# ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

|  |  |
| --- | --- |
| Скорочення | Позначення |
| ПЗ | Програмне забезпечення |
| КП | Курсовий проєкт |
| БД | База даних |
| Бот | *Telegram*-чатбот |
| *VS* | *Visual Studio Code* |
| *Py* | *Python* |
| *ID* | Унікальний ідентифікатор |

# ЗМІСТ

[ПЕРЕЛІК ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ 3](#_Toc196771884)

[ВСТУП 5](#_Toc196771886)

[1 ПОСТАНОВКА ЗАДАЧІ 8](#_Toc196771887)

[2 ОПИС МАТЕМАТИЧНОГО МЕТОДУ РІШЕННЯ ЗАДАЧІ 14](#_Toc196771888)

[2.1 Визначення структури вхідних даних 14](#_Toc196771889)

[2.2 Структура збереження даних у JSON 15](#_Toc196771890)

[2.3 Визначення структури вихідних даних 15](#_Toc196771891)

[2.4 Математичний опис задачі 16](#_Toc196771892)

[3. ОПИС АЛГОРИТМУ РІШЕННЯ ЗАДАЧІ 17](#_Toc196771893)

[3.1 Опис стандартних функцій та процедур 17](#_Toc196771894)

[4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 19](#_Toc196771895)

[4.1 Вибір програмного засобу 19](#_Toc196771896)

[4.3 Програмування та налагодження програми 20](#_Toc196771897)

[4.4 Процес документування коду 27](#_Toc196771898)

[5 ІНСТРУКЦІЯ З ВИКОРИСТАННЯ ПЗ 29](#_Toc196771899)

# ВСТУП

*Telegram* [11] – популярний месенджер, який дозволяє користувачам обмінюватися повідомленнями, відео, музикою та файлами. Блогери використовують сервіс для публікації свого контенту та ведення бізнесу – продажу товарів і послуг. Окрім цього, у *Telegram* є можливість створювати власних ботів – програми всередині месенджера, що автоматизують спілкування.

*Telegram* використовує власний протокол шифрування *MTProto*. *MTProto API (Telegram API)* – *API*, через який ваш додаток *Telegram* взаємодіє із сервером. *Telegram API* повністю відкритий, тож будь-який розробник може створити власний клієнт месенджера.

*Telegram*-чатбот (Бот) [7] – міні програма всередині месенджера, які керуються текстовими командами в чаті за принципом «запит – відповідь».

Для створення ботів було розроблено *Telegram Bot API* – надбудову над *Telegram* *API*. Як зазначено на офіційному сайті [3]: "Щоб використовувати *Bot API*, вам не потрібно нічого знати про те, як працює протокол шифрування *MTProto* – наш допоміжний сервер сам оброблятиме все шифрування та зв’язок із *Telegram API*. Ви підключаєтеся до сервера через простий *HTTPS*-інтерфейс, який надає спрощену версію *Telegram API*”, це свідчить що особливих технічних знань зі створенням безпечного з’єднання по безпечним протоколам мати немає необхідності і користуватись *Telegram Bot API* можна навіть недосвідченим програмістам.

Бот може виконувати наступні завдання:

* керувати спільнотами;
* бронювати місця в кафе;
* завантажувати файли;
* обробляти платежі;
* фільтрувати повідомлення в чатах;
* допомагати з реєстрацією користувачів;
* відповідати на запитання (*support*-бот);
* продавати послуги;
* виконувати роль розважального бота.

Кожен бот має унікальну назву (*username*) тому легко знаходити потрібного бота через пошук у *Telegram*.

Можливості ботів обмежені лише фантазією та навичками розробників, тому компанії активно використовують їх для щоденних завдань.

У квітні 2022 року команда *Telegram* представила друге покоління ботів: розробники отримали інструменти для створення повноцінних інтерфейсів на *JavaScript* прямо в месенджері. Такі боти відходять від стандартної моделі «запит – відповідь у чаті» [3].

Курсовий проєкт (КП) передбачає створення бота, який дозволяє зберігати, організовувати та прослуховувати музичні файли. Особливістю такого програмного забезпечення є простота використання, швидкий доступ до музичних файлів та можливість створення персоналізованих плейлистів.

В ході досліджень, щодо реалізації поставленого завдання, виявлено функціональні боти стилістику та графічний інтерфейс яких надав змогу зрозуміти можливості *Telegram* в рамках дисципліни:

* *@eyesaverbot* – Бот для завантаження та прослуховування музики за допомогою знаходження її за назвою;
* *@SaveOFFbot / @SaveYoutubeBot* – Боти для завантаження та зберігання відео/фото/аудіо файлів з різних джерел;
* *@iris\_black\_bot* – Дуже функціональний та популярний бот в *Telegram*, допомагає керувати чатом/сітками чатів, виконує розважальну функцію та допомагає ідентифікувати користувача;
* *@telegraph* – Офіційний бот від Telegram який допомагає робити статті/книги і тому подібне.
* *@BotFather* – Головний бот Telegram який надає змогу почати створювати свого бота.

