Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

Бакалаврська дипломна робота на тему:

«ТЕЛЕГРАМ-БОТ ПАБЛІШЕР НА AIOGRAM ДЛЯ АВТОМАТИЗАЦІЇ КЕРУВАННЯ КОНТЕНТОМ У ТЕЛЕГРАМ КАНАЛАХ»

Виконав: студент IV курсу, групи
1АКІТ-20б
спеціальності 151 – Автоматизація та
комп'ютерно-інтегровані технології

(шифр і назва спеціальності)

Сметанюк Є. О.
(прізвище та ініціали)

Керівник: к.т.н., доцент кафедри АІІТ
Кабачій В. В.
(прізвище та ініціали)
«17» 06 2024 р.

Рецензент: Денисюк В.О.
к.т.н., доц. каф. КН
(прізвище та ініціали)
«18» 06 2024 р.

Допущено до захисту
завідувач кафедри АІІТ
Бісікало О. В.
«18» 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти перший (бакалаврський)
Галузь знань 15 – Автоматизація та приладобудування
Спеціальність 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітньо-професійна програма Інтелектуальні комп'ютерні системи управління

ЗАТВЕРДЖУЮ
завідувач кафедри АІІТ
д.т.н., проф. Бісікало О. В.
«13» 03 2024 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Сметанюку Євгенію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Телеграм-бот паблішер на Aiogram для автоматизації керування контентом у телеграм каналах»
керівник роботи Кабачій Владислав Володимирович, к.т.н, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «11»березня 2024 року №80

2. Термін подання студентом роботи 10.06.2024 р.
3. Вихідні дані до роботи:
- кількість запланованих повідомлень у Telegram – до 100;
- середовище розробки PyCharm Professional;
- мова програмування Python;
- бази даних MySQL, Redis

4. Зміст текстової частини (перелік питань, які потрібно розробити) провести аналіз існуючих телеграм ботів паблішерів, їх переваги та недоліки, Запропонувати варіант з врахованими недоліками аналогів, розробити блок-схему взаємодії телеграм бота, схему бази даних та програмне забезпечення.
5.Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень) Блок-схема взаємодії телеграм бота з користувачами. Схема таблиць баз даних та їх відношення.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завд прий

7. Дата видачі завдання 12.03.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Термін виконання		Пр
		початок	закінчення	
1	Вибір, узгодження та затвердження теми БДР	03.10.2023	09.10.2023	
2	Аналіз предметної області та опрацювання літературних джерел.	12.03.2024	29.03.2024	
3	Постановка задач для вирішення в БДР	01.04.2024	26.04.2024	
4	Вирішення поставлених задач	29.04.2024	10.05.2024	
5	Підтвердження достовірності отриманих результатів	13.05.2024	24.05.2024	
7	Оформлення пояснювальної записки та графічної частини	27.05.2024	31.05.2024	
8	Перевірка ПЗ на відповідність вимогам	03.06.2024	07.06.2024	
9	Попередній захист, доопрацювання, та рецензування БДР	10.06.2024	15.06.2024	
10	Захист БДР ЕК	18.06.2024	18.06.2024	

Студент _____ Сметанюк Є. О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Кабачій В. В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 88 сторінок формату А4, у тому числі і додатків, на яких є 43 рисунків, 2 таблиці, список використаних джерел містить 20 найменувань.

Досліджено процес розробки телеграм-бота паблішера, призначеного для ефективної автоматизації процесу публікації контенту у телеграм-каналах. Робота зосереджена на детальному аналізі предметної області, ідентифікації недоліків і визначенні відсутності важливого функціоналу у аналогів, зокрема з огляду на потреби контент-менеджерів та SMM-менеджерів.

Телеграм-бот реалізований з використанням мови програмування Python та бібліотеки aiogram, з інтеграцією Redis для кешування та зберігання проміжкових даних. Бот має власну базу текстів та медіа, що дозволяє ефективно керувати контентом. Основні функціональні можливості включають можливість запланувати пост на певну дату та час, встановлення затримки, вибір рандомного контенту (тексту чи медіа), використання інлайнових кнопок та інші опції для зручного управління публікаціями. Крім того, реалізована можливість планування публікацій у циклічному режимі, що дозволяє автоматизувати повторювані пости.

Розробка програмного забезпечення здійснювалась у середовищі розробки PyCharm, з використанням сучасних методів та практик програмування для забезпечення надійності, ефективності та зручності використання. Робота включає детальний аналіз та опис процесу розробки, використані технології та підходи, а також оцінку отриманих результатів.

Ключові слова: Телеграм, бот, паблішер, постинг, aiogram, соціальна мережа.

ABSTRACT

The bachelor thesis consists of 88 pages of A4 format, including appendices, which contain 43 figures, 2 tables, the list of used sources contains 20 items.

The process of developing a Telegram publisher bot designed for effective automation of the process of publishing content in Telegram channels has been studied. The work is focused on a detailed analysis of the subject area, identification of shortcomings and determination of the lack of important functionality in analogues, in particular with regard to the needs of content managers and SMM managers.

The Telegram bot is implemented using the Python programming language and the aiogram library, with Redis integration for caching and intermediate data storage. The bot has its own database of texts and media, which allows you to effectively manage content. The main functionality of the bot includes the ability to schedule a post for a certain date and time, setting a delay, choosing random content (text or media), using inline buttons and other options for convenient management of publications. In addition, the bot supports the possibility of scheduling publications in cyclic mode, which allows you to automate repeated posts.

The development of the bot was carried out in the PyCharm development environment, using modern programming methods and practices to ensure reliability, efficiency and ease of use. The work includes a detailed analysis and description of the development process, used technologies and approaches, as well as an evaluation of the obtained results.

Keywords: Telegram, bot, publisher, posting, aiogram, social network.

ЗМІСТ

ВСТУП

Актуальність. Тема розробки телеграм-бота паблішера для автоматизації постингу контенту у телеграм-каналах є надзвичайно актуальною у сучасному цифровому світі. З поглибленням інтернет-технологій та зростанням соціальних медіа платформ, важливість ефективного управління контентом стає все більш очевидною для бізнесу, маркетологів та інших фахівців у сфері цифрового маркетингу.

Автоматизовані інструменти для планування та публікації контенту, такі як телеграм-боти, стають необхідними для ефективного управління соціальними медіа каналами. Вони дозволяють зекономити час та зусилля контент-менеджерів, дозволяючи їм фокусуватися на стратегічних завданнях, а не на рутинних операціях.

Зростання конкуренції в онлайн просторі також підсилює значення швидкості та реагування на зміни в аудиторії. Телеграм-боти забезпечують можливість запланувати та публікувати контент у стратегічно важливий час, а також швидко реагувати на актуальні події або тренди.

Окрім того, зростає значення персоналізації контенту та залучення аудиторії. Телеграм-боти дозволяють налаштувати різноманітність форматів та часу публікацій, використовуючи різні канали комунікації, такі як текст, зображення, відео, анімація тощо.

Отже, розробка паблішера для автоматизації постингу у телеграм-каналах є актуальною та важливою задачею, що відповідає сучасним вимогам цифрового маркетингу та сприяє підвищенню ефективності управління контентом в онлайн середовищі.

Об’єкт дослідження: процес розроблення автоматизованого управління контентом у чат-ботах.

Предмет дослідження: методи, моделі та технологічні засоби розроблення і застосування чат-бота для автоматизації керування контентом у ньому.

Мета роботи: створення телеграм-бота, який автоматизує процес управління контентом в телеграм каналах, забезпечивши користувачам весь потрібний для цього функціонал.

Методи дослідження: аналіз і синтез, методи автоматизації управління контентом у телеграм каналах, методи розробки і моделювання бота, застосування мови програмування Python та бібліотеки aiogram з метою побудови автоматизованого чат-боту.

Основне завдання — спрощення та оптимізація рутинної роботи з публікаціями, що дозволяє економити час і підвищувати ефективність керування контентом.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Основна мета створення ПЗ полягає в тому, щоб автоматизувати процес створення постів у телеграм каналах та зекономити значну кількість часу організації яка хоче інформувати свою аудиторію через один або декілька телеграм каналів.

1.1 Аналіз актуальності розробки

У цифровій епохі, коли Інтернет є невід'ємною складовою нашого повсякденного життя, наявність у онлайн-просторі стає важливою для багатьох видів діяльності. Підприємства будь-якої форми власності, навчальні установи, рекламні агенції та інші організації потребують ефективного розміщення реклами та підтримки їхніх продуктів або послуг, привертання уваги аудиторії та інформування про події та новини.

Завдяки широкому доступу до інтернету, компанії можуть досягти своєї цільової аудиторії в будь-який час та в будь-якій точці світу. Вони можуть використовувати соціальні мережі, електронну пошту, блогінг, пошукову оптимізацію (SEO) та рекламу на інтернет-платформах. Кожен з цих шляхів має свої унікальні переваги і обмеження, і вибір методу залежить від конкретних цілей бізнесу та характеру аудиторії. Наприклад, соціальні мережі можуть бути ефективними для залучення уваги нових клієнтів, тоді як електронна пошта може бути корисною для збереження зв'язку з існуючими клієнтами. Блогінг може допомогти підвищити експертність бренду, а реклама на інтернет-платформах може допомогти залучити увагу цільової аудиторії. Телеграм-канали можуть відрізнятися своєю прямою взаємодією з аудиторією, можливістю публікації ексклюзивного контенту та високою конверсією від підписників до клієнтів. Використання різних методів просування в поєднанні може допомогти досягти найкращих результатів для бізнесу.

Зв'язок із сучасними тенденціями використання інтернет-технологій підкреслює важливість розробки та ведення телеграм-каналів. Оскільки месенджер Telegram вважається одним з найпопулярніших не тільки серед українських користувачів, а й у світі, використання цієї платформи стає привабливим засобом спілкування з аудиторією (рисунок 1.1). Створення та якісне ведення каналу в Telegram може забезпечити численні переваги для організацій та підприємств:

- Залучення аудиторії: публікація цікавого та актуального контенту допомагає залучити увагу нових користувачів до каналу.
- Швидке інформування про новини та події: канали дозволяють оперативно інформувати підписників про новини, події, акції та інші важливі аспекти діяльності організації.
- Брендування та підвищення свідомості про бренд: регулярна публікація контенту в каналі допомагає підтримувати увагу до бренду та підвищує його впізнаваність серед аудиторії.
- Взаємодія з аудиторією: канали надають можливість взаємодії з аудиторією через коментарі, опитування, голосування та інші інтерактивні функції.
- Маркетингові можливості: канали можуть бути використані для проведення маркетингових кампаній, рекламних акцій та просування продуктів або послуг.

