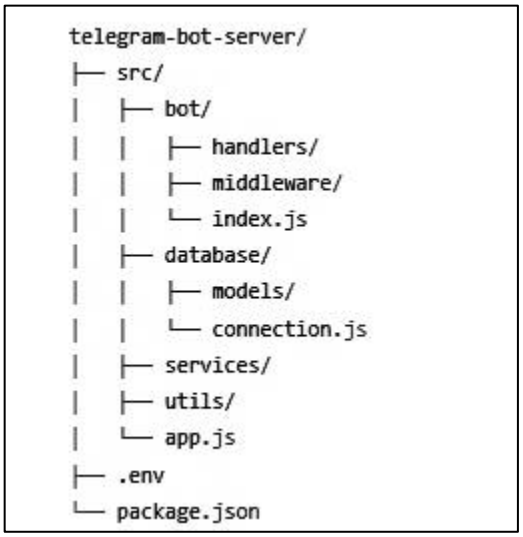


Рисунок 3.1 – Структура серверної частини

Каталог bot/ відповідає за взаємодію безпосередньо з Telegram API. У ньому містяться обробники команд користувача (handlers/), проміжне

програмне забезпечення для перевірки даних чи логування (middleware/), а також файл index.js, який ініціалізує запуск бота та підключення всіх модулів.

Папка database/ містить два основні елементи: models/, де зберігаються схеми для роботи з MongoDB, та connection.js, який відповідає за встановлення з'єднання з базою даних.

У директорії services/ розміщено основну бізнес-логіку застосунку. Тут зосереджені запити до зовнішніх API, обробка отриманих даних, логіка пошуку музики тощо.

Каталог utils/ містить допоміжні функції: хелпери, валідатори, логери або інші корисні модулі, які використовуються в різних частинах застосунку.

Файл app.js є головним вхідним файлом програми, у якому відбувається конфігурація сервера та запуск основного процесу.

На верхньому рівні проєкту також знаходиться .env – файл з конфігураційними змінними середовища (наприклад, токени, URL бази даних), та package.json, який містить інформацію про залежності, скрипти та основні метадані Node.js-проєкту.

На рисунку 3.2 зображено основний клас TelegramBotServer, що реалізує серверну логіку Telegram-бота за допомогою бібліотеки node-telegram-bot-api та фреймворку Express. Цей клас об'єднує функціональність HTTP-сервера та Telegram-бота з підтримкою webhook або polling-запитів, в залежності від середовища.

У конструкторі класу ініціалізується токен бота, створюється інстанс Telegram-бота та Express-додатку, а також викликаються методи налаштування middleware, webhook та обробників команд. Налаштування middleware включає логування HTTP-запитів та підтримку JSON-парсингу. Webhook налаштовується лише за наявності відповідної змінної середовища; у іншому випадку активується polling для зручності локальної розробки.

Метод setupHandlers() підключає обробники для команд /start, /help, звичайних повідомлень користувача, а також callback-кнопок. Обробник /start надсилає вітальне повідомлення з інтерактивною клавіатурою та викликає

метод saveUser() для збереження інформації про користувача. Команда /help генерує список доступних функцій, таких як інтерактивні меню, персоналізовані відповіді, обробка файлів, тощо.

```
const TelegramBot = require('node-telegram-bot-api');
const express = require('express');
require('dotenv').config();

class TelegramBotServer {
    constructor() {
        this.token = process.env.TELEGRAM_BOT_TOKEN;
        this.bot = new TelegramBot(this.token, { polling: false });
        this.app = express();
        this.port = process.env.PORT || 3000;

        this.setupMiddleware();
        this.setupWebhook();
        this.setupHandlers();
    }

    setupMiddleware() {
        this.app.use(express.json());

        // Middleware для логування
        this.app.use((req, res, next) => {
            console.log(`${new Date().toISOString()} - ${req.method} ${req.path}`);
            next();
        });
    }

    setupWebhook() {
        const webhookUrl = process.env.WEBHOOK_URL;
        const webhookPath = '/webhook';

        if (webhookUrl) {
            // Використання webhook для production
            this.bot.setWebHook(`${webhookUrl}${webhookPath}`);

            this.app.post(webhookPath, (req, res) => {
                this.bot.processUpdate(req.body);
```

Рисунок 3.2 – Основний клас логіки

Клас також обробляє звичайні текстові повідомлення, фото, документи та callback-запити через відповідні методи. Для callback-запитів передбачено обробку клавіш, що відкривають меню, профіль або довідку. Метод showProfile() демонструє базову інформацію про користувача, зокрема дату реєстрації та кількість повідомлень, а showMenu() створює головне меню у вигляді кнопок з різними опціями.