Розробники *Telegram* активно розвивають месенджер, тому важливо, щоб бібліотека оновлювалася так само часто й підтримувала всі нововведення сервісу. Найчастіше ботів пишуть на мові *Python* [1] (Py) через простоту та зручність цієї мови.

Зручність та популярність в розробці ботів, наявності бібліотек для роботи з *Telegram API* та можливості ефективно працювати з аудіофайлами стали критеріями вибору мови *Py*, як інструменту.

Однією з переваг *Telegram* є відкрите *API* з гарною документацією, тому ботів можна розробляти майже будь-якою мовою програмування, але зручніше використовувати вже готові бібліотеки.

В якості бібліотеку мови *Py* для комунікації з *Telegram* обрано *API telebot*.

*Telebot (pyTelegramBotAPI)* [4] – бібліотека для створення простих ботів із невеликою кількістю користувачів, яка підтримує основні методи *Telegram API*.

До недоліків *telebot* слід віднести складну організацію асинхронних запитів і масштабованості навантаження, але з переваг – низький поріг входу для новачків.

*Visual Studio Code (VS)* – безкоштовне легке та потужне середовище розробки, створене компанією *Microsoft.* *VS* підтримує велику кількість мов програмування, зокрема *Py*, і забезпечує зручну роботу завдяки вбудованому терміналу, розширенням, автодоповненням коду та засобам налагодження.

*VS* використовується як середовище розробки через свою зручність, підтримку *Py* і наявність плагінів для ефективної роботи з кодом і тестуванням.

Для демонстрації програмного коду та оновлень використана технологія *GitHub.*

*GitHub* – веб-платформа для зберігання, керування та спільної розробки проєктів на основі системи контролю версій *Git*. Вона дозволяє розробникам ефективно відслідковувати зміни в коді, працювати над проєктами у командах, створювати гілки для тестування нових функцій, а також об'єднувати результати роботи через *pull request. GitHub* забезпечує інструменти для контролю версій, документації, автоматизації процесів *CI/CD* та обміну кодом у відкритому або приватному режимі. Завдяки інтеграції з іншими сервісами і багатій екосистемі, *GitHub* став стандартом де-факто для ведення сучасних проєктів у програмуванні.

# 1 ПОСТАНОВКА ЗАДАЧІ

У рамках КП з дисципліни «Інструментальні засоби візуального програмування» визначення завдання розробити бота для прослуховування музики з використанням мови програмування *Py* та бібліотеки *Telebot у VS.*

Бот має забезпечити інтуїтивно зрозумілий інтерфейс взаємодії з користувачем через *Telegram*, а також надійно працювати з БД для зберігання інформації про музичні файли та плейлисти. Особливу увагу приділяється зручності використання, швидкому доступу до потрібної композиції та створенню плейлистів. На рисунку 1 за допомогою нотацій *UML – USE CASE* діаграма, відображені функціональні вимоги до програмного забезпечення (ПЗ):

* прослухати музику;
* завантажити музику;
* створити плейлист.

|  |
| --- |
|  |
| Рисунок 1 – Функціональні вимоги до ПЗ |

Метою КП є створення ПЗ – бота *“NLAI”* який вирішує наступні завдання:

1. Додавання аудіофайлів до бази даних (БД) із збереженням їхніх унікальних ідентифікаторів (*ID*). На рисунку 2 зображена діаграма послідовності функціональної вимоги “Завантажити музику”.

|  |
| --- |
| Изображение выглядит как снимок экрана, текст, Шрифт, круг  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 2 – діаграма послідовності функції “Завантажити музику” |

В таблиці 1 представлена специфікація прецеденту “Завантажити музику”.

|  |  |
| --- | --- |
| Таблиця 1 – специфікація функціональної вимоги “Завантажити музику” | |
| Завантажити музику | |
| Опис | Виклик цієї функції дає запит на завантаження музики до бази даних. |
| Актори | Користувач |
| Передумова | Користувач надіслав файл формату *mp3* чи *ogg* в чат бота. |
| Тригер | Користувач запустив бота |
| Успішний сценарій | 1. Користувач відкриває чат бота та прописує /*start*, щоб бот мав інформацію о користувачеві. 2. Натискає кнопку Почати. 3. Натискає кнопку Додати Музику. 4. Додає файл формату mp3 до чату. 5. Бот оброблює файл і при не наявності в бібліотеці додає музику для прослуховування. 6. Натискає кнопку вийти щоб вийти з стану додавання музики. |
| Альтернативний сценарій | 1. Якщо формат файлу не вірний то нічого не зберігається. 2. Якщо музика вже збережена і користувач надсилає ідентичний файл, то нічого не зберігається. |

1. Відтворювання аудіофайлів на запит користувача. На рисунку 3 зображена діаграма послідовності функціональної вимоги “Прослухати музику”.