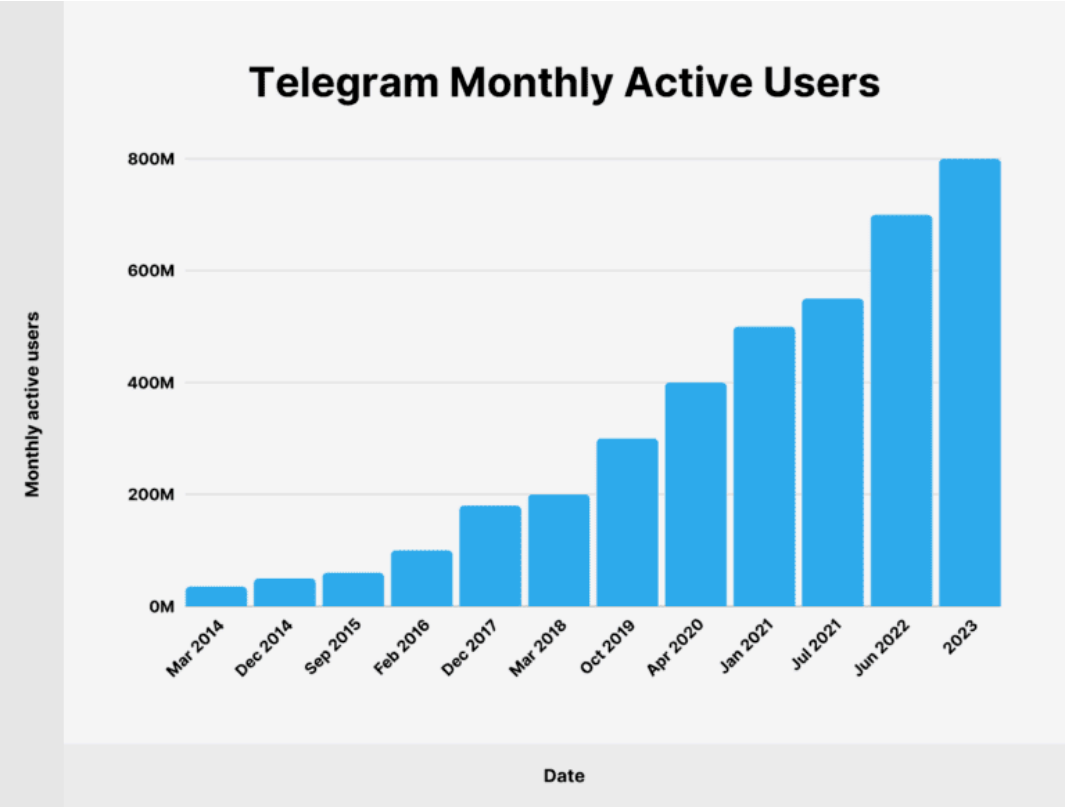
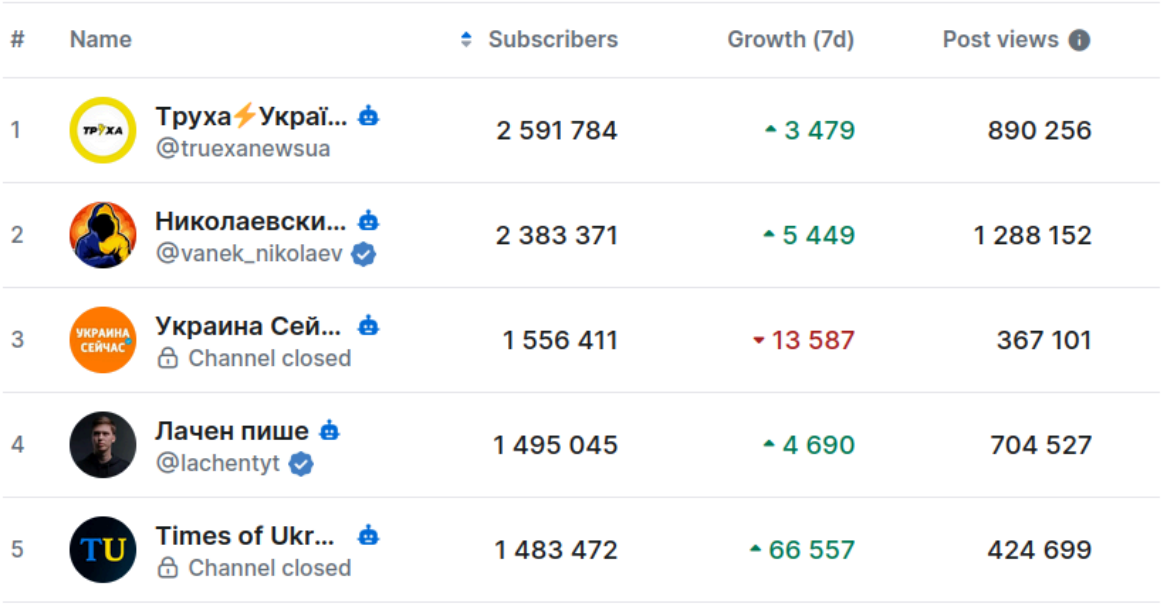


Рисунок 1.1 – Активність користувачів у телеграм з 2014 по 2023 рік

За аналітичними даними сервісу telemetr.io Україна посідає 10 місце за загальною кількістю аудиторії в телеграм каналах та 11 місце за кількістю телеграм каналів у країні. Ці показники свідчать про високу активність українських користувачів у цій платформі та про значний інтерес до використання Телеграму як засобу комунікації та отримання інформації (рисунок 1.2).



#	Name	Subscribers	Growth (7d)	Post views
1	Труха⚡Украї... @truexanewsua	2 591 784	▲ 3 479	890 256
2	Николаевски... @vanek_nikolaev	2 383 371	▲ 5 449	1 288 152
3	Украина Сей... Channel closed	1 556 411	▼ 13 587	367 101
4	Лачен пише @lachentyt	1 495 045	▲ 4 690	704 527
5	Times of Ukr... Channel closed	1 483 472	▲ 66 557	424 699

Рисунок 1.2 – Топ 5 українських телеграм каналів з найбільшою кількістю підписників станом на 17 травня 2024 року. [1].

1.2 Аналіз предметної області

Telegram – це месенджер та соціальна платформа, яка швидко набирає популярність як в Україні, так і по всьому світу. Ось кілька ключових особливостей та можливостей Telegram. [2].:

1) Безпека та конфіденційність: Telegram відомий своєю високою захищеністю та шифруванням даних. Він пропонує різні функції, такі як секретні чати та можливість встановлення терміну дії повідомлень, щоб забезпечити конфіденційність користувачів.
2) Групові чати та канали: користувачі можуть створювати групові чати для спілкування з друзями, колегами або спільнотами, а також канали для розповсюдження контенту широкій аудиторії.
3) Боти та інтеграції: Telegram має розгорнуту систему ботів, які дозволяють автоматизувати різні завдання, такі як нагадування, пошук інформації, організація зборів тощо. Більше того, він інтегрується з іншими сервісами, що дозволяє зручно спілкуватися з іншими платформами.

4) Функції обміну контентом: Telegram підтримує різноманітні типи контенту, включаючи текст, фотографії, відео, аудіо, стікери, файли тощо. Користувачі можуть легко ділитися різними матеріалами з іншими користувачами.
5) Поширена популярність: Україна – одна з країн, де Telegram має велику популярність серед користувачів месенджерів. Це створює сприятливі умови для використання цієї платформи як інструменту для комунікації, спілкування та розповсюдження контенту.

Щодо безпеки, то у Telegram шифрування даних грає важливу роль у забезпеченні конфіденційності та безпеки користувачів. Основна функція шифрування – це захист інформації від несанкціонованого доступу під час її передачі через мережу Інтернет.

Telegram використовує протокол шифрування MTProto, який розроблений самими розробниками платформи. Він використовує сучасні криптографічні алгоритми для забезпечення високого рівня безпеки даних. Цей протокол забезпечує кінець-до-кінця шифрування, що означає, що дані шифруються на пристрої відправника і розшифровуються лише на пристрої отримувача, уникнувши можливого перехоплення чи зчитування даних третьою стороною під час їх передачі.

Шифрування в Telegram також включає функції, такі як секретні чати, які дозволяють користувачам встановлювати додатковий рівень захисту для своїх приватних розмов. У секретних чатах використовується потрійне шифрування, що робить їх особливо надійними. Також існує функція "знищення повідомлень", яка дозволяє користувачам автоматично видаляти відправлені повідомлення через певний час після їхнього перегляду.

В цілому, шифрування даних у Telegram є важливим елементом безпеки, який допомагає забезпечити приватність і конфіденційність користувачів під час спілкування та обміну інформацією в месенджері.

Telegram надає широкі можливості для спілкування, обміну контентом та автоматизації завдань, що робить його популярним серед користувачів та бізнесів у всьому світі. Також Telegram є відкритою платформою для розробників. Він надає API, що дозволяє створювати власні боти, додатки та розширення. Це відкриває безліч можливостей для створення власного функціоналу та інтеграції з іншими сервісами.

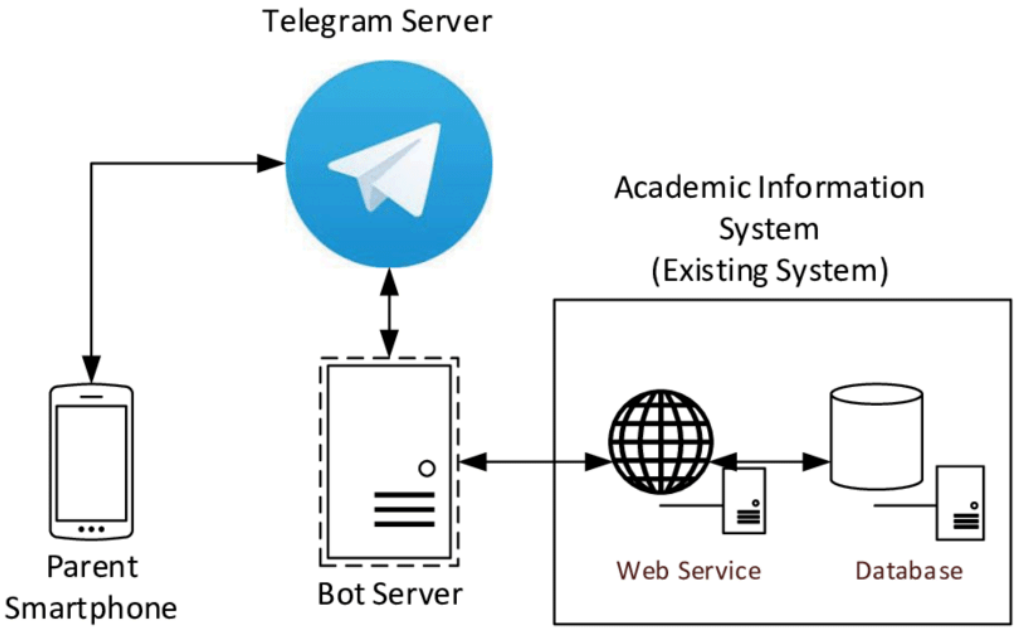


Рисунок 1.3 – Клієнт-серверна архітектура Telegram API

API Telegram бота забезпечує зв'язок між користувачем і ботом. Коли користувач надсилає повідомлення, воно проходить через сервери Telegram, які визначають, якому боту його направити (рисунок 1.3). Бот може отримувати повідомлення через вебхуки або полінг.

При використанні вебхуків сервер Telegram надсилає повідомлення на вказану URL-адресу бота. При полінгу бот періодично запитує сервер Telegram про нові повідомлення.

Як тільки бот отримує повідомлення, воно обробляється згідно з логікою, закладеною в коді бота. Бот може виконувати різні дії, такі як витягування тексту або медіафайлів, збереження даних у базі даних або надсилання

відповідей користувачу через функції API Telegram (sendMessage, sendPhoto, тощо).

Телеграм канали – це медіа-канали у месенджері Telegram, де адміністратори можуть публікувати контент для своїх підписників. Канали можуть бути створені для різних цілей, таких як поширення новин, інформаційних повідомлень, аналітичних матеріалів, реклами, розважального контенту, освітніх матеріалів тощо. [3]. У телеграм-каналах користувачі не мають можливості надіслати повідомлення у канал, якщо вони не є адміністратором, оскільки ці канали призначені для односторонньої розсилки інформації від адміністратора до підписників. Однак, якщо адміністратор каналу дозволив коментування під постами, то користувачі можуть обговорювати контент, опублікований у каналі, в коментарях. Це надає можливість аудиторії висловлювати свої думки, задавати питання та обмінюватися думками з іншими учасниками каналу. Такий механізм коментування під постами дозволяє створювати активну спільноту навколо контенту, що публікується, і залучати аудиторію до діалогу та обміну думками.

Телеграм-боти – це програмне забезпечення, яке дозволяє автоматизувати багато різних завдань у месенджері Telegram. Вони можуть мати різноманітні функції, такі як надсилання користувачам інформації про новини, погоду, курси валют тощо, проведення опитувань і збір відгуків від користувачів, надсилання регулярних оновлень або сповіщень, можуть надавати підтримку клієнтам і відповіді на питання, реалізація ігор, розважальних сервісів і багато іншого. Ці боти можуть бути створені для різних цілей, від особистого використання до бізнесу та корпоративного сектору. Вони дозволяють автоматизувати багато рутинних процесів і полегшують взаємодію з користувачами через месенджер. В залежності від потреби, бот також може виконувати певні завдання у телеграм-каналах або телеграм-групах. [4].

Телеграм-бот паблішер – це інструмент для автоматизації процесу публікації контенту в телеграм-каналах. Телеграм-бот паблішер став

незамінним інструментом для контент-менеджерів та SMM-менеджерів у сфері маркетингу та комунікацій. Він значно спрощує їхню роботу, дозволяючи ефективно керувати публікаціями та вибудовувати розклад постингу, що важливо для підтримки активної комунікації з аудиторією через Telegram.

Паблішери дозволяють планувати та автоматизувати процес публікації контенту, що дозволяє заздалегідь підготувати матеріали і визначити оптимальний час для їхньої публікації. Це дозволяє забезпечити регулярне оновлення каналу і підтримувати інтерес аудиторії.

Основні функції телеграм-бота паблішера включають можливість планувати час та дату публікації, керувати списком контенту, а також вивчати статистику публікацій. Деякі паблішери можуть навіть автоматично оптимізувати час публікації, враховуючи активність аудиторії.

Для організацій та підприємств, які використовують Telegram для комунікації зі своєю аудиторією, використання телеграм-бота паблішера стає незамінним інструментом для підтримки і підвищення ефективності їхньої комунікаційної стратегії. Він допомагає зекономити час та зусилля, що дозволяє зосередитися на створенні якісного контенту та взаємодії з аудиторією.

Основні функції ботів паблішерів:

1. Автоматизація постингу: боти паблішери можуть автоматично публікувати контент у заданий час, дату та якщо потрібно з затримкою. Це дозволяє заздалегідь планувати публікації та підтримувати активність каналу без постійної зайнятості.
2. Інлайн-кнопки та взаємодія: боти підтримують додавання інлайн-кнопок до публікацій, що дозволяє створювати інтерактивний контент і забезпечувати зворотний зв'язок від аудиторії. Наприклад, користувачі можуть голосувати, переходити за посиланнями або взаємодіяти з іншими елементами публікацій.