Додатково передбачено базову симуляцію збереження та отримання даних про користувача з бази даних, що в реальному випадку підключалося б

до MongoDB або іншого сховища. Метод start() запускає HTTP-сервер, виводячи у консоль повідомлення про успішний запуск.

Цей клас є основою для побудови Telegram-бота з багатим функціоналом, хорошою структурою та можливістю масштабування.

3.2 Налаштування бази даних MongoDB

У цьому підрозділі розглядається процес налаштування бази даних MongoDB, яка використовується для зберігання інформації про користувачів чат-бота, їх активність, а також інші допоміжні дані, необхідні для повноцінної роботи системи. MongoDB обрано завдяки її гнучкості, простоті масштабування та зручності зберігання даних у форматі, схожому на JSON. Це дозволяє зручно працювати з динамічними структурами, які добре підходять для сучасних веб- і чатбот-застосунків.

Першим кроком у процесі налаштування є підключення до бази даних. Це виконується на початку запуску сервера. Підключення зазвичай здійснюється через URI-адресу, яка вказується в окремому файлі конфігурації середовища. Це дозволяє легко змінювати параметри доступу до бази залежно від середовища (розробка, тестування або продакшн). Після успішного підключення сервер отримує доступ до всіх колекцій та документів, що зберігаються в базі.

Далі створюються спеціальні структури – моделі, які описують, які саме поля буде містити кожен документ у базі. Наприклад, для користувачів це можуть бути Telegram ID, ім’я, нікнейм, дата реєстрації, кількість надісланих повідомлень тощо. Кожна така модель дозволяє системі стандартизовано взаємодіяти з відповідною інформацією.

Збереження користувача до бази відбувається під час першого запуску бота користувачем. Якщо база ще не містить інформацію про даного користувача, система створює новий запис, в якому фіксується основна

інформація. Крім того, при кожній взаємодії з ботом можна оновлювати окремі поля, наприклад кількість повідомлень або останню активність.

На рисунку 3.3 зображено структуру моделі користувача User, яка реалізована за допомогою бібліотеки Mongoose для роботи з базою даних MongoDB у середовищі Node.js. Ця модель відповідає за збереження, оновлення та обробку інформації про кожного користувача, який взаємодіє з Telegram-ботом.