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, Шрифт, дизайн  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 3 – діаграма послідовності функції “Прослухати музику” |

В таблиці 2 представлена специфікація прецеденту “Прослухати музику”.

|  |  |
| --- | --- |
| Таблиця 2 – специфікація функціональної вимоги “Прослухати музику” | |
| Прослухати музику | |
| Опис | Виклик цієї функції дає запит на надсилання музики |
| Актори | Користувач |
| Передумова | Музика, була додана до бази даних. |
| Тригер | Натискання кнопки прослухати музику. |
| Успішний сценарій | 1. Користувач відкриває чат бота та прописує /*start* , щоб бот мав інформацію о користувачеві. 2. Натискає кнопку Почати 3. Натискає кнопку Прослухати музику 4. Проглядає список доданої музики та вибирає яку хоче прослухати 5. Натискає кнопку прослухати |
| Альтернативний сценарій | Якщо музики немає то переглянути доступну для прослуховування музику неможливо. |

1. Організовування музики у вигляді плейлистів. На рисунку 4 зображена діаграма послідовності функціональної вимоги “Створити плейлист”.

|  |
| --- |
| C:\Users\stma0917\Downloads\Telegram Desktop\Playlist.png |
| Рисунок 4 – діаграма послідовності функції “Прослухати музику” |

В таблиці 3 представлена специфікація прецеденту “Створити плейлист”.

|  |  |  |
| --- | --- | --- |
| Таблиця 3 – специфікація функціональної вимоги “Створити плейлист” | | |
| Створити плейлист | |
| Опис | Виклик цієї функції дає змогу створити та прослухати створений плейлист |
| Актори | Користувач |
| Передумова | Музика, була додана до бази даних. |
| Тригер | Натискання кнопки плейлист. |
| Успішний сценарій | 1. Користувач відкриває чат бота та прописує /*start* , щоб бот мав інформацію о користувачеві. 2. Натискає кнопку Почати. 3. Натискає кнопку Плейлист. 4. Натискає кнопку створити Плейлист. 5. Додає назву плейлисту. 6. Натискає кнопку редагувати плейлисти. 7. Вибирає створений плейлист. 8. Додає музику яку хоче в плейлист. 9. Натискає кнопку прослухати плейлисти. 10. Вибирає плейлист. 11. Бот відправляє файли. |
| Альтернативний сценарій | 1. Якщо немає музики то користувач не зможе додати музику до плейлиста. 2. Якщо не було створено плейлиста то додати музику до неіснуючого плейлиста не вийде. |

# 2 ОПИС МАТЕМАТИЧНОГО МЕТОДУ РІШЕННЯ ЗАДАЧІ

## 2.1 Визначення структури вхідних даних

У процесі взаємодії з ботом користувач надсилає певні вхідні дані, які обробляються програмою. До таких вхідних даних належать:

1. Команди користувача - текстові повідомлення, що починаються з символу «/», наприклад:

* /*start* - запуск бота;
* /*info* - отримання довідкової інформації;
* /*help* – отримання свого *ID* користувача.

1. Аудіофайли - медіафайли, надіслані користувачем. *Telegram* передає їх у форматі *JSON*-об’єкта, який містить:
   * *file\_id* - унікальний ідентифікатор файлу в *Telegram*;
   * *file\_name* - назва файлу (може бути відсутньою);
   * *user\_id* - ідентифікатор користувача який додав файл;
2. Інтерактивні елементи – натискання на кнопки, створені за допомогою *InlineKeyboardMarkup* або *ReplyKeyboardMarkup*, що дозволяють користувачу, наприклад, вибрати трек або підтвердити додавання до плейлиста.
3. Додаткова інформація:
   * *user\_id* - ідентифікатор користувача;
   * *username* - *Telegram*-нік користувача (якщо є);
   * *message\_id* - ідентифікатор повідомлення;
   * *chat\_id* - ідентифікатор чату, який використовується для відповіді.

*JSON (JavaScript Object Notation)* [5,8] – текстовий формат обміну даними, який є простим для читання людиною та обробки комп’ютером. Він використовується для зберігання структурованої інформації у вигляді пар "ключ-значення".

У даному проєкті *JSON* використовується для збереження таких параметрів:

* + Назва треку;
  + Ідентифікатор файлу;
  + Додаткова інформація (за потреби).

## 2.2 Структура збереження даних у *JSON*

Надалі представлено фрагмент *JSON* файлу для збереження даних:

*{*

*"Касета - SadSvit.mp3": [*

*{*

*"file\_id": "CQACAgIAAxkBAAIJEGgGQG",*

*"user\_id": 853521065*

*}*

*],*

*}*

Таким чином, основними вхідними даними є текстові команди, медіафайли та дії користувача у вигляді натискань кнопок. Усі вони передаються у вигляді структурованих об'єктів *JSON* через Telegram *API* та обробляються за допомогою бібліотеки *Telebot.*

Вхідними даними для роботи бота є інформація про музичні треки, яка зберігається у форматі *JSON*.