3. Рандомізація контенту: можливість рандомізації контенту, що дозволяє випадково вибирати медіа або текстові матеріали для публікації з певного каталогу створеного заздалегідь.
4. Вбудовані каталоги медіа (бази медіа): можливість створити певний каталог у який можна завантажити заготовлені текста, медіафайли, голосові повідомлення та відеоповідомлення. Це зручно тим, що при кожному створенні нового посту не потрібно завантажувати новий файл, а можна обрати заздалегідь завантажений з потрібного каталогу.
5. Циклічні публікації: можливість налаштування циклічних публікацій дозволяє регулярно повторювати певний контент у встановлені дні та години. Це корисно для нагадування про важливі події або продукти.
6. Власна база каналів: один бот може керувати багатьма телеграм каналами незалежно.
7. Швидкий постинг: якщо потрібно одразу опублікувати пост.

SMM – це абревіатура, яка означає "соціальний медіа маркетинг" (Social Media Marketing). Це стратегія маркетингу, яка використовує соціальні медіа платформи, такі як Facebook, Instagram, Twitter, LinkedIn, YouTube та інші, для просування брендів, продуктів або послуг.

Основна мета SMM полягає в тому, щоб створити взаємодію з аудиторією, залучити її увагу та збільшити свідомість про бренд. Це досягається через публікацію цікавого та корисного контенту, взаємодію з користувачами, відповіді на їх запитання та коментарі, організацію конкурсів та акцій, а також використання рекламних інструментів платформ для залучення нової аудиторії.

SMM є важливою складовою частиною цифрового маркетингу, оскільки соціальні медіа платформи відіграють ключову роль у спілкуванні з аудиторією та формуванні її думок та уявлень про бренд.

Найкращі практики SMM включають в себе:

1. Підготовка ресурсу:
    - Підготовка профілів на обраних платформах та налаштування їх для загального доступу.
    - Надання корисної та відповідної інформації користувачам, які стежать за вашим профілем.
    - Регулярне оновлення публікацій та використання актуальних фотографій та контенту.
2. Залучення аудиторії:
    - Публікація ексклюзивного контенту та проведення акцій на різних соціальних мережах.
    - Збереження балансу між обговоренням вашої компанії та загальним життям/індустрією у ваших публікаціях.
    - Розміщення повідомлень на різних платформах та каналах.
3. Зацікавленість аудиторії:
    - Швидка відповідь на питання, коментарі та зауваження - протягом кількох хвилин.
    - Наявність кваліфікованих працівників цілодобово.
    - Ввічлива, уважна та професійна взаємодія.

Може виникнути питання, для чого створювати бот для постингу, якщо все це можна робити в телеграмі. У телеграмі є вбудована функція планування повідомлення але немає можливості зациклити пост. Тобто, при потребі, для того, щоб кожен день виходило 150 постів з певною тематикою медіа та текстом протягом місяця, потрібно вручну запланувати 4500 постів. А якщо це потрібно зробити у 5 каналах то це 22500 постів. Більше того, у телеграмі є обмеження щодо відкладених (запланованих) повідомлень, а саме до 100 повідомлень у кожному чаті.

Було проведено дослідження, на реальному телеграм каналі у якому виходить в середньому 150 різних постів на день. Для цього знадобиться створити всього 9 постів, які будуть автоматично виходити кожного дня з

різними медіафайлами. Якщо це робити вручну, потрібно було б створити 22500 постів, це більше рівно у 2500 разів (рисунок 1.4)

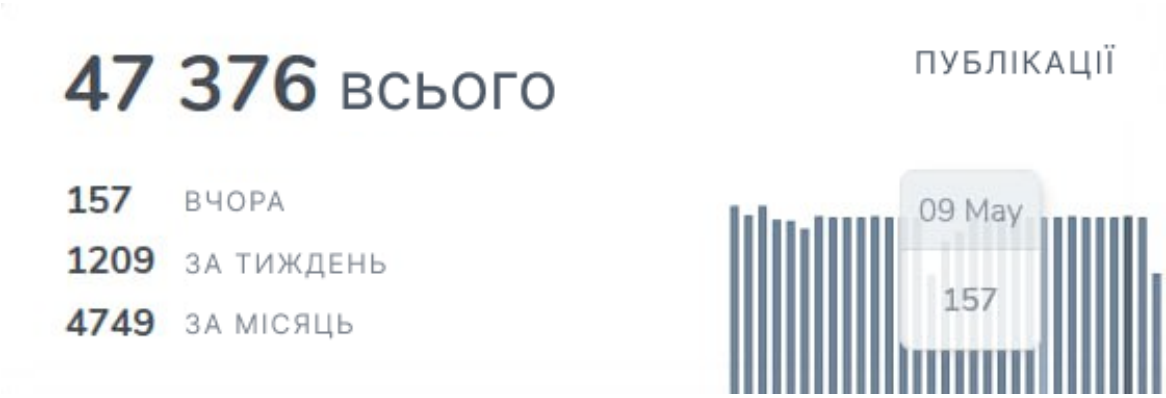


Рисунок 1.4 – Статистика виходу постів у досліджуваному телеграм каналі у сервісі (TGStat). [6].

1.3 Аналіз аналогів та конкурентних методів

BaslayBot – це безкоштовний Telegram бот для відкладеного постингу, який пропонує розширені можливості для оформлення постів. Бот дозволяє редагувати альбоми, змінювати та налаштовувати кнопки, додавати водяні знаки до фото, відео та альбомів, а також створювати кнопки для коментарів з реакціями. Крім цього, BaslayBot підтримує контент-плани для відкладених і опублікованих постів, таймери для відкріплення постів, кілька варіантів авто підпису та автоматичний прийом заявок. Користувачі можуть створювати власні боти для відкладеного постингу за допомогою команди /newbot та налаштовувати їх відповідно до своїх потреб (рисунок 1.5). [14].

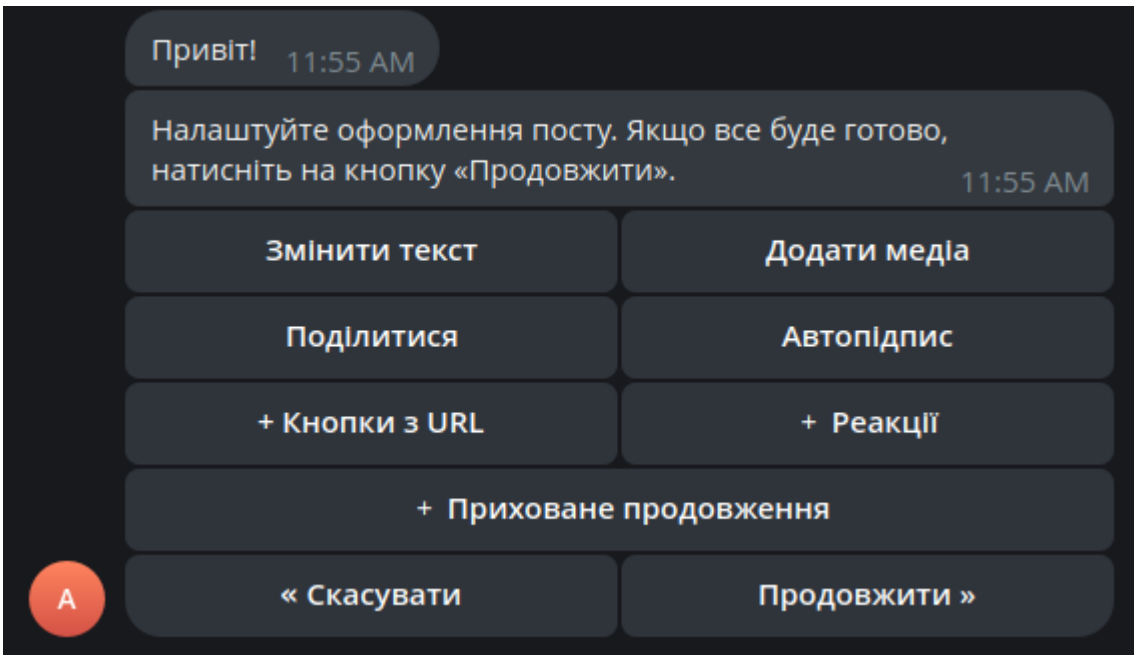


Рисунок 1.5 – Головне меню паблішера Baslay

Основні функції включають:

- Редагування та організація альбомів і медіафайлів.
- Додавання водяних знаків на фото і відео.
- Налаштування кнопок для постів.
- Відкладений постинг з автоматичним підписом і таймером для відкріплення постів.

Може здаватися, що у боті Baslay є весь необхідний функціонал але функція циклічного постингу є платною. Це дозволяє користувачам бота планувати регулярні публікації тільки за додаткову оплату, що є важливим фактором для багатьох контент-менеджерів та SMM-спеціалістів.

По-друге, Baslay не підтримує прийом відеоповідомлень та голосових повідомлень, що обмежує можливості користувачів у створенні різноманітного контенту. Це значно стискає функціональність і робить менш інтерактивний та багатогранний підхід до ведення Telegram-каналів.

Controller Bot - інструмент, призначений для допомоги власникам Telegram-каналів у керуванні та плануванні публікацій. Він спрощує процес

підтримки активності каналу, дозволяючи користувачам створювати пости заздалегідь і планувати їхнє майбутнє публікування, що забезпечує безперервну взаємодію з аудиторією навіть у відсутності адміністратора каналу. Однією з ключових функцій Controller Bot є можливість надання детальної статистики щодо активності каналу, наприклад, відстеження кількості активних учасників з часом. Ці дані допомагають власникам каналів краще розуміти свою аудиторію та відповідно налаштовувати свій контент (рисунок 1.6). [15]

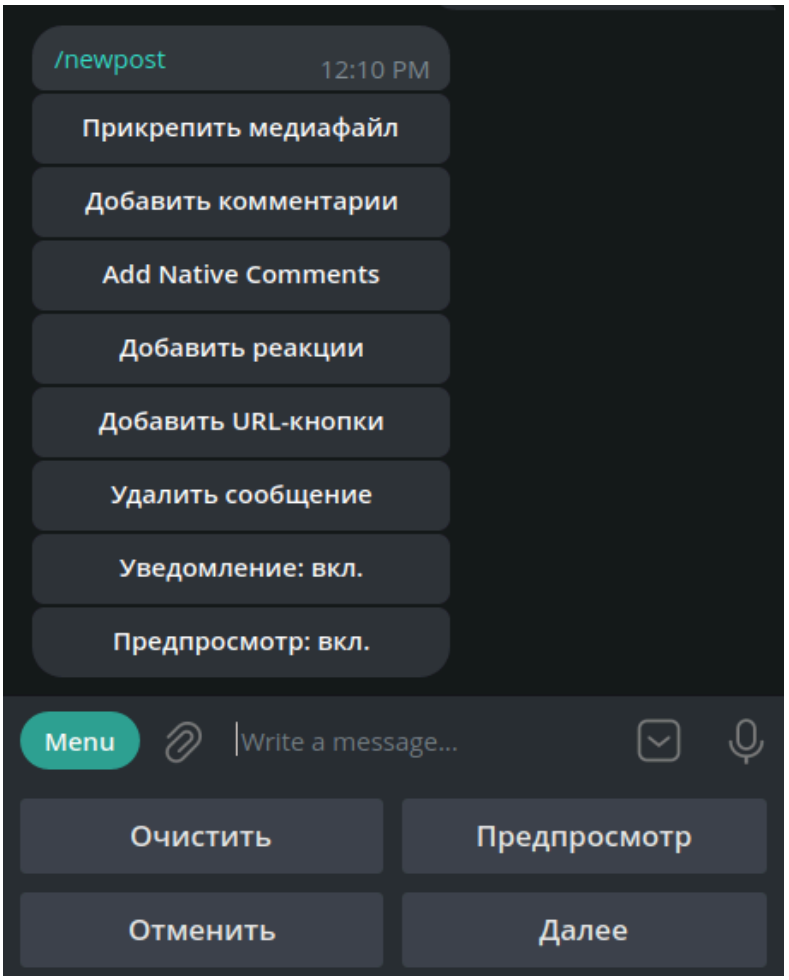


Рисунок 1.6 – Головне меню Controller Bot

З мінусів можна виділити інтуїтивно незрозумілий інтерфейс, який може сповільнювати процес планування постингу. Також цей бот не підтримує формати голосових повідомлень та відеоповідомлень, що є важливим фактором.

Notepost - переймає функціонал у BaslayBot, додає кілька інтеграцій та додає до нього статистику, яка є у боті Controller Bot (рисунок 1.7). [16].