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
    telegramId: {
        type: Number,
        required: true,
        unique: true
    },
    firstName: String,
    lastName: String,
    username: String,
    languageCode: String,
    isBot: {
        type: Boolean,
        default: false
    },
    registrationDate: {
        type: Date,
        default: Date.now
    },
    lastActivity: {
        type: Date,
        default: Date.now
    },
    messageCount: {
        type: Number,
        default: 0
    },
    settings: {
        notifications: {
            type: Boolean,
            default: true
        },
        language: {
            type: String,
            default: 'uk'
        }
```

Рисунок 3.3 – Структура моделі користувача User

У центрі схеми представлено об’єкт userSchema, що визначає всі поля користувача. Основним ідентифікатором є telegramId, який є обов’язковим (required) та унікальним (unique), що дозволяє ідентифікувати кожного користувача без помилок. Додатково модель зберігає ім’я (firstName),

прізвище (lastName), нікнейм (username) та код мови інтерфейсу (languageCode), які Telegram надає у відповідь при першій взаємодії користувача з ботом.

Поле isBot вказує, чи є об’єкт ботом, а не живою людиною. Це може бути корисно для фільтрації ботів з бази. Також модель автоматично зберігає дату реєстрації (registrationDate) та дату останньої активності (lastActivity), що дозволяє відслідковувати поведінку користувачів у часі.

Окремо фіксується кількість повідомлень, які користувач надіслав боту (messageCount). Це поле автоматично оновлюється через метод updateActivity, який змінює дату останньої активності на поточну та збільшує лічильник повідомлень на одиницю. Завдяки цьому можна реалізувати елементи аналітики або гейміфікації.

У підоб’єкті settings визначено налаштування користувача: можливість увімкнення чи вимкнення сповіщень (notifications) та обрана мова інтерфейсу (language), яка за замовчуванням встановлена як українська.

Отже, дана модель є центральною частиною логіки зберігання користувачів у системі чат-бота, забезпечуючи зручне й ефективне керування їхніми даними, активністю та індивідуальними налаштуваннями.

3.3 Підключення зовнішніх музичних API

Для реалізації функціональності пошуку музики у Telegram-боті необхідно інтегрувати зовнішні API, які надають доступ до музичних баз даних, метаданих треків та аудіопотоків. Найбільш зручним і популярним рішенням для цього є використання Spotify Web API або Last.fm API. Обидва сервіси дозволяють отримувати інформацію про виконавців, альбоми, окремі пісні, жанри та рейтинги.

Одним із основних інструментів для взаємодії з музичними API в Python-проєкті є бібліотека Spotipy, яка є обгорткою для Spotify Web API. Для її

використання необхідно зареєструвати застосунок у Spotify Developer Dashboard, отримати client_id та client_secret, а також налаштувати редіректURL для авторизації, навіть якщо бот працює в обмеженому режимі без доступу до персональних даних користувача.

Після отримання токену доступу Telegram-бот може надсилати запити до API для пошуку треків за назвою, виконавцем чи жанром. Наприклад, користувач вводить назву пісні в боті, і система виконує пошук через зовнішнє API, повертаючи список результатів з назвами, обкладинками альбомів, попереднім переглядом аудіо (якщо доступно), а також посиланням на повну версію треку в Spotify.

На рисунку 3.4 зображено код серверної частини Telegram бота, інтегрованого з Spotify API для пошуку музичних треків. Код демонструє повний цикл роботи бота від ініціалізації до обробки користувацьких запитів.

```python
# Налаштування Spotify API
client_credentials_manager = SpotifyClientCredentials(
    client_id=os.getenv("SPOTIPY_CLIENT_ID"),
    client_secret=os.getenv("SPOTIPY_CLIENT_SECRET")
)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

# Команда /start
async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
    await update.message.reply_text("Привіт! Введи назву пісні, яку хочеш знайти 🎵")

# Обробка текстових повідомлень — пошук треку
async def search_song(update: Update, context: ContextTypes.DEFAULT_TYPE):
    query = update.message.text
    results = sp.search(q=query, type='track', limit=1)
    if results['tracks']['items']:
        track = results['tracks']['items'][0]
        track_name = track['name']
        artist_name = track['artists'][0]['name']
        url = track['external_urls']['spotify']
        await update.message.reply_text(f"🔊 {track_name} — {artist_name}\n🎧 {url}")
    else:
        await update.message.reply_text("Нічого не знайдено 😕")

# Запуск бота
async def main():
    app = ApplicationBuilder().token("ВАШ_TELEGRAM_BOT_TOKEN").build()
    app.add_handler(CommandHandler("start", start))
    app.add_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, search_song))
    await app.run_polling()
```

Рисунок 3.4 – Інтеграція Spotify API

У верхній частині коду виконується підключення необхідних бібліотек: spotipy для роботи з Spotify API, python-telegram-bot для створення Telegram

бота, та dotenv для безпечного зберігання конфіденційних даних. Далі відбувається налаштування аутентифікації Spotify через клієнтські credentials, які зчитуються з файлу змінних середовища.

Основна функціональність бота реалізована через дві асинхронні функції: start() для обробки команди /start, яка відправляє привітальне повідомлення користувачу, та search_song() для пошуку музичних треків за текстовим запитом. Функція пошуку використовує Spotify API для знаходження треків, витягує назву пісні, виконавця та посилання на Spotify, після чого повертає ці дані користувачу у форматованому вигляді.

У функції main() створюється екземпляр додатку Telegram бота з використанням токена, реєструються обробники команд та повідомлень, після чого запускається polling режим для отримання оновлень від Telegram серверів. Код завершується точкою входу, яка запускає асинхронний цикл подій для роботи бота.

Крім Spotify, альтернативним або додатковим джерелом інформації може бути Last.fm API, який також підтримує пошук треків, артистів та альбомів. Для використання цього API достатньо API-ключа, який можна отримати через реєстрацію розробника. Перевагою Last.fm є орієнтація на статистику прослуховувань та рекомендації.

Last.fm – популярна музична платформа, яка збирає статистику прослуховувань користувачів та надає рекомендації. Через їх API можна отримувати інформацію про треки, артистів, альбоми, а також статистику прослуховувань.

На рисунку 3.5 зображено фрагмент коду реалізації класу LastFmBot, який взаємодіє з API сервісу Last.fm для пошуку музичних треків, отримання інформації про виконавців та їхніх топ-треків.

Клас містить три основні методи. Метод search_track здійснює пошук треків за назвою та повертає список відповідних результатів.

Метод get_artist_info відповідає за отримання детальної інформації про артиста, зокрема його біографії та посилань на ресурси.

Метод get_top_tracks повертає список найпопулярніших треків обраного виконавця, що дозволяє швидко ознайомитися з його найвідомішими композиціями.