## 2.3 Визначення структури вихідних даних

Вихідними даними бота є відповіді, які він надсилає користувачу через *Telegram API*, а також службові дії, пов’язані з обробкою та зберіганням даних у базі. Вихідні дані поділяються на кілька основних категорій:

1. Текстові повідомлення - відповіді, які бот надсилає у відповідь на команди чи дії користувача:
   * Привітання після /*start*;
   * Підтвердження успішного додавання аудіофайлу;
   * Повідомлення про помилки (наприклад, "Ви не в режимі додавання музики натисніть…");
   * Інформація про треки або плейлисти.
2. Мультимедійні повідомлення:
   * Відправка аудіофайлів користувачу за запитом (*send\_audio*);
   * Надсилання файлів із бази даних за їхнім *ID*.
3. Інтерактивні елементи:
   * Кнопки вибору треків або плейлистів (*InlineKeyboardButton,* *ReplyKeyboardMarkup*);
   * Повідомлення зі вбудованими кнопками для подальших дій (наприклад, «Додати до плейлиста», «Видалити», «Наступний трек»).
4. Операції з базою даних (внутрішні вихідні дані):
   * Збереження метаданих аудіофайлу (*ID*, ім’я, користувач)
   * Оновлення інформації про плейлисти;
   * Зчитування списку треків або параметрів користувача для формування відповідей.
5. Повідомлення у вигляді списків або таблиць:
   * Перелік доступних треків;
   * Вміст конкретного плейлиста.

Вихідні дані можуть мати різну форму - від простих текстових повідомлень до складних структур із кнопками та аудіо, що забезпечує зручність взаємодії з користувачем.

## 2.4 Математичний опис задачі

Задачу розробки бота для прослуховування музики можна формалізувати у вигляді математичної моделі, яка описує вхідні дані, внутрішні обчислення та вихідні результати.

Множини вхідних об'єктів:

* + *U={u1,u2,...,un}* – множина користувачів.
  + *C = {c1,c2,...,cm}* – множина команд (наприклад, */start, /info, /help*).
  + *A = {a1,a2,...,ak}* – множина аудіофайлів, де кожен елемент *ai* описується кортежем: *ai* = *(id ,name ,user)*
  + *P = {p1,p2,...,pl}* – множина плейлистів, де кожен плейлист є підмножиною множини аудіофайлів: *pi ⊆ A*

Функції обробки:

* + *f : C × U → R* - функція, яка відображає команду користувача на відповідь бота.
  + *g : A × U → DB* - функція додавання аудіофайлу до бази даних.
  + *h : P × U → MSG* - функція генерації повідомлення з плейлистом для конкретного користувача.

Відношення:

* + *rel1 ⊆ U × A* - відношення між користувачами і треками (хто що завантажив).
  + *rel2 ⊆ P × A* - відношення між плейлистами і треками (які треки входять у який плейлист).

Мета. Побудувати систему, яка реалізує коректне відображення команд і дій користувача на відповідні результати:

*∀(c,u) ∈ C × U, f(c,u)* = очікувана відповідь

*∀ai ∈ A, g(ai,u) ⇒ ai ∈ DB*

Таким чином, задача моделюється як побудова інформаційної системи, яка реалізує відображення вхідних запитів на відповідні дії, дотримуючись заданої структури даних та логіки взаємодії.

# 3. ОПИС АЛГОРИТМУ РІШЕННЯ ЗАДАЧІ

Розробка бота для прослуховування музики передбачає реалізацію послідовного алгоритму обробки запитів користувача, взаємодії з *Telegram API* та базою даних. Основний алгоритм роботи програми має вигляд:

1. Ініціалізація бота з використанням токена *Telegram*.
2. Очікування команд або дій від користувача.
3. Обробка отриманої команди.
4. Формування відповіді або результату дії.
5. Надсилання відповіді користувачу у вигляді повідомлення або медіа.
6. Збереження або оновлення інформації у базі даних.

Усі команди обробляються за допомогою функцій бібліотеки *Telebot*, яка забезпечує зручну роботу з *Telegram API*.

## 3.1 Опис стандартних функцій та процедур

У рамках реалізації бота використовуються наступні ключові функції та обробники:

1. *@bot.message\_handler(commands=['start'])``;*
   * Призначення: надсилає привітальне повідомлення користувачу;
   * Вивід: текстове повідомлення з короткою інструкцією та ін-лайн кнопкою «почати».
2. *@bot.message\_handler(commands=['help']);*
   * Призначення: надає користовачу його *ID* в *Telegram*;
   * Вивід: текстове повідомлення з текстом «Ваш *ID*: (*ID*) ».
3. *@bot.message\_handler(content\_types=['info']);*
   * Призначення: надає інформацію про розробника;
   * Вивід: текстове повідомлення з інформацією про розробника.
4. *@bot.callback\_query\_handler(func=lambda call: call.data == 'music');*
   * Призначення: надає основне меню користування ботом;
   * Вивід: текстове повідомлення з текстом інформацією про кнопки.
5. *@bot.callback\_query\_handler(func=lambda call: call.data == 'add\_music');*

* Призначення: надає користувачу змогу додати свою музику;
  + Вивід: текстове повідомлення з текстом" Надішліть файл з музикою в цей чат, щоб додати його".