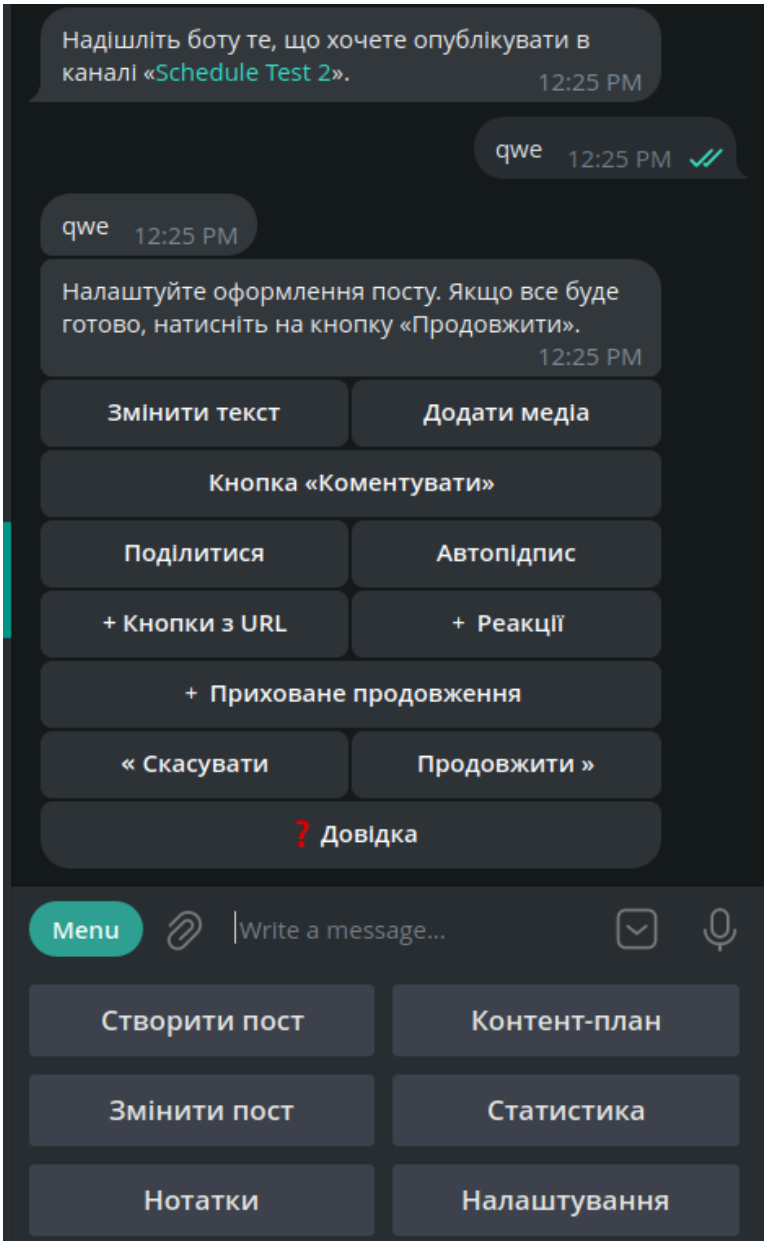


Рисунок 1.7 – Головне меню Notepost

Notepost це хороший допоміжний сервіс з великим набором цікавих функцій, але недоліком є те, що бот платний. За кожен канал потрібно платити 3$ на місяць. Також набір кнопок на головному меню виглядає досить навантажено, і з точки зору UX клавіатуру варто створювати більш інтуїтивно зрозумілою.

Отже, було проаналізовано та враховано всі недоліки та слабкі сторони аналогів. На основі цього дослідження буде спроектовано власний Telegram бот паблішер, який пропонує вдосконалені функції для автоматизації постингу.

Новий бот надаватиме безкоштовну функцію циклічного постингу, буде приймати відеоповідомлення та голосові повідомлення, що робить його більш гнучким та зручним. Завдяки врахуванню цих аспектів, бот забезпечуватиме ефективніше та інтуїтивніше управління Telegram-каналами, перевершуючи існуючі рішення на ринку.

1.4 Постановка задачі

Розробити телеграм-бота для автоматизації постингу в телеграм-каналах. Розроблений телеграм-бот повинен забезпечувати ефективне управління контентом в телеграм-каналах, спрощуючи процес постингу для контент-менеджерів та SMM-спеціалістів. Бот повинен мати інтуїтивно зрозумілий інтерфейс та широкий функціонал для роботи з текстами, медіа та плануванням публікацій. Використання сучасних технологій та бібліотек має забезпечити високу продуктивність та гнучкість у роботі бота.

1. Розробка структури бота у вигляді блок-схеми.
2. Створення інтерфейсу користувача:
    - Розробити головне меню бота з основними функціями: створення постів, редагування постів, перегляд запланованих постів, управління телеграм-каналами, доступ до бази даних текстів та медіа.
3. Функціонал постингу:
    - Реалізувати можливість створення постів з текстом, медіа-файлами (фото, відео, аудіо) та інлайновими кнопками.
    - Забезпечити функцію планування постів на конкретну дату та час.
    - Додати можливість встановлення затримки між постами та циклічного планування публікацій.
    - Реалізувати вибір випадкових медіа-файлів або текстів із бази даних для урізноманітнення контенту.

4. Управління контентом:
    - Розробити функціонал для додавання телеграм-каналів.
    - Забезпечити зручний доступ до бази текстів та медіа-файлів для швидкого створення контенту.
5. Технічна реалізація:
    - Використати мову програмування Python для розробки бота.
    - Застосувати бібліотеку aiogram для взаємодії з Telegram API та забезпечення асинхронної обробки запитів.
    - Під’єднати базу даних Redis для кешування та зберігання проміжних даних, що забезпечить швидкий доступ до необхідної інформації.
    - Розробка повинна здійснюватися в середовищі Pycharm для забезпечення зручного інтерфейсу та корисних інструментів для відлагодження та тестування коду.
6. Тестування та відлагодження:
    - Провести тестування бота для забезпечення стабільної роботи та виявлення можливих помилок.
    - Відлагодити код для забезпечення максимальної ефективності та продуктивності роботи бота.

2 АНАЛІЗ МЕТОДІВ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Методи розробки

Існує два шляхи створення телеграм боту, це програмна реалізація та за допомогою конструктора. Конструктори, які пропонуються різними платформами, надають широкий набір можливостей для створення ботів без необхідності програмування. Вони зазвичай мають інтуїтивний інтерфейс, що дозволяє користувачам швидко та легко втілювати свою ідею, встановлювати правила їхньої поведінки та налаштовувати реакції на різні команди або події у вигляді графічних ланцюжків без знання програмування. Однією з переваг конструкторів є швидкість розгортання - вони дозволяють створювати ботів за короткий час без потреби в програмуванні. До того ж, деякі конструктори мають готові шаблони та функції, які можна використовувати для створення різних типів ботів, від простих до складних.

SendPulse – це платформа для автоматизації маркетингу, яка надає інструменти для створення та керування різними видами комунікацій, такими як email-розсилки, SMS-повідомлення, web push-повідомлення та месенджери. Основна мета SendPulse – допомогти бізнесам ефективно взаємодіяти зі своєю аудиторією, автоматизуючи процеси розсилок та покращуючи залученість користувачів. Також у сервіса є можливість створення телеграм ботів за допомогою вбудованого конструктора. [7].

Щоб створити бот у сервісі SendPulse, потрібно спочатку звернутись до телеграм бота BotFather та створити у ньому токен нового бота (рисунок 2.1). Потім отриманий токен ми маємо надати SendPulse. [8].

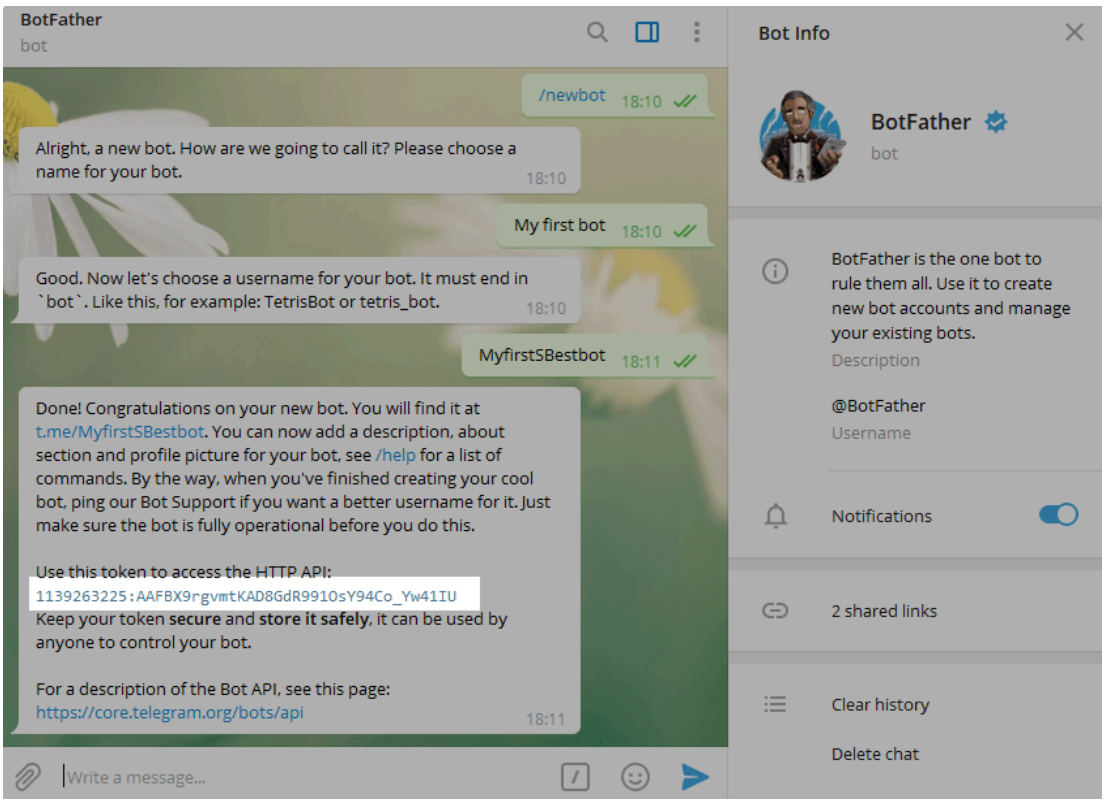


Рисунок 2.1 – Отримання токену

Далі у розділі Telegram у SendPulse вставляємо токен у форму та підключаємо бота (рисунок 2.2).

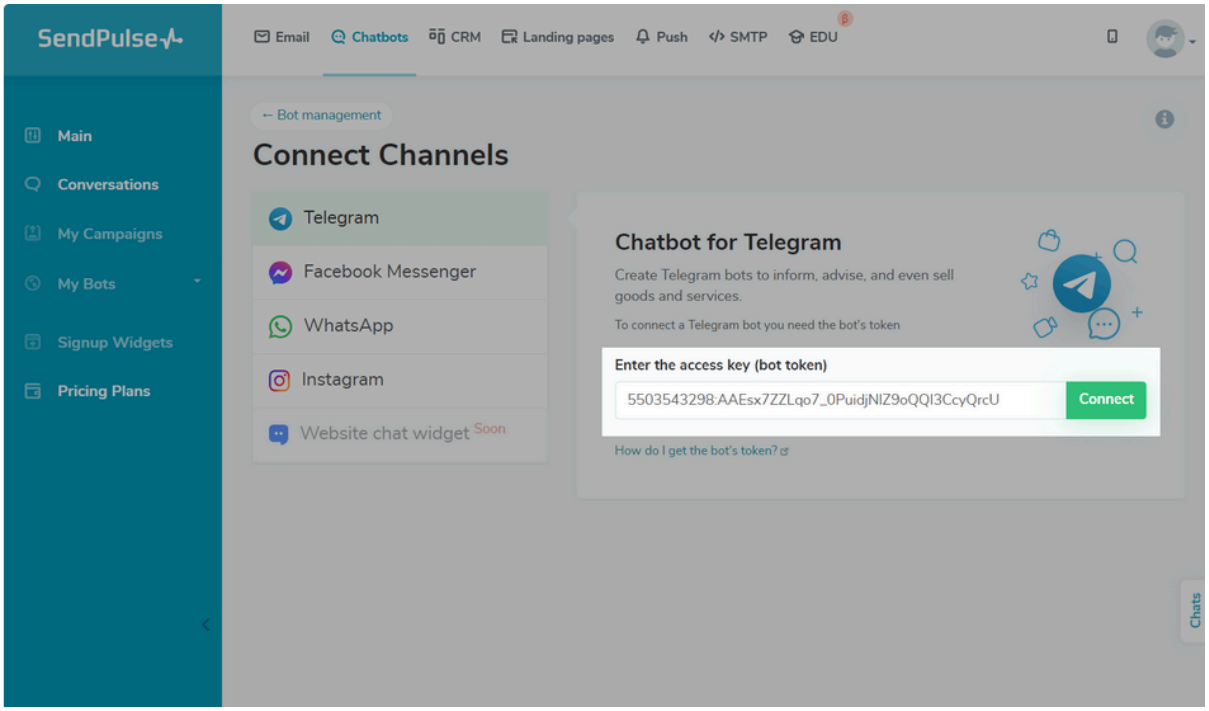


Рисунок 2.2 – Процес під’єднання

Після цього можна отримати вікно конструктора у якому можна створювати різні сценарії (рисунок 2.3).