```python
# Завантаження змінних середовища
load_dotenv()

# Налаштування Last.fm API
LASTFM_API_KEY = os.getenv("LASTFM_API_KEY")
LASTFM_BASE_URL = "http://ws.audioscrobbler.com/2.0/"

# Налаштування логування
logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO)

class LastFmBot:
    def __init__(self):
        self.api_key = LASTFM_API_KEY

    def search_track(self, query, limit=5):
        """Пошук треків через Last.fm API"""
        params = {
            'method': 'track.search',
            'track': query,
            'api_key': self.api_key,
            'format': 'json',
            'limit': limit
        }

        try:
            response = requests.get(LASTFM_BASE_URL, params=params)
            data = response.json()

            if 'results' in data and 'trackmatches' in data['results']:
                return data['results']['trackmatches']['track']
```

Рисунок 3.5 – Інтеграція Last.fm API

Для кожного методу виконується HTTP-запит до Last.fm API з передачею необхідних параметрів, включно з API-ключем, що зберігається у змінній середовища. Усі відповіді обробляються у форматі JSON, а у випадку помилок ведеться логування.

Цей клас є основою для подальшої інтеграції музичного функціоналу у Telegram-бота, що дозволяє надавати користувачам актуальну інформацію про музику.

3.4 Тестування функціоналу та обробка типових сценаріїв

Тестування Telegram ботів є критично важливим етапом розробки, оскільки боти взаємодіють з реальними користувачами в реальному часі.

Правильно організоване тестування дозволяє виявити помилки на ранніх стадіях, забезпечити стабільну роботу в різних сценаріях та підготувати бота до несподіваних ситуацій.

Модульне тестування (Unit Testing) полягає у перевірці окремих компонентів або функцій Telegram-бота. Це дозволяє виявити помилки на ранньому етапі розробки, ізолюючи кожен модуль від інших залежностей.

Інтеграційне тестування спрямоване на перевірку взаємодії між різними модулями бота, а також з зовнішніми API (наприклад, Telegram API або сторонні сервіси). Таке тестування дозволяє виявити проблеми в обміні даними між частинами системи.

Функціональне тестування перевіряє основні сценарії використання бота з точки зору користувача. Метою є переконатися, що бот правильно реагує на команди, надає відповіді та виконує заплановану логіку.

Навантажувальне тестування проводиться для оцінки продуктивності бота під високим навантаженням. Це дозволяє визначити, як система працює при великій кількості одночасних запитів і чи витримує вона пікові навантаження без збоїв.

На рисунку 3.6 зображено структуру фікстур, які використовуються для організації асинхронного тестування Telegram-бота за допомогою бібліотеки pytest.

Зокрема, представлено фікстуру mock_update, що створює мок-об’єкт Update – базовий контейнер події у Telegram API, який містить інформацію про вхідне повідомлення. Цей об’єкт імітує основні властивості, необхідні для тестування: текст повідомлення (text), унікальний ідентифікатор чату (chat_id), дані про користувача, який надіслав повідомлення (from_user), а також асинхронний метод reply_text, що дозволяє імітувати відповідь бота без фактичного звернення до серверів Telegram.

Додатково на рисунку представлена фікстура mock_context, яка відповідає за створення мок-об’єкта контексту виконання команди або обробника події (ContextTypes.DEFAULT_TYPE). Цей контекст містить поля для передачі аргументів команди (args), а також словники для збереження

даних користувача (user_data) і даних, пов'язаних з конкретним чатом (chat_data). Наявність такої фікстури дозволяє ефективно тестувати логіку, що залежить від поточного стану сесії користувача або специфічних параметрів взаємодії.