1. *@bot.message\_handler(content\_types=['audio']) ;*

* Призначення: обробник аудіо файлів для зберігання. Якщо користувач не в режимі додавання музики то *user\_data = {'adding\_music': False}* щоб бот не оброблював аудіо увесь час;
* Вивід: якщо успішний сценарій то "Файл успішно додано до вашої бібліотеки!".

1. *@bot.callback\_query\_handler(func=lambda call: call.data == 'music\_option');*

* Призначення: надає користувачу змогу прослухати додані треки;
* Вивід: Повідомлення з першою доданою музикою та кнопками: назад та пагінації (сторінками) тобто наступна та попередня музика.

1. *@bot.callback\_query\_handler(func=lambda call: call.data == 'playlist');*

* Призначення: надає користувачу меню користування плейлистами;
* Вивід: Повідомлення з інформацією про кнопки.

1. *@bot.callback\_query\_handler(func=lambda call: call.data == 'create\_playlist');*

* Призначення: надає користувачу змогу створити плейлист;
* Вивід: Повідомлення" Плейлист '*{playlist\_name}*' успішно створено!".

1. *def save\_music(music\_data):* - зберігає *ID* файлу в *json*;
2. *def load\_music():* - завантажує музику з файлу *json*;
3. *def load\_playlists():* - Завантажує плейлисти з файлу у глобальну змінну *“playlists”;*
4. *def save\_playlists():* - Зберігає плейлисти у файл *json*;
5. *def get\_user\_playlists(user\_id):* - Повертає плейлисти користувача за його *ID*;
6. *def update\_user\_playlists(user\_id, updated\_playlists):* - оновлює інформацію в плейлистах.

# 4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 4.1 Вибір програмного засобу

Для розробки бота обрано наступне ПЗ та інструменти:

* 1. Мова програмування *Python. Py* обраний завдяки своїй зручності та популярності в розробці ботів, наявності бібліотек для роботи з *Telegram API* та можливості ефективно працювати з аудіофайлами.
  2. Бібліотека *Telebot. Telebot* – потужна бібліотека для створення ботів на Py, яка забезпечує простий інтерфейс для взаємодії з *Telegram API*. Вона підтримує обробку повідомлень, медіафайлів, клавіатур, кнопок і дозволяє легко реалізувати необхідний функціонал.
  3. Середовище розробки *Visual Studio Code (VS).* *VS* використовується як середовище розробки через свою зручність, підтримку *Py* і наявність плагінів для ефективної роботи з кодом і тестуванням.
  4. *JSON* файли для зберігання даних. У цьому проєкті замість бази даних використовується *JSON* для зберігання інформації про аудіофайли та їх *ID*. Такий підхід обраний через простоту реалізації та достатність для невеликого проєкту. Кожен аудіофайл зберігається у форматі *JSON* з відповідними метаданими: *ID*, назва, тривалість тощо.
  5. Інші інструменти. Бібліотека *hashlib* [6] використовується для генерації унікальних хешів *callback data*, щоб уникнути проблем з обробкою запитів.
  6. Бібліотека *logging* використовується для тестування та налагодження використовуються стандартні модулі *Py, logging* для журналювання виконання програми.

**4.2 Визначення вимог до технічних засобів**

Технічні характеристики комп’ютера, на якому розроблявся курсовий проєкт:

* Тип комп’ютера: *Ноутбук Thin A15*
* Процесор: *AMD Ryzen 5 5500U (6 ядер, до 4.0 ГГц)*
* Обсяг оперативної пам’яті: *16 ГБ DDR4*
* Накопичувач: *SSD 512 ГБ*
* Тип монітора: *IPS-матриця, 15.6 дюйма, Full HD (1920×1080)*
* Тип принтера: *Відсутній (використовувався віртуальний друк у PDF)*
* Операційна система: *Windows 11 Home*
* Мова програмування: *Python 3.11*
* Середовище розробки: *Visual Studio Code 1.89.0*
* Додаткове програмне забезпечення: *розширення Python для VS Code,* бібліотека *pyTelegramBotAPI,* бібліотека *hashlib,* розширення *JSON Tools.*

Мінімальні вимоги для запуску програми:

* Тип комп’ютера: ПК або ноутбук середнього рівня
* Процесор: *не гірше Intel Core i3 або AMD Ryzen 3*
* Оперативна пам’ять: *мінімум 4 ГБ*
* Вільний простір на диску: *мінімум 100 МБ*
* Операційна система: *Windows 10, Windows 11, або Linux Ubuntu 20.04+*
* Обов'язкове програмне забезпечення:
  + Інтерпретатор *Python 3.10* або вище
  + Модулі *Python: pyTelegramBotAPI, hashlib*
  + Наявність стабільного доступу до Інтернету для обміну даними з *Telegram API.*

## 4.3 Програмування та налагодження програми

Основні етапи програмування включали:

1. Створення бота через сервіс *BotFather* та отримання токена доступу (рисунок 5) [9,10,12,14-17].