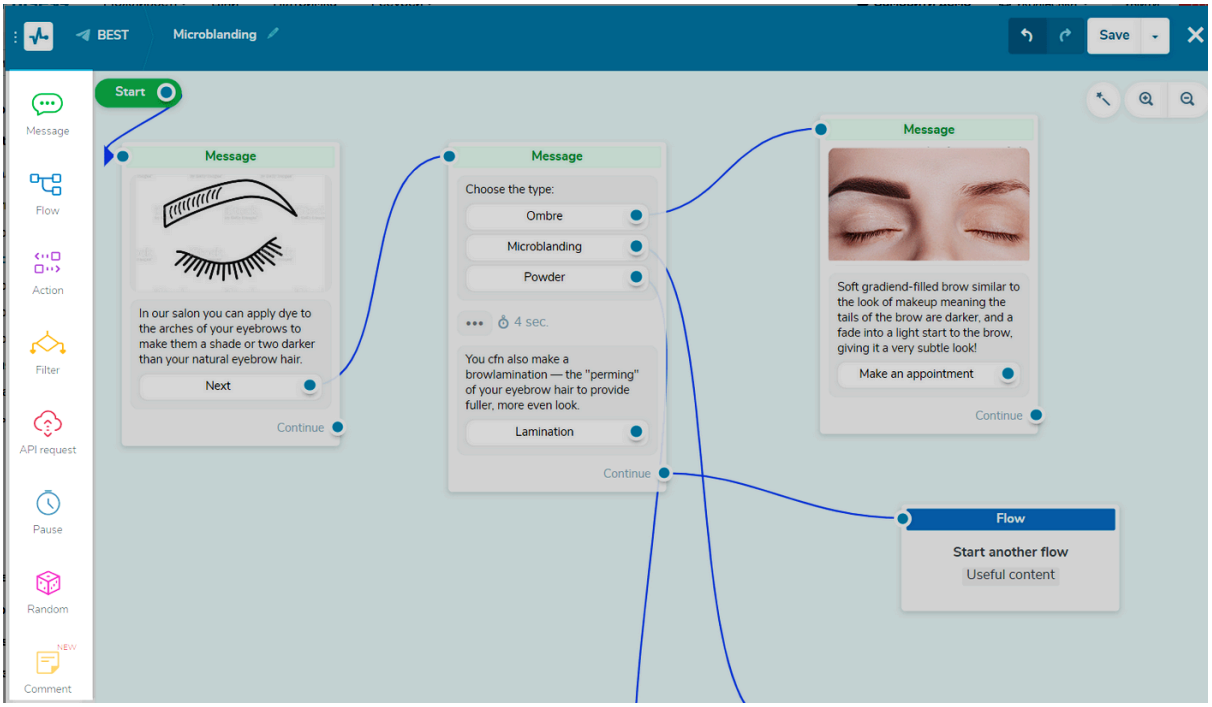


Рисунок 2.3 – Приклад створення бота за шаблоном

Проте цей метод може мати певні недоліки. По-перше, оскільки конструктори надають обмежений функціонал порівняно з програмною реалізацією, можуть виникнути проблеми в специфічній обробці певних даних та гнучкому налаштуванні. Деякі продвинуті функції можуть бути недоступними у конструкторі. По-друге, ви стаєте залежними від стороннього сервісу, який надає конструктор. Тому може виникнути ризик, що сервіс припинить свою роботу або змінить свої умови користування та тарифи, що може вплинути на бота.

До того ж, конструктори можуть мати обмежену підтримку або важкість у розумінні документації, що може ускладнити вирішення проблем та розвиток бота у майбутньому.

Натомість, програмна реалізація телеграм-бота надає більшу гнучкість та контроль над функціональністю та відкриває широкі можливості для розробників. Вони мають повний контроль над функціональністю і поведінкою бота, оскільки можуть взаємодіяти безпосередньо з Telegram API. Це дозволяє реалізувати різноманітні функції, включаючи обробку повідомлень, реагування на команди користувачів, роботу з базами даних та медіафайлами. Взаємодіяти з Telegram API можна за допомогою різних мов програмування, але якщо потрібно розробити бота швидко та якісно, надається перевага мові програмування Python. У Python є багато різних бібліотек які вдосконалюють та спрощують взаємодію з API.

2.1.1 Реалізація програмним шляхом:

Переваги:

- Гнучкість: повний контроль над функціоналом та логікою свого бота. Ви можете налаштувати його точно під свої потреби та вимоги.
- Розширені можливості: можливість використовувати будь-які бібліотеки та інструменти Python для створення розширеного функціоналу, такого як аналітика, обробка даних, інтеграція з іншими сервісами тощо.
- Доступ до API: прямий доступ до Telegram API, що дозволяє максимально використовувати можливості платформи та створювати різноманітний функціонал.

  Недоліки:
- Складність розробки: необхідно мати базові знання програмування та розуміння принципів роботи Telegram API.
- Часові витрати: створення бота програмним шляхом може зайняти більше часу, особливо якщо ви новачок у програмуванні.

2.1.2 Реалізація за допомогою конструктора:

Переваги:

- Простота використання: конструктори надають інтуїтивно зрозумілий інтерфейс, що дозволяє створювати ботів без необхідності в програмуванні.
- Швидкість розгортання: за допомогою конструктора можна створити простого бота за декілька хвилин.
- Можливість роботи без програмування: навіть люди без технічних навичок можуть створити свого бота за допомогою конструктора.

Недоліки:

- Обмежений функціонал: конструктори зазвичай мають обмежений функціонал порівняно з програмною реалізацією, що може обмежити реалізацію запланованих ідей та можливості налаштування.
- Залежність від стороннього сервісу: бот стає залежними від конструктора, і його припинення роботи може призвести до проблем з ботом.

2.2 Вибір мови програмування

Телеграм ботів можна писати на різних мовах програмування, залежно від уподобань розробника та вимог проекту. Перед розробкою було розглянуто декілька популярних мов програмування та бібліотек, які використовуються для створення Telegram ботів. [9].

Python є однією з найпопулярніших мов для створення Telegram ботів завдяки простоті синтаксису та великій кількості доступних бібліотек.

- Aiogram: асинхронна бібліотека для розробки ботів з високою продуктивністю.

- python-telegram-bot: синхронна бібліотека з широким функціоналом і підтримкою останніх оновлень API Telegram.
- Telethon: бібліотека для роботи з API Telegram і управління обліковим записом користувача.

JavaScript та його надбудова TypeScript також широко використовуються для створення ботів завдяки популярності Node.js.

- Telegraf: потужна бібліотека для створення ботів на Node.js з підтримкою асинхронних функцій.
- Grammy: бібліотека для створення ботів з TypeScript, яка надає типізований доступ до API Telegram.

PHP є популярною мовою для веб-розробки і також може використовуватися для створення Telegram ботів.

- MadelineProto: бібліотека для створення ботів і клієнтів на PHP.
- Telegram Bot API PHP SDK: проста бібліотека для інтеграції з API Telegram.

Java забезпечує високу продуктивність та надійність, що робить її підходящою для створення складних ботів.

- TelegramBots: бібліотека для створення Telegram ботів на Java з підтримкою основних функцій API Telegram.

C# та .NET часто використовуються для корпоративних рішень, включаючи створення Telegram ботів.

- Telegram.Bot: офіційна бібліотека для створення ботів на .NET з підтримкою всіх функцій API Telegram.

Go (Golang) відомий своєю ефективністю та швидкістю, що робить його підходящим для створення високопродуктивних ботів.

- tgbotapi: популярна бібліотека для створення Telegram ботів на Go.
- telebot: проста у використанні бібліотека для розробки ботів на Go.

Java та Golang краще обирати під проект в якому планується велике навантаження та масштабний проект, де потрібно виконувати багато ресурсоємних завдань. Мінусом є те, що потрібно пожертвувати швидкістю розробки через складність синтаксису.

PHP в даному випадку не підходить по причині відсутності асинхронності, тому швидкодія буде меншою. Також у PHP менш розвинені бібліотеки для створення телеграм ботів та відповідно менше ком’юніті.

Python – це високорівнева мова програмування загального призначення, створена Гвідо ван Россумом і вперше випущена в 1991 році. Вона підтримує кілька парадигм програмування, таких як об'єктно-орієнтоване, процедурне, аспектно-орієнтоване та функціональне програмування. З моменту створення Python став однією з найпопулярніших мов програмування у світі, що використовується в різноманітних сферах, включаючи веб-розробку, науку про дані, автоматизацію, розробку ігор, штучний інтелект та багато іншого.

Проаналізувавши різні мови програмування для створення телеграм бота, було вирішено обрати Python. Це обґрунтоване рішення базується на кількох ключових факторах.

Основні характеристики Python:

- Простота та читабельність: синтаксис Python спрямований на те, щоб бути легко читаним та зрозумілим.
- Велика стандартна бібліотека: Python містить багато модулів та пакетів для вирішення різноманітних завдань, від роботи з інтернет-протоколами до обробки файлів.
- Інтерактивність та інтеграція: Python легко інтегрується з іншими мовами та платформами, такими як C, C++, Java та інші.

Перш за все, ця мова відома своєю простотою та легкістю у вивченні, що робить її ідеальним вибором для розробників будь-якого рівня досвіду. Читабельний синтаксис Python дозволяє швидко розробляти та впроваджувати нові функціональні можливості.

Python також має багатий набір бібліотек та фреймворків, які значно спрощують процес розробки телеграм ботів. Зокрема, бібліотека aiogram, яка призначена для інтеграції з Telegram API та асинхронної обробки запитів, робить боти на Python більш ефективними та швидкими у відповіді на запити користувачів.

Окрім цього, є підтримка асинхронного програмування, що є важливою перевагою для ботів, які повинні обробляти велику кількість запитів одночасно. Завдяки бібліотеці asyncio можна легко реалізувати асинхронну поведінку в коді на Python.

Іншим важливим аспектом є активна спільнота розробників Python, що забезпечує великий обсяг документації, прикладів та підтримки. Це значно прискорює процес розробки, оскільки завжди можна знайти готові рішення або отримати допомогу від інших програмістів.

Завдяки цим перевагам, Python став оптимальним вибором для створення телеграм бота, здатного ефективно обробляти запити користувачів, зберігати дані у базі даних та забезпечувати високу продуктивність і надійність роботи.

2.3 Вибір середовища розробки

Редактори коду – це програми для написання та редагування коду з додатковими можливостями, такими як форматування, автозавершення та підсвічування синтаксису.

У інтегрованих середовищ розробки (IDE) функціональність ширша, ніж у редакторів коду, але вони потребують більше системних ресурсів. Середовище розробки для Python зазвичай включає редактор коду, налагоджувач та компілятор. Існують IDE, які спеціалізуються на Python, але більшість підтримує кілька мов програмування.

Редактори коду краще підходять для створення невеликих програм, тоді як IDE призначені для роботи з масштабними проектами.

Середовище розробки, або IDE (Integrated Development Environment), – це програмне забезпечення, що об'єднує всі необхідні інструменти для написання, редагування, тестування та налагодження програмного коду в одному інтерфейсі. Воно створено для того, щоб полегшити та прискорити процес розробки програмного забезпечення, забезпечуючи розробників усіма необхідними функціями в одному місці [18].

Вибір середовища розробки для роботи з Python є важливим кроком, який може значно вплинути на ефективність та комфорт розробника. Існує безліч середовищ розробки, кожне з яких має свої унікальні переваги та функціональні можливості.

PyCharm – це популярне інтегроване середовище розробки (IDE) для Python, розроблене компанією JetBrains. Воно пропонує широкий спектр функцій, таких як автозаповнення коду, рефакторинг, інструменти для налагодження та тестування, а також інтеграцію з системами контролю версій, як-от Git. PyCharm має безкоштовну версію Community та платну версію Professional, яка включає додаткові можливості для веб-розробки та підтримки баз даних.

Переваги:

- Розширений функціонал: PyCharm надає широкий набір функцій для розробки, включаючи автозаповнення, налагодження, аналіз коду та підтримку систем контролю версій.
- Спільнота та документація: Велика активна спільнота користувачів PyCharm та докладна документація забезпечують доступність ресурсів для вирішення проблем та отримання порад.
- Безкоштовне користування Professional версією при наявності студентського.

Недоліки:

- Великі вимоги до ресурсів: PyCharm може бути важким для використання на слабших комп'ютерах через великий обсяг використаних ресурсів.

Spyder – це IDE, спеціалізоване на наукових обчисленнях та аналізі даних, зручне для використання завдяки своєму інтерфейсу, аналогічному до Matlab.

Переваги:

- Орієнтація на наукові обчислення: Spyder має інтегровану підтримку для наукових бібліотек, що робить його ідеальним вибором для аналізу даних та обчислень.
- Простота використання: Інтерфейс Spyder схожий на Matlab, що робить його зрозумілим для користувачів, знайомих з цим середовищем.