```
    @pytest.mark.asyncio
    async def test_start_command_success(self, mock_update, mock_context):
        """Тест успішного виконання команди /start"""
        # Виконуємо команду
        await start_command(mock_update, mock_context)

        # Перевіряємо, що відповідь була надіслана
        mock_update.message.reply_text.assert_called_once()

        # Перевіряємо вміст відповіді
        call_args = mock_update.message.reply_text.call_args[0][0]
        assert "Привіт" in call_args
        assert "🎵" in call_args

    @pytest.mark.asyncio
    async def test_start_command_with_user_data(self, mock_update, mock_context):
        """Тест команди /start з даними користувача"""
        # Налаштовуємо дані користувача
        mock_update.message.from_user.first_name = "Олександр"
        mock_update.message.from_user.username = "alex_user"

        await start_command(mock_update, mock_context)

        # Перевіряємо персоналізацію
        call_args = mock_update.message.reply_text.call_args[0][0]
        assert "Олександр" in call_args or "alex_user" in call_args

    @pytest.mark.asyncio
    @patch('src.services.music_service.MusicService.search_track')
    async def test_search_song_found(self, mock_search, mock_update, mock_context):
        """Тест успішного пошуку пісні"""
        # Налаштовуємо мок для успішного пошуку
        mock_search.return_value = {
            'name': 'Test Song',
            'artist': 'Test Artist',
            'url': 'https://example.com/song',
```

Рисунок 3.6 – Структура асинхронного тестування

Також на рисунку 3.6 продемонстровано використання фікстури event_loop, яка створює новий цикл подій asyncio для асинхронного виконання корутин у тестах. Ця фікстура забезпечує правильне ініціалізування та завершення асинхронного середовища, що є критичним для коректного запуску та завершення тестів, які перевіряють асинхронний функціонал бота.

Загалом, наведені фікстури формують комплексне середовище для модульного тестування Telegram-бота. Вони дозволяють ізолювати внутрішню логіку застосунку від зовнішніх API і сервісів, забезпечують

контрольоване тестування типових сценаріїв користувацької взаємодії, а також допомагають відслідковувати і коректно обробляти можливі виняткові ситуації. Такий підхід значно підвищує якість програмного забезпечення, сприяє автоматизації тестування і знижує ризики появи помилок у робочому середовищі.

На рисунку 3.7 зображено структуру тестових сценаріїв для комплексної перевірки функціональності Telegram-бота MusicBot із використанням бібліотеки pytest. Тут представлено два основні класи тестів: TestUserScenarios та TestEdgeCases, які охоплюють типові випадки взаємодії користувача з ботом, а також нетипові та граничні ситуації.

```
        return MusicBot()

    @pytest.mark.asyncio
    async def test_new_user_onboarding(self, music_bot, mock_update, mock_context):
        """Сценарій: новий користувач відкриває бота"""
        # Симулюємо нового користувача
        mock_context.user_data = {}

        # Виконуємо команду /start
        await music_bot.start_command(mock_update, mock_context)

        # Перевіряємо, що користувач отримав привітання
        mock_update.message.reply_text.assert_called()

        # Перевіряємо збереження даних користувача
        assert 'first_interaction' in mock_context.user_data

    @pytest.mark.asyncio
    async def test_repeated_user_interaction(self, music_bot, mock_update, mock_context):
        """Сценарій: повторний користувач"""
        # Симулюємо існуючого користувача
        mock_context.user_data = {
            'first_interaction': '2023-01-01',
            'search_count': 5
        }

        await music_bot.start_command(mock_update, mock_context)