Щоб почати створювати бота, потрібно створити бота та отримати унікальний токен у офіційному боті від *Telegram @BotFather.*

Для цього потрібно ввести команду */newbot* Потім назву бота і *@username* бота Все бот створено. *@BotFather* Видасть унікальний токен.

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, программное обеспечение, Веб-сайт  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 4 – Фрагмент отримання токена доступу |

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 5 – Додаткові налаштування бота |

У цьому ж боті можна налаштувати візуальне відображення в *Telegram* (опис, аватарка, швидкий список команд при введенні / у полі введення тексту) та додаткові параметри, які можуть бути корисними в різних ситуаціях.(рисунки 5-6).

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, Шрифт  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 6 – Приклад підказок для бота в чаті |

Розробка програми здійснювалась за допомогою мови програмування *Py* у середовищі розробки *VS* тому переходимо до середовища розробки.

|  |
| --- |
| Изображение выглядит как текст, Шрифт, снимок экрана, дизайн  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 7 – Імпортовані бібліотеки |

1. Імпортування необхідних бібліотек (*telebot, hashlib, json*)( Рисунок 7).
2. Реалізація логіки бота для обробки команд користувача (старт, вибір музики, прослуховування).

Оскільки мені зараз не потрібно використати *webhook* для бота щоб не тратити ресурси, буде використовуватись *polling* в кінці коду *bot.polling(none\_stop=True)* щоб був безкінечний цикл(рисунок 8).

|  |
| --- |
|  |
| Рисунок 8 – Строка у коді |

1. Звернення до токену. Оскільки я буду користуватися бібліотекою *telebot* це виконується через [Назва змінної] = [звернення до бібліотеки].[звернення до класу] (‘Унікальний Токен’):

*bot = telebot.TeleBot (‘токен’)*

1. Обробка команди */start* та інших команд через / *@bot.massage\_handlers(commands=[‘start’]) (рисунок 9-10):*

*def main(massage): ….*

|  |
| --- |
| Изображение выглядит как текст, снимок экрана  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 9 – Строка у коді |
|  |
| Рисунок 10 – Кінцевий вигляд |

1. Інлайн-кнопки, треба імпортувати з бібліотеки:

telebot types from telebot import types

*markup=types.InlineKeybordMarkup()*

*markup.add(types.InlineKeyboardButton(команда)*.

Для створення кнопки:

*[Назва кнопки]=types.InlineKeyboardButton(команда)*

Для створення рядів кнопок:

*markup.row (назви кнопок).*

Щоб обробляти натискання кнопки треба там де зазначено “команда” ввести ключе слово.

Кінцевий вигляд(рисунок 11):

*‘Назва кнопки’ callback\_data=’ключове слово’* *test=types.InlineKeyboardButton(‘test’ callback\_data=’test’).*

Щоб їх оброблювати треба створити обробник (рисунок 12,13):

*@bot.callback\_query\_handler(func=lambda callback: True)*

*def callback\_test(callback):*

*if callback.data == ‘test’:*

*bot. …*

|  |
| --- |
|  |
| Рисунок 11 – Ін-лайн кнопка у коді |
|  |
| Рисунок 12 – Обробник у коді |

|  |
| --- |
|  |
| Рисунок 13 – Кінцевий вигляд |

1. Збереження даних про музику у форматі *JSON* для подальшого доступу (рисунок 14, 15).

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, Шрифт  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 14 – Функція збереження у *JSON* |
| Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 15 – Структура даних у *JSON* файлі |

Налагодження програми включало:

* Тестування кожної функції окремо для перевірки коректності роботи команд;
* Виправлення помилок при обробці повідомлень від користувача;
* Перевірка коректності збереження та зчитування даних з *JSON*-файлів;
* Тестування роботи бота в реальному середовищі Telegram із використанням особистого облікового запису.

В результаті налагодження забезпечено стабільну роботу програми, правильну обробку введення користувача та виконання відповідних дій.

## 4.4 Процес документування коду

Для спрощення розуміння структури коду та підвищення зручності його підтримки було використано автоматичну генерацію документації за допомогою інструменту *pdoc.* Документація відображає опис функцій, класів та модулів проєкту [13].