Недоліки:

- Обмежений функціонал для загального використання: У порівнянні з PyCharm, Spyder може мати обмежені можливості для загального використання, особливо за межами наукових досліджень.

PyDev – це плагін для Eclipse, який додає підтримку Python, забезпечуючи інтеграцію з цим вже відомим середовищем розробки.

Переваги:

- Інтеграція з Eclipse: Для користувачів, які вже знайомі з Eclipse, використання PyDev є логічним вибором, оскільки це плагін для цього середовища розробки.

Недоліки:

- Важкість використання: Для тих, хто не знайомий з Eclipse, може виникнути важкість в оволодінні цим середовищем.

IDLE – просте середовище розробки, поставляється разом з Python та підходить для початківців або швидкого прототипування.

Переваги:

- Простота: IDLE має дуже простий інтерфейс та легкий використання, що робить його відмінним для початківців.

Недоліки:

- Обмежені можливості: IDLE може бути обмеженим у функціоналі порівняно з іншими середовищами розробки.

Wing – це IDE для Python, яке пропонує розширені функції для розробки, такі як налагоджувач, підтримка автозаповнення коду, рефакторинг та інструменти для тестування. Wing також забезпечує інтеграцію з різними системами контролю версій та підтримку веб-розробки.

Переваги:

- Продуктивність: Wing має великий набір функцій, включаючи потужний налагоджувач та підтримку веб-розробки.
- Інтеграція з іншими інструментами: Wing легко інтегрується з різними системами контролю версій та іншими інструментами для розробки.

Недоліки:

- Платність: Одним з головних недоліків Wing є те, що деякі з його найпотрібніших функцій доступні тільки у платних версіях.

Таблиця 2.1 – Порівняння середовищ розробки

IDE	Ціна	Розмір (МБ)	Підтримка мов
PyCharm	Безкоштовно або $250/рік	150-176 MB	JavaScript, Node.js, CoffeeScript, TypeScript, File Watchers, Dart, RESTful Web Services, Emmet, HTML, Style Sheets
Spyder	Безкоштовно	361-427 MB	Python
PyDev	Безкоштовно	300 MB	Python, C++, Coffeescript, HTML, Javascript, CSS
IDLE	Безкоштовно	15.6 MB	Python
Wing	Безкоштовно або $180/рік	400 MB	Python

Після ретельного аналізу різних інтегрованих середовищ розробки для Python, вирішено обрати PyCharm Professional. Це рішення було зумовлено можливістю безкоштовного використання продуктів Jetbrains для студентів за наявності студентського квитка. PyCharm Professional відзначається широким спектром функцій та інструментів, які сприяють ефективній розробці програмного забезпечення на Python.

2.4 Вибір бібліотеки

Aiogram – це найпопулярніша бібліотека для розробки телеграм-ботів у Python. Вона надає зручний та ефективний інтерфейс для взаємодії з Telegram API, що дозволяє розробникам швидко та легко створювати різноманітні функції для своїх ботів. Однією з ключових переваг aiogram є асинхронність, що дозволяє обробляти багато запитів одночасно та забезпечує високу продуктивність бота. Бібліотека також має багатий функціонал, який включає в себе можливість обробки повідомлень, реакцій на команди користувачів, роботу з клавіатурами, відправку медіафайлів та багато іншого. Документація aiogram є досить повною та зрозумілою. Не менш важливою є наявність активної спільноти та підтримки, яку можна також легко знайти у Telegram. [10]

Переваги бібліотеки Aiogram:

- Виключно асинхронна бібліотека
- FSM – кінцевий автомат або машина станів, за допомогою якої зручно зберігати проміжкові дані.
- Middlewares для обробки даних (апдейтів) перед тим як вони попадуть у функцію.

Недоліки Aiogram:

- Щоб користуватись асинхронною бібліотекою, потрібно знати про відмінність із синхронністю та уникати синхронних функцій у асинхронних.
- Порівняно з іншими може бути складнішим

У контексті aiogram, FSM (Finite State Machine) використовується для створення складних ботів, де кожна дія користувача може залежати від попередніх дій або поточного стану діалогу. FSM дозволяє вам визначати різні стани, через які проходить користувач, і визначати переходи між цими станами залежно від дій користувача. Це особливо корисно для реалізації діалогових сценаріїв, де відповідь користувача може змінювати логіку роботи бота. Запитуючи “Did you like to write bots?” FSM переходить в стан очікування відповіді на це запитання (рисунок 2.4). Відповіддю може бути не тільки введений текст але й натиснута інлайнова кнопка, яку можна надати користувачу. Після відправки тексту або натискання на кнопку, у програмі можна отримати ці дані та оперувати ними. В залежності від відповіді можна вести користувача різними гілками для потрібних цілей.

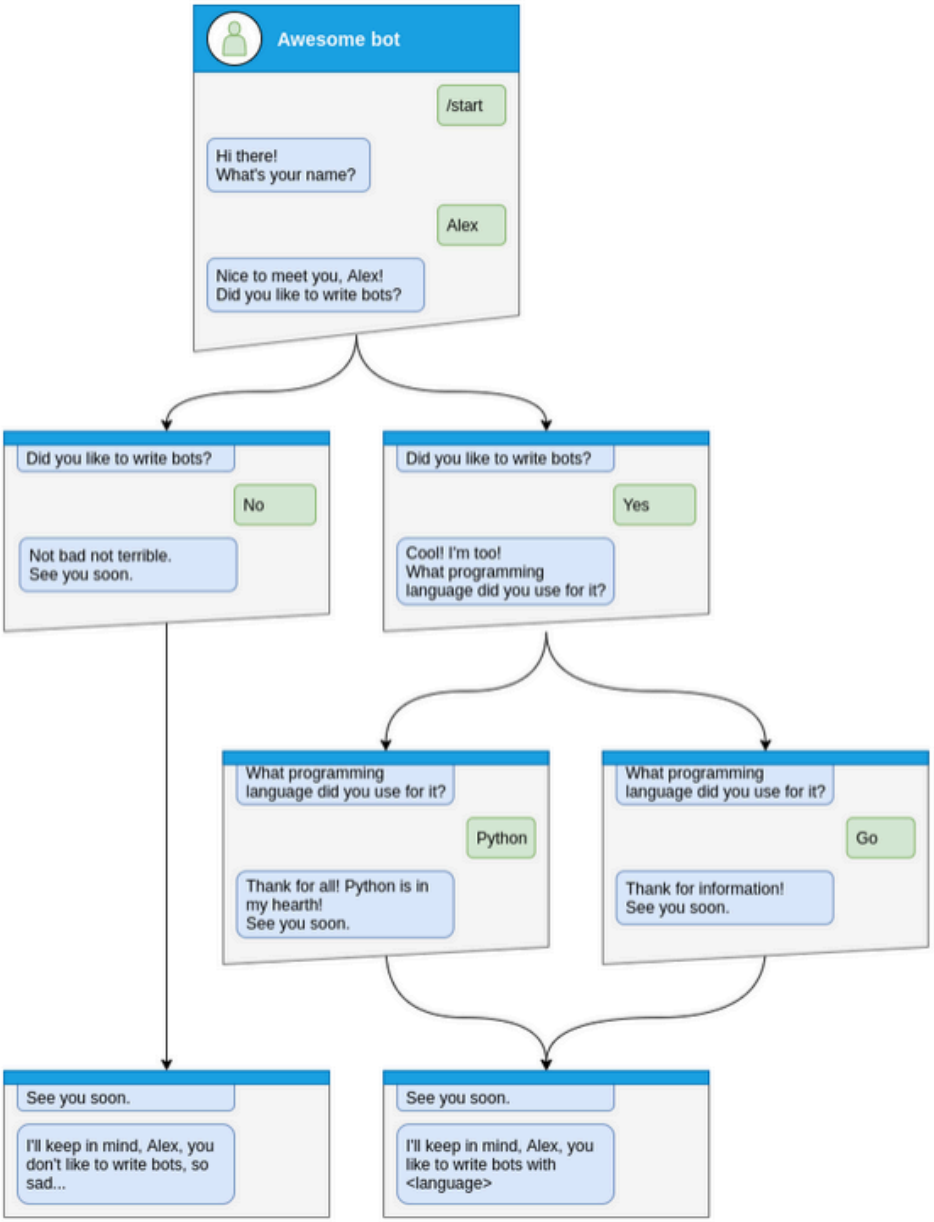


Рисунок 2.4 – Логіка роботи FSM

Існує кілька інших бібліотек для розробки телеграм-ботів у Python такі як pyTelegramBotAPI, Telebot, python-telegram-bot. У кожної є свої переваги та недоліки, тому треба підбирати бібліотеку суто під свої потреби. [11].

Якщо це щось просте на десяток хендлерів і немає можливості довго розбиратися, то у python-telegram-bot дуже докладні wiki, документація та багато прикладів. Можна без досвіду дуже швидко розібратись. Це хороший вибір з малими знаннями програмування.

Переваги python-telegram-bot:

- Інтеграція із зовнішніми сервісами: Бібліотека полегшує інтеграцію робота з іншими сервісами та базами даних.
- Обробка медіа: python-telegram-bot надає якісні засоби обробки зображень, аудіо та інших медіа-файлів.
- Простий та інтуїтивний синтаксис.
- Можливість писати синхронний та асинхронний код
- Одна з найбільших документацій

Недоліки бібліотеки python-telegram-bot:

- Бібліотека не володіє таким ж активним комьюніті як Aiogram, в якого у репозиторії є посилання на спілки по країнам.
- Вразлива до масштабних навантажень

Telethon відмінно підходить, коли вам потрібно підключити Python до облікового запису в Telegram і автоматизувати його. За допомогою цієї бібліотеки Python-скрипт може керувати аккаунтом у Telegram, надаючи вам можливість взаємодіяти з Telegram як з окремою сутністю, а не лише через ботів. [12]

Цей підхід не підходить для всіх проектів та не підходить для всіх ситуацій. Він може бути корисним, наприклад, якщо ви хочете почати спілкування з користувачами першими, оскільки боти не можуть цього зробити. Проте, ризик отримати блокування через спам у такому випадку вище, ніж при роботі з ботом.

Переваги Telethon:

- Можливість вивантажувати файли більше 40 Мб
- Спілкується з серверами телеграму не через HTTP протокол а MTProto
- Програмно керувати телеграм акаунтом.

Недоліки Telethon:

- Немає можливості створити телеграм бот не прив'язаний до акаунту.
- При розсилці повідомлень можна отримати блокування за спам.

MTProto (Mobile Telegram Protocol) – це власний протокол обміну повідомленнями, розроблений компанією Telegram для забезпечення швидкої та безпечної передачі даних між клієнтами та серверами Telegram. Повідомлення шифруються за допомогою 256-бітного симетричного шифрування AES, 2048-бітного RSA, а також обміну ключами за допомогою алгоритму Диффі-Хеллмана. Хоча специфікація протоколу MTProto не є повністю відкритою, Telegram надає деяку інформацію та документацію для розробників, яка дозволяє розробляти власні клієнти та інтеграції з Telegram (рисунок 2.5). [13].

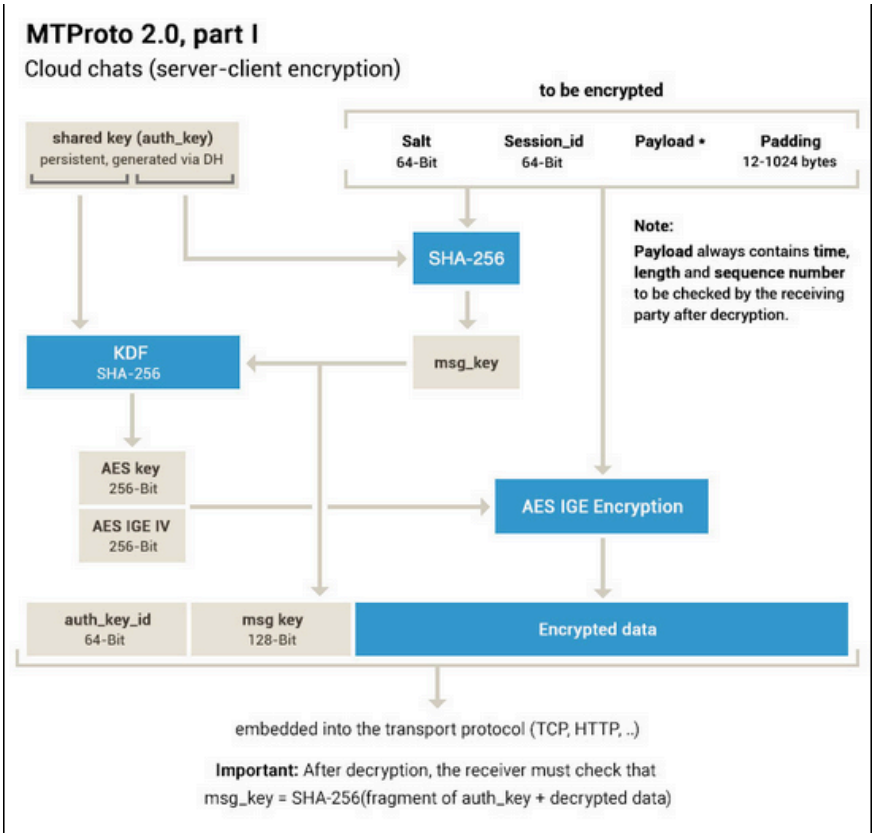


Рисунок 2.5 – Шифрування сервер клієнт

HTTP (Hypertext Transfer Protocol) – це протокол передачі гіпертекстових даних, який використовується для взаємодії між клієнтами та серверами у Всесвітній павутині. AIogram використовує HTTP-запити для взаємодії з серверами Telegram.