        # Перевіряємо персоналізоване привітання
        call_args = mock_update.message.reply_text.call_args[0][0]
        assert "знову" in call_args.lower() or "снова" in call_args.lower()

    @pytest.mark.asyncio
    @patch('src.services.music_service.MusicService.search_track')
```

Рисунок 3.7 – Структура тестових сценаріїв

Клас TestUserScenarios містить тести, що імітують поведінку реальних користувачів. Наприклад, тест test_new_user_onboarding перевіряє сценарій першої взаємодії нового користувача з ботом, контролюючи коректність вітального повідомлення та ініціалізацію даних сесії. Тест

test_repeated_user_interaction моделює повернення вже зареєстрованого користувача і перевіряє персоналізоване привітання. Сценарії test_successful_search_flow і test_multiple_searches_session перевіряють коректність пошуку музичних композицій та обробку кількох запитів у межах однієї сесії. Також увагу приділено тесту test_api_failure_recovery, який імітує помилки зовнішнього API та перевіряє здатність бота відновлювати нормальну роботу після збоїв.

Другий клас TestEdgeCases зосереджений на перевірці стійкості системи до нетипових і складних ситуацій. Тест test_special_characters_in_query демонструє, що бот коректно працює із запитами, які містять спеціальні символи, емодзі та тексти різними мовами. Сценарій test_concurrent_requests імітує одночасну обробку кількох паралельних запитів, що є важливим для підтримки високої продуктивності й відсутності блокувань. Тест test_rate_limiting перевіряє реалізацію механізмів обмеження частоти запитів, запобігаючи потенційним зловживанням.

Всі тести побудовані із застосуванням мок-об’єктів та патчінгу, що дозволяє ізолювати логіку бота від зовнішніх сервісів і API, тим самим гарантувати повторюваність і стабільність результатів. Для асинхронних функцій використовується декоратор @pytest.mark.asyncio, що забезпечує коректне виконання корутин у тестовому середовищі.

Таким чином, на рисунку 3.7 наочно представлено підхід до організації комплексного тестування, що охоплює як типові сценарії користувацької взаємодії, так і різноманітні граничні випадки, що суттєво підвищує надійність і якість розробленого програмного продукту.

На рисунку 3.8 зображено реалізований інтерфейс телеграм-бота SongSeeker, призначеного для пошуку та надання музичних треків користувачам. Центральна частина інтерфейсу містить повідомлення від бота у вигляді музичних файлів, кожне з яких супроводжується назвою треку, виконавцем, обсягом файлу та можливістю взаємодії (наприклад, надсилання реакції у вигляді сердечка).

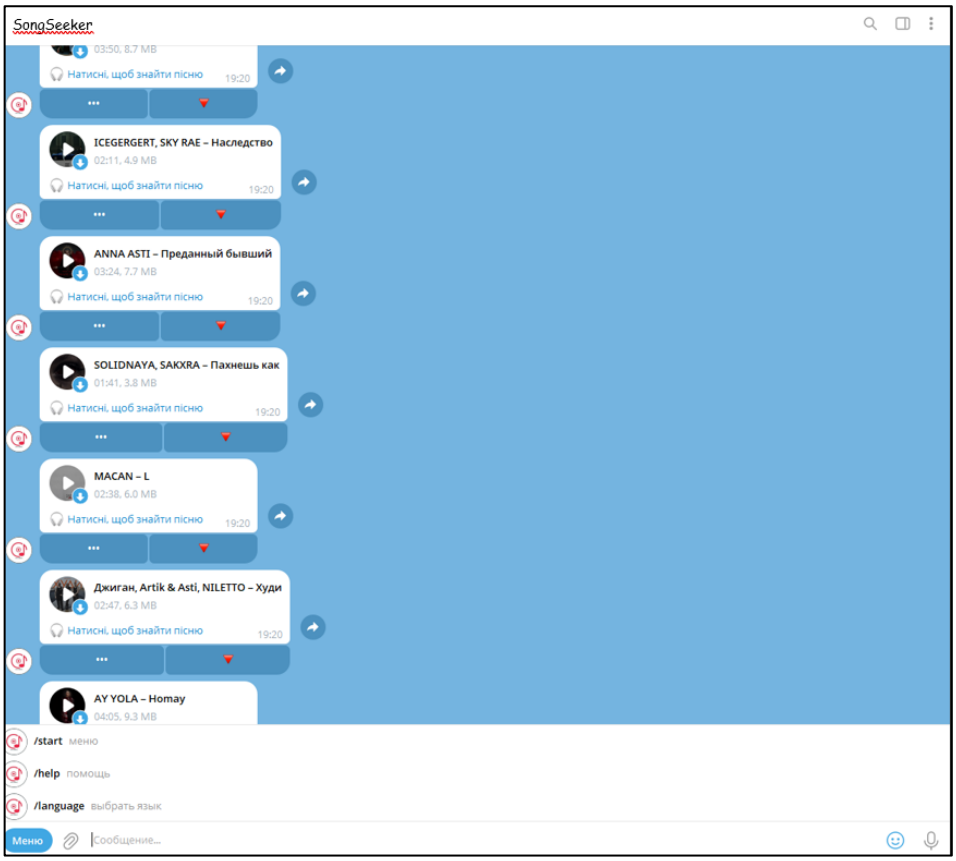


Рисунок 3.8 – Головне поле пошуку

Пошук музики у телеграм-боті SongSeeker реалізовано через інтерактивну взаємодію користувача з ботом у режимі реального часу. Користувач вводить запит – ключове слово або назву композиції/виконавця, на що бот автоматично здійснює пошук відповідних аудіотреків у власній базі даних. У якості сховища використовується MongoDB, що забезпечує гнучке та масштабоване зберігання інформації про музичні файли, включаючи метадані (назву треку, ім’я виконавця, розмір файлу, формат, дату завантаження тощо).