Для генерації документацій використовувалась наступна команда:

*python -m pdoc botnlai.py -o docs*(рисунок 16)

|  |
| --- |
|  |
| Рисунок 16 – Генерація документації |

Під час розробки програмного продукту дотримано стандарту документування коду *PEP 257*. Кожен модуль, клас і функція супроводжуються докстрінгами, які описують їх призначення, параметри та результати роботи.

Докстрінги оформлено відповідно до основних вимог *PEP 257*[2]:

* Однорядковий докстрінг: Для коротких функцій або методів використовується однорядковий докстрінг. Він повинен починатися з великої літери та закінчуватися крапкою.

Наприклад:

"""Ця функція обчислює середнє значення.""""

* Багаторядковий докстрінг: Для більш комплексних модулів, класів або функцій використовується багаторядковий докстрінг. Він починається з короткого загального опису, за яким слідує детальніше пояснення. Перший рядок відокремлюється порожнім рядком від решти тексту.

Наприклад:

"""Цей клас представляє собою модель користувача.

Клас містить методи для створення, оновлення та видалення користувачів з бази даних."""

* Використання імперативу: Докстрінги функцій і методів повинні описувати дії у формі імперативу. Це означає, що потрібно використовувати активний стан і наказовий спосіб. Наприклад, "Обчислює суму двох чисел" замість "Ця функція обчислює суму двох чисел".
* Докстрінги для кожного публічного модуля, класу, функції та методу: Кожен публічний елемент коду повинен мати докстрінг, що пояснює його призначення та використання. Це допомагає розробникам швидко зрозуміти, для чого призначений той чи інший елемент, без потреби глибокого занурення в код.
* Форматування: Дотримання чіткого і послідовного форматування, включаючи відступи та пусті рядки, щоб покращити читабельність докстрінгів. Докстрінги мають бути легко читабельними та естетично приємними, сприяючи швидкому розумінню коду.

# 5 ІНСТРУКЦІЯ З ВИКОРИСТАННЯ ПЗ

Інтерфейс користувача реалізовано у вигляді взаємодії через чат-бот у додатку *Telegram*. Бот обробляє текстові команди користувача та надсилає відповідні повідомлення.

Порядок роботи з програмою:

1. Запуск бота(рисунок 17). Користувач відкриває чат із ботом у *Telegram* і натискає кнопку «*Start*» або вводить команду */start.*

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 17 – Вигляд команди старт |

1. Головне меню (рисунок 18). Натиснувши кнопку «Почати» бот надсилає повідомлення з кнопками «Додати музику», «Плейлисти», «Прослухати музику».

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 18 – Вигляд головного меню |

1. Додати музику (рисунок 19). Користувач надає музичний файл до чату з ботом, якщо файл відповідає всім стандартам, то бот зберігає інформацію про цей файл.

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 19 – Додавання музики до бота |

1. Прослуховування музики (рисунок 20). Бот надсилає файл обраного музичного треку у вигляді аудіо-повідомлення.

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, круг, Шрифт  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 20 – Меню прослуховування музики |

1. Плейлисти (рисунок 21). Бот надсилає меню керування плейлистами та їх прослуховуванням.

|  |
| --- |
| Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение  Контент, сгенерированный ИИ, может содержать ошибки. |
| Рисунок 21 – Меню плейлистів |

1. Створення плейлистів (рисунок 22). Користувач пише у чат назву плейлиста бот її оброблює і зберігає.

|  |
| --- |
|  |
| Рисунок 22 – Створення плейлистів |

1. Редагування плейлистів (рисунок 23-24). Користувач обирає плейлист який хоче редагувати та додає музику до плейлиста.

|  |
| --- |
|  |
| Рисунок 23 – Процес обирання плейлиста на редагування |
|  |
| Рисунок 24 – Процес редагування плейлистів |

1. Прослуховування плейлиста (рисунок 25-26). Користувач обирає плейлист для прослуховування та натискає кнопку з назвою плейлиста.

|  |
| --- |
|  |
| Рисунок 25 – Процес обирання плейлиста |
|  |
| Рисунок 26 – Бот видає обранний плейлист |

1. За допомогою інших команд користувач може:

*/info* : Дізнатись про розробника програмного забезпечення (рисунок 27);

*/help* : Дізнатись свій унікальній ідентифікатор у мережі *Telegram* (рисунок 28).

|  |
| --- |
|  |
| Рисунок 27 – Команда info |

|  |
| --- |
|  |
| Рисунок 27 – Команда help |

Інтерфейс простий і інтуїтивно зрозумілий, що дозволяє користувачу швидко зорієнтуватися у функціоналі програми без додаткового навчання.

**ВИСНОВКИ**

У ході виконання курсового проєкту було досягнуто мету, а саме розроблено ПЗ Telegram-бот для прослуховування музики - *“NLAI”*, який вирішує наступні завдання:

* додавання аудіофайлів до БД;
* відтворювання аудіофайлів на запит користувача;
* організовування музики у вигляді плейлистів.