Отже, AIogram взаємодіє з серверами Telegram не за допомогою MTProto протоколу а за допомогою HTTP. Наприклад, надсилаючи повідомлення через бота, AIogram створює HTTP-запит до серверів Telegram з необхідними даними, такими як ідентифікатор чату та текст повідомлення. Після цього він очікує відповіді від сервера Telegram і обробляє її для відповідного подальшого виконання.

Цей підхід дозволяє ботам, створеним за допомогою Aiogram, ефективно спілкуватися з користувачами та обробляти їх запити, забезпечуючи при цьому високу продуктивність та швидкість реакції.

Також для планування постів потрібно використати планувальник. APScheduler є бібліотекою для Python, яка дозволяє планувати виконання функцій у певні моменти часу або з певною періодичністю. З її допомогою можна створювати розклади виконання задач, автоматизувати процеси в коді. Вона підтримує різні типи тригерів, включаючи виконання задачі у певний момент часу (date), з певним інтервалом (interval) або на підставі певного розкладу. APScheduler також має можливість зберігати розклад у різних механізмах, таких як у пам'яті, у базі даних або на диску. Вона також дозволяє реалізувати додаткову логіку перед виконанням, після виконання або при помилці задачі. З версії 3.x APScheduler підтримує асинхронні задачі.

2.5 Вибір бази даних

У проекті телеграм бота, база даних є критично важливою для організації, зберігання та доступу до інформації. Зокрема, вона забезпечує можливість зберігання даних про користувачів, канали, медіафайли та каталоги. Наявність

централізованої бази даних дозволяє ефективно управляти цими даними, забезпечуючи швидкий доступ і обробку запитів.

Для зберігання даних було обрано MySQL. Цей вибір обумовлений кількома причинами. MySQL відома своєю ефективністю у роботі з великими обсягами даних і підтримкою складних запитів, що є важливим для нашого проекту. Крім того, MySQL легко інтегрується з мовою програмування Python, на якій написаний наш бот, що спрощує розробку і підтримку коду. Якщо порівнювати MySQL з популярним аналогом PostgreSQL, то PostgreSQL має більше можливостей і швидший для запису та оновлення даних. Однак, у нашому випадку більшість операцій будуть спрямовані на читання даних, де MySQL демонструє високу швидкість за рахунок паралелізму [19].

Крім основної бази даних MySQL, для зберігання планувань було обрано Redis. Redis – це високопродуктивне ключ-значення сховище, яке дозволяє швидко зберігати та отримувати дані. Всі дані, збережені у Redis, зберігаються в оперативній пам'яті, що забезпечує дуже швидкий доступ до даних. Завдяки цьому Redis часто використовується для задач, де швидкість доступу є критично важливою, таких як кешування, управління сесіями та обробка потоків даних в реальному часі. Використання Redis у якості сховища планувань дозволяє забезпечити швидкий доступ до даних про заплановані пости, що є критично важливим для бота, оскільки затримки у виконанні планувань можуть призвести до невчасної публікації контенту. Redis також підтримує функціонал автоматичного видалення ключів після заданого часу, що дозволяє ефективно управляти тимчасовими даними [20].

Таким чином, комбінація MySQL для основних даних та Redis для планувань забезпечує надійну, швидку та ефективну систему зберігання даних, що відповідає всім вимогам телеграм бота.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТЕЛЕГРАМ БОТУ

3.1. Проектування логіки взаємодії з користувачем

Розробка блок-схеми взаємодії користувача з ботом допоможе візуалізувати всі можливі сценарії та шляхи взаємодії, визначити ключові етапи та функції. Використання інструментів, таких як Miro, дозволить створити наочні блок-схеми для полегшення розуміння процесів. Таким чином в подальшому значно спрощується процес розробки, так як тримати всю структуру в голові досить складно.

3.1.1 Схема процесу авторизації

За вимогами потрібно зробити телеграм бот приватним, тому треба створити механізм авторизації. Після того, як бот отримує update (оновлення), має відбуватись перевірка, чи є такий користувач у базі, якщо у користувача є доступ, він отримає головне меню, інакше буде повідомлений про те, що у нього немає доступу (рисунок 3.1).

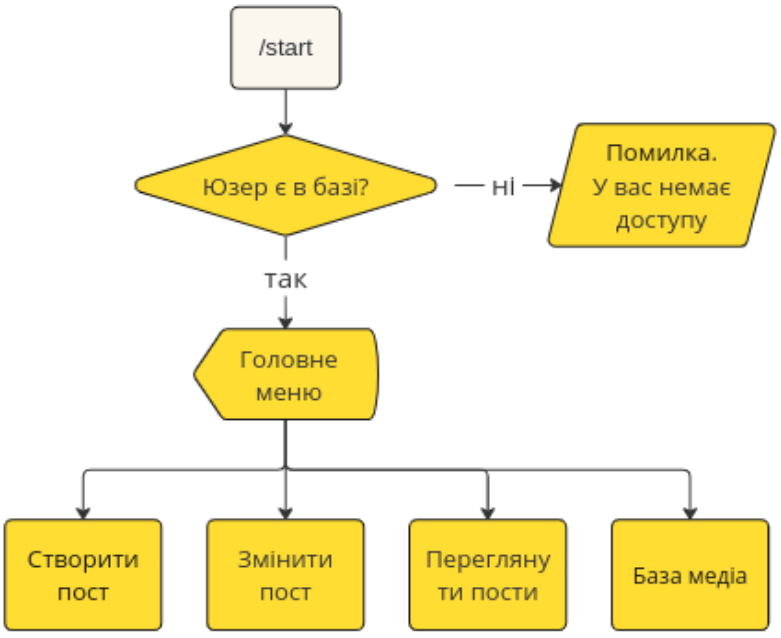


Рисунок 3.1 – Блок-схема авторизації

3.1.2 Схема процесу створення поста

Одна із найбільш функціональних частин телеграм бота – це механізм створення поста. Користувачу потрібно надати велику кількість можливостей але при цьому зберегти простий та зрозумілий інтерфейс. У блок-схемі продемонстровано частину механізму створення поста (рисунок 3.2). Щодо циклічних постів, то користувач може вказати, на скільки хвилин затримати пост, а також скільки днів між публікаціями потрібно пропустити, а якщо говорити про заплановані пости, тоді потрібно просто обрати дату та час публікації (рисунок 3.2).

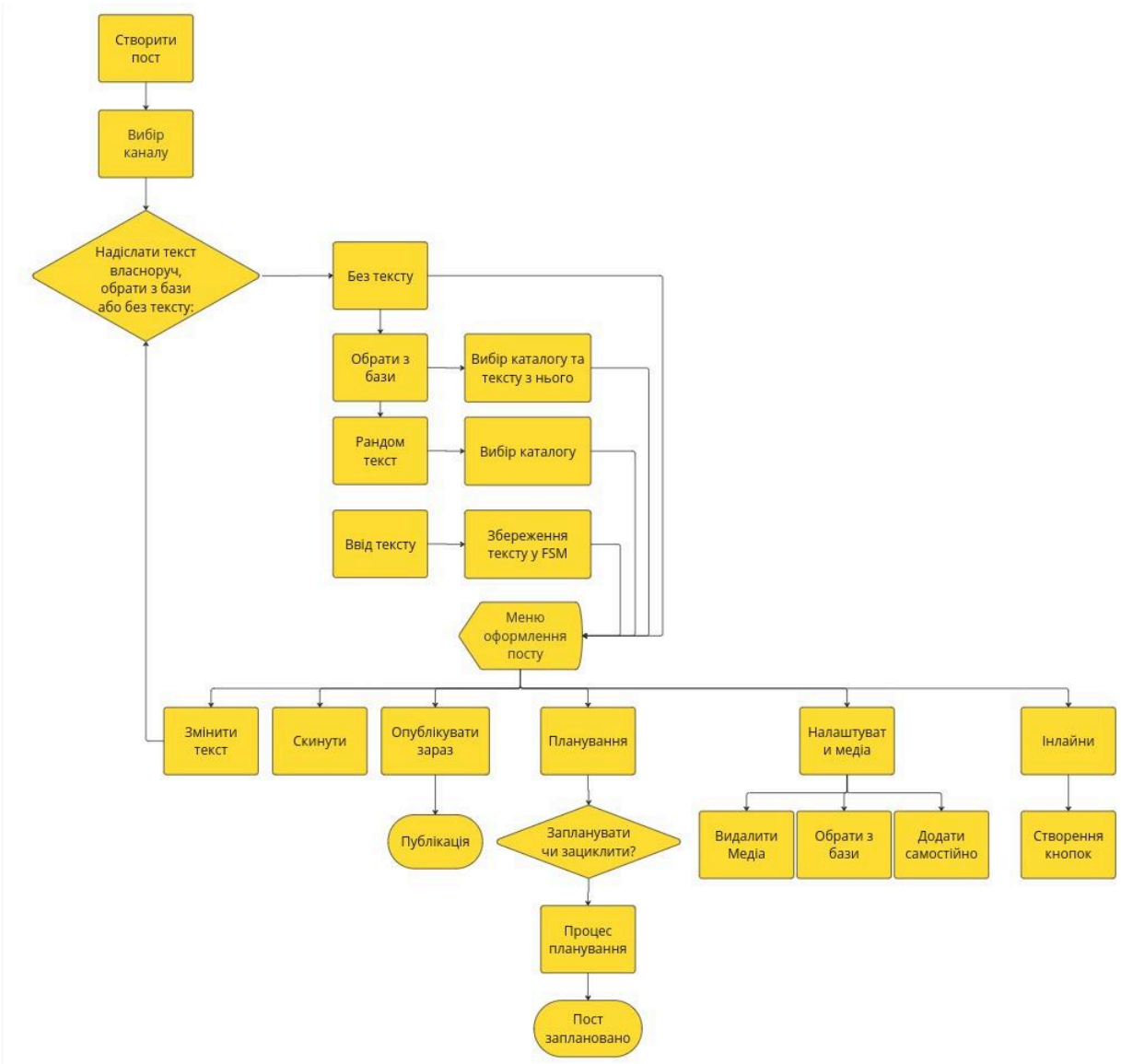


Рисунок 3.2 – Блок-схема створення поста

3.1.3 Схема бази медіа бота

У телеграм каналах часто виникає потреба регулярно випускати публікації з одними й тими ж медіафайлами. Наприклад, для кожного дня тижня може бути призначена своя унікальна картинка або відео. Це особливо актуально для каналів, які публікують тематичні пости, нагадування, розклад заходів або регулярні рубрики. Власна база медіа дозволяє зберігати всі необхідні медіафайли (фотографії, відео, документи, аудіозаписи, відеоповідомлення) безпосередньо на сервері бота, а не на пристрої користувача. Таке рішення допомагає уникнути ще однієї можливої проблеми, а саме прив'язку до пристрою. Тобто, доступ до медіа буде не в власника пристрою, а у всіх у кого є доступ до керування певного каналу (рисунок 3.3).