Після опрацювання запиту бот надсилає користувачу список знайдених композицій у вигляді повідомлень з аудіофайлами. Кожне повідомлення містить елемент керування (наприклад, кнопку “сердечко” або реакцію), що дозволяє залишити вподобання або зберегти трек.

У нижній частині інтерфейсу розташоване меню швидкого доступу до основних функцій. Команда /start використовується для запуску або перезапуску бота, дозволяючи користувачу розпочати взаємодію з ним з початку. Команда /help відображає довідкову інформацію щодо використання

бота, пояснюючи доступні функції та команди. Команда /language дає змогу змінити мову інтерфейсу, що забезпечує зручність для користувачів з різними мовними вподобаннями.

Таке компонування забезпечує інтуїтивну зручність навігації, а використання стриманого синього фону створює комфортне візуальне середовище для взаємодії. Елементи інтерфейсу розміщені логічно, що сприяє ефективному використанню функціоналу та мінімізує навантаження на користувача під час пошуку музики.

3.5 Висновки до розділу 3

У розділі 3 було розглянуто процес розробки та реалізації Telegram-бота для пошуку музики з використанням Last.fm API. Було описано архітектуру бота, основні функціональні компоненти, а також механізми обробки користувацьких запитів і взаємодії з зовнішніми API.

В ході роботи над розділом було розроблено базову структуру бота з основними командами, реалізовано функції пошуку треків та отримання інформації про виконавців. Особлива увага була приділена забезпеченню стабільної роботи через обробку потенційних помилок API та підтримку асинхронної взаємодії.

Також у розділі було описано підхід до тестування функціоналу, включаючи розробку тестових сценаріїв для перевірки типових та крайніх випадків використання. Це дозволило забезпечити високу якість програмного продукту, підвищити його надійність і комфорт користування.

Отже, проведені дослідження та практична реалізація підтверджують можливість ефективного використання Telegram-ботів для музичних сервісів, а також демонструють потенціал інтеграції з API сторонніх сервісів для розширення функціоналу та підвищення користувацького досвіду.

ВИСНОВКИ

У кваліфікаційній роботі було розглянуто основні етапи проєктування та реалізації телеграм-бота для пошуку музики, розробленого з використанням мови програмування Python для серверної логіки та бази даних MongoDB для зберігання інформації.

Під час виконання роботи було реалізовано низку важливих завдань, спрямованих на створення функціонального Telegram-бота для роботи з музичними сервісами. Спочатку було проаналізовано сучасні музичні сервіси та API, зокрема Last.fm і Spotify, а також вивчено доступні інструменти та технології для розробки чатботів. На основі цього було сформульовано як функціональні, так і нефункціональні вимоги до системи. Далі спроєктовано архітектуру бота, що охоплює обробку запитів користувачів, інтеграцію з музичними API та взаємодію з базою даних. Серверна частина була реалізована мовою Python із використанням асинхронних бібліотек, що забезпечує ефективну та швидку обробку запитів. Для збереження користувацьких даних і історії пошуків було налаштовано та інтегровано базу даних MongoDB. Реалізовано механізми пошуку музики, отримання інформації про треки та виконавців через зовнішні API. Також впроваджено базові сценарії взаємодії користувача з ботом у Telegram. На завершальному етапі проведено тестування основних функціональних можливостей бота, зокрема модульне та інтеграційне тестування.