У процесі роботи були вирішені наступні завдання:

1. Проведено аналіз предметної області.
2. Визначено вимоги до ПЗ.
3. Описано математичний метод рішення задачі.
4. Розроблено структуру вхідних та вихідних даних.
5. Реалізовано алгоритм роботи бота з використанням мови програмування *Python* та бібліотеки *telebot.*
6. Розроблена БД для збереження інформації про музичні треки у форматі *JSON*-файлів.
7. Створено “user friendly” інтерфейс взаємодії користувача з ботом через месенджер Telegram.

У результаті тестування підтверджено стабільну роботу ПЗ *“NLAI”*, правильну обробку користувацьких запитів та функціональність визначених функціональними вимогами команд.

Розроблене ПЗ може бути використаний як основа для створення більш складних ботів із розширеним функціоналом, наприклад авторизація користувачів або інтеграція із зовнішніми музичними сервісами, видалення треків та плейлистів тощо.

**ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. *Python Software Foundation – Python 3 Documentation* [Електронний ресурс] – Режим доступу: <https://docs.python.org/3/>
2. *Van Rossum G. – PEP 257 – Docstring Conventions* [Електронний ресурс] – Режим доступу: <https://peps.python.org/pep-0257/>
3. *Telegram – Telegram Bot API* [Електронний ресурс] – Режим доступу: <https://core.telegram.org/bots/api> – Дата звернення: 26.04.2025.
4. *Telebot (pyTelegramBotAPI) – Documentation* [Електронний ресурс] – Режим доступу: <https://pytba.readthedocs.io/en/latest/>
5. *Official JSON Website – Introducing JSON* [Електронний ресурс] – Режим доступу: <https://www.json.org/json-en.html>
6. *Python Documentation – Hashlib – Secure Hash and Message Digest Algorithms* [Електронний ресурс] – Режим доступу: <https://docs.python.org/3/library/hashlib.html>
7. *Wikipedia – Telegram Bots* [Електронний ресурс] – Режим доступу: <https://en.wikipedia.org/wiki/Telegram_Bot_API>
8. *W3Schools – JSON Tutorial* [Електронний ресурс] – Режим доступу: <https://www.w3schools.com/js/js_json_intro.asp>
9. *YouTube* – Створення *Telegram*-бота на *Python* (Навчальний курс) [Електронний ресурс] – Режим доступу: <https://www.youtube.com/watch?v=6ZfuNTqbHE8>
10. *YouTube – Python Telegram Bot Tutorial: Creating a Simple Bot* [Електронний ресурс] – Режим доступу: <https://www.youtube.com/watch?v=RhKLuStY0Kw>
11. *Telegram* – Офіційний сайт [Електронний ресурс] – Режим доступу: <https://telegram.org/>
12. *Telegram – Bots*. Навчальний посібник [Електронний ресурс] – Режим доступу: <https://core.telegram.org/bots/tutorial>
13. *Python Telegram Bot Documentation* [Електронний ресурс] – Режим доступу: <https://docs.python-telegram-bot.org/en/v20.0a2/>
14. *Make.com – Telegram Bot Integration* [Електронний ресурс] – Режим доступу: <https://apps.make.com/telegram-bot>
15. *SendPulse* – Форматування тексту в чат-ботах Telegram [Електронний ресурс] – Режим доступу: <https://sendpulse.com/ru/knowledge-base/chatbot/telegram/format-text>
16. *YouTube* – Канал *Indently* – навчання створенню ботів [Електронний ресурс] – Режим доступу: <https://www.youtube.com/@indently>
17. *YouTube Playlist* – Створення *Telegram*-ботів на *Python* *(Indently)* [Електронний ресурс] – Режим доступу: <https://www.youtube.com/playlist?list=PL0lO_mIqDDFUev1gp9yEwmwcy8SicqKbt>

# ДОДАТОК А

|  |
| --- |
|  |
| Рисунок 28 – Діаграма послідовності – Головна форма програми |

# ДОДАТОК Б

З метою зручного доступу до вихідного коду та демонстрації роботи застосунку, було використано платформу *GitHub*. Це дозволяє не лише зберігати проєкт у віддаленому репозиторії, але й забезпечити відкритий перегляд, спрощене тестування та можливість колективної роботи. Посилання на репозиторій із кодом курсового проєкту представлено за наступним посиланням: <https://github.com/HiToRi4kaa/NLAI>.

# ДОДАТОК В

На рисунку 29 забрежено альтернативний та успішний сценарій додавання музики.

|  |
| --- |
|  |
| Рисунок 29 – Форма – Додавання музики |

|  |
| --- |
|  |
| Рисунок 30 – Форма – Створення плейлиста та головне меню плейлистів. |

На рисунку 30 зображено успішний сценарій створення плейлиста та головне меню плейлистів