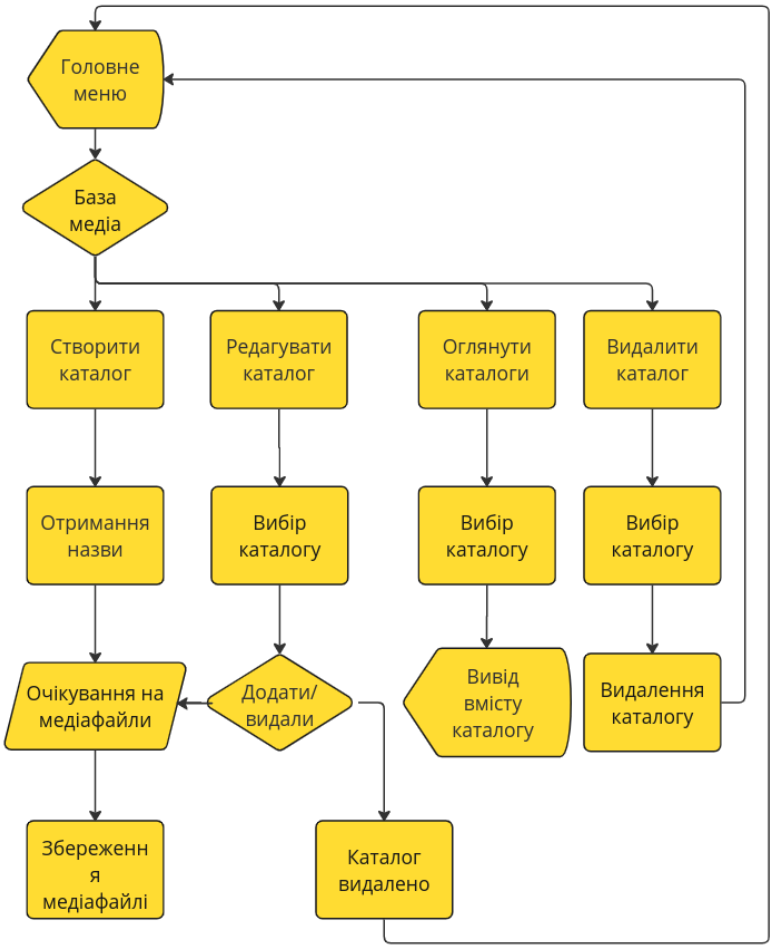


Рисунок 3.3 – Блок-схема бази медіа

3.1.4 Проектування сторони адміністратора

Так як телеграм бот має бути приватним і надавати доступ потрібно зі сторони користувача а не сервера, було прийнято рішення створити сторону адміністратора. Щоб перейти в меню адміністратора, потрібно ввести команду /moderator, після чого можна буде надати доступ користувачу або скасувати. Також можна переглянути список користувачів, у яких є доступ (рисунок 3.4).

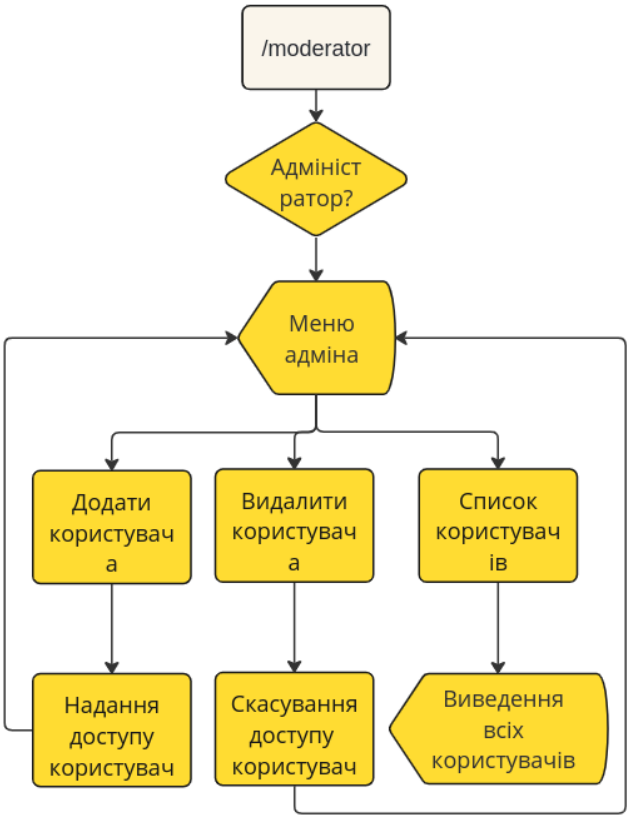


Рисунок 3.4 – Блок-схема сторони адміністратора

Отже, створена структура взаємодії користувача з телеграм ботом значно спростить процес створення бота як програмного забезпечення. А також блок-схеми є важливим допоміжним інструментом для розуміння логіки роботи бота та визначення його функціональності.

3.2 Створення бази даних та таблиць

Основне навантаження бота буде на операції читання даних, а не запису чи оновлення. MySQL, завдяки своїй архітектурі та оптимізації для паралельного виконання запитів, часто виявляється швидшим аналогів у таких сценаріях. Це дозволяє забезпечити високу швидкість відповіді на запити користувачів, що є критично важливим для зручності та ефективності роботи бота.

У даному випадку було прийнято рішення створити 4 таблиці (рисунок 3.5):

1. Catalog – таблиця каталогів у яких будуть зберігатись медіа
2. Channel – таблиця для каналів, якими керують користувачі
3. Media – у цій таблиці будуть зберігатись file_id кожного медіафайлу, який зберігається в каталозі.
4. User – зберігатимуться користувачі телеграм бота.

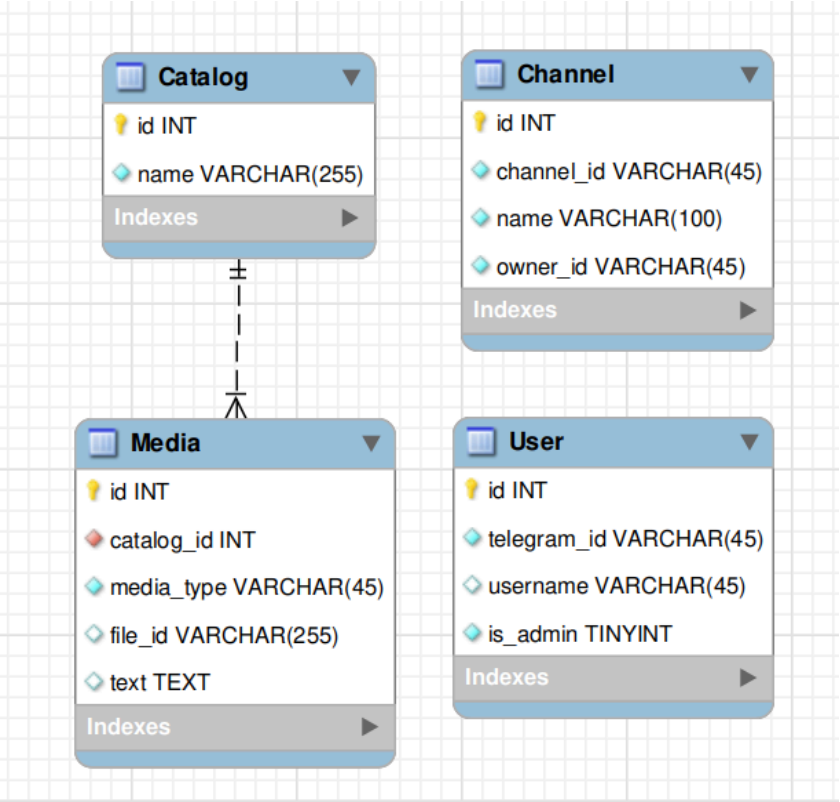


Рисунок 3.5 – Схема таблиць телеграм бота бази даних

3.3 Створення структури проекту

Для створення Telegram бота, спочатку необхідно отримати токен у BotFather. BotFather – це офіційний бот від Telegram, який допомагає створювати нових ботів і керувати ними. Після запуску BotFather у Telegram, слід використати команду /newbot, яка ініціює процес створення нового бота. BotFather попросить вас надати ім'я та унікальне користувацьке ім'я для вашого бота, після чого згенерує токен доступу (рисунок 3.6). Цей токен є важливим, оскільки він дозволяє вашому боту взаємодіяти з Telegram API.

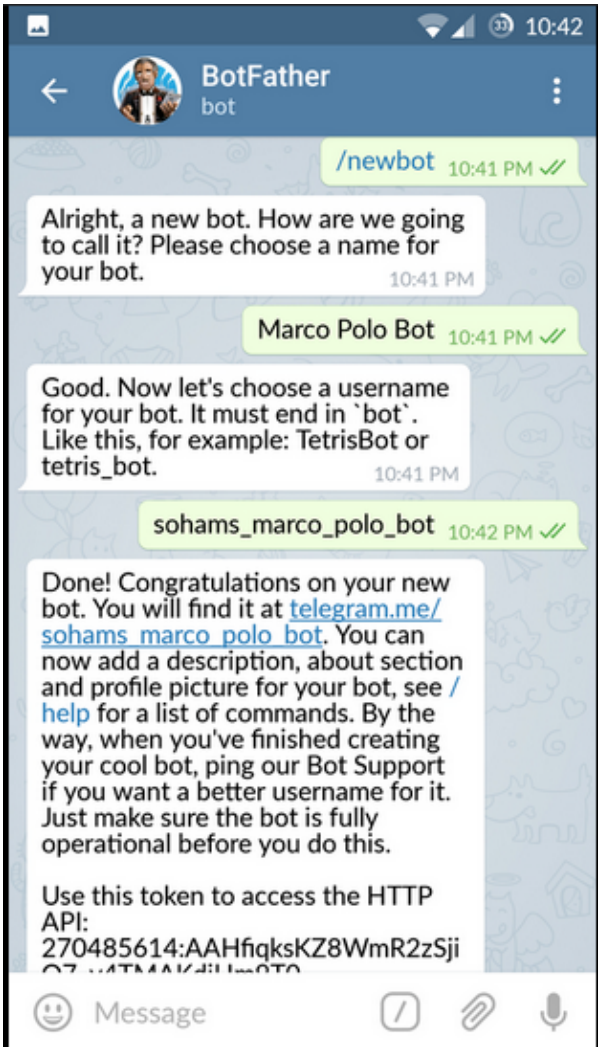


Рисунок 3.6 – Отримання токену в BotFather

Після отримання токену потрібно розпочати структурування проекту. Структура проекту для бота буде складатись з модуля створення бота, модуля

запуску, окремі модулі для обробки команд і повідомлень, модулі кнопок, утиліти а також конфігураційний файл для збереження налаштувань і токену. Така організація дозволяє легко керувати кодом, розширювати функціональність бота та підтримувати його в майбутньому.

Таблиця – 3.1 Призначення Python модулів проекту

Назва модуля	**Призначення**
handlers/admin.py	Хендлери для взаємодії зі стороною адміністрітора; Кінцевий автомат FSMAdmin для адмін-сторони; Функція реєстрації хендлерів;
handlers/catalog_handlers.py	Обробники повідомлень які стосуються керування каталогами медіа; Функція реєстрації хендлерів;
handlers/client.py	Кінцевий автомат FSMClient для клієнтської сторони; Хендлери для обробки запитів клієнта; Функція реєстрації хендлерів;
handlers/plan_loop_handlers.py	Хендлери, які відповідають за планування постів; Функція реєстрації хендлерів;
keyboards/kb_admin.py	Створення кнопок для меню сторони адміністратора;
keyboards/kb_client.py	Створення кнопок для меню сторони клієнта; Функції створення пагінації;
migrations/env.py	Налаштування для підключення до бази даних та інструкції для Alembic щодо того, які таблиці та схеми мають бути мігрувані.

config.py	Отримання токену бота, паролю бази даних, назву, хост та користувача з файлу .env
create_bot.py	Підключення Redis; Створення scheduler; Реєстрація бота та диспетчера;
db_manage.py	Підключення до MySQL; Створення моделей таблиць; Функції взаємодії з базою даних;
start_bot.py	Реєстрація всіх хендлерів; Підключення логування; Запуск полінгу;
utils/utils.py	Допоміжні функції, утиліти;

3.4 Програмна реалізація

Перед початком роботи необхідно налаштувати середовище розробки і встановити необхідні бібліотеки та інструменти (рисунок 3.7). Віртуальна оболонка буде використовуватись venv. Після активації віртуального середовища можна встановити бібліотеки.

Віртуальне середовище – це ізольоване середовище, в якому можна встановлювати та використовувати різні версії пакетів Python для різних проектів. Воно дозволяє уникнути конфліктів між версіями пакетів та забезпечує чистоту управління залежностями у вашому програмному проекті.

```
python3 -m venv venv
source venv/bin/activate  # Або venv\Scripts\activate на Windows
pip install aiogram
pip install sqlalchemy
pip install alembic
```

Рисунок 3.7 – Створення віртуального середовища та встановлення бібліотек

Для того щоб створити перший хендлер, який буде реагувати на команду /start необхідно написати асинхронну функцію, яка приймає 1 обов'язковий аргумент callback. У цьому випадку буде ще прийматись аргумент state, у якому містяться стани та збережені дані з кінцевого автомата FSM (рисунок 3.8).

```
async def start_command(message: Message, state: FSMContext):
    await state.finish()
    if message.chat.type != types.ChatType.GROUP:
        await message.answer(text=f'Вітаю, {message.from_user.username}\n'
                                  f'Це бот для відкладеного постингу. Для початку роботи '
                                  f'скористайтеся командами або головним меню.\n'
                                  f'/addchannel - підключення нового каналу',
                             reply_markup=main_kb)
```

Рисунок 3.8 – Обробник команди /start

await state.finish() виконується для того, щоб при виконанні цієї функції, всі попередні дані та стани очікування обнулилися.

Функція має знати, що вона має відповідати саме на команду /start, тому потрібно прив'язати команду до функції. Щоб це зробити, потрібно створити функцію у якій будуть прив'язуватись до команд всі обробник, тобто реєструватись (рисунок 3.9). Ця функція буде створена у цьому ж модулі handlers/client.py а виконуватись один раз при