Робота поділена на три основні розділи. У першому розділі виконано аналіз існуючих рішень у сфері музичних чатботів та обґрунтовано вибір технологічного стеку: Python як потужної мови програмування для розробки серверної логіки та MongoDB як гнучкої NoSQL бази даних. У другому розділі описано процес проєктування системи на концептуальному, логічному та фізичному рівнях. Третій розділ присвячений реалізації ключових компонентів бота, опису взаємодії з API музичних сервісів і методам тестування розробленого програмного забезпечення.

Отримані результати можуть бути використані як основа для створення масштабованих і зручних у використанні музичних чатботів у різних сферах, зокрема для персоналізованого музичного супроводу, рекомендаційних систем і сервісів розважального контенту. Запропоновані рішення відкривають перспективи для подальшої інтеграції з іншими платформами та застосуваннями штучного інтелекту для покращення користувацького досвіду.

ПЕРЕЛІК ПОСИЛАНЬ

1. How to Build a Telegram Bot: A Beginner’s Step-by-Step Guide. *Medium*. URL: https://tjtanjin.medium.com/how-to-build-a-telegram-bot-a-beginners-step-by-step-guide-c671ce027c55 (дата звернення: 01.03.2025).

2. Building Telegram Bot with Python-Telegram-Bot: A Comprehensive Guide. *Medium*. URL: https://medium.com/@moraneus/building-telegram-bot-with-python-telegram-bot-a-comprehensive-guide-7e33f014dc79 (дата звернення: 11.03.2025).

3. Bradshaw S., Brazil E., Chodorow K. MongoDB : The Definitive Guide : Powerful and Scalable Data Storage : 3rd ed. Sebastopol : O’Reilly Media, 2020. 511 p.

4. Karpov V, Netto D. Professional AngularJS. Birmingham : Wrox, 2015. 408 p.

5. Hellmann D. The Python 2 Standard Library by Example. URL: https://doughellmann.com/blog/the-python-standard-library-by-example/ (дата звернення: 26.03.2025).

6. Turnbull J. The Docker Book. James Turnbull Publishing, 2014. 436 p.

7. Merge cascade plugin. *Flowie: For Bitbucket Cloud*. URL: https://tinyurl.com/ytmwcnej (дата звернення: 18.04.2025).

8. Using Py Dotenv (python-dotenv) Package to Manage Env Variables. *Configuration-as-Code*. URL: https://configu.com/blog/using-py-dotenv-python-dotenv-package-to-manage-env-variables/ (дата звернення: 20.04.2025).

9. Structurizr DSL cookbook. *Structurizr*. URL: https://docs.structurizr.com/dsl/cookbook/ (дата звернення: 21.04.2025).

10. UML Modeling, Software Process and Tool. *Visual Paradigm*. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-modeling-software-process-and-tool/ (дата звернення: 25.04.2025).

11. Welcome to Lucidchart : Lucidchart Documentation. *Lucid*. URL: https://help.lucid.co/hc/en-us/articles/11970952773652-Welcome-to-Lucidchart (дата звернення: 27.04.2025).

12. Pillai A. B. Software Architecture with Python: Design and architect highly scalable, robust, clean, and high performance applications in Python. Birmingham : Packt Publishing, 2017. 556 p.

13. Raj S. Building Chatbots with Python: Using Natural Language Processing and Machine Learning. Birmingham : Apress, 2018. 211 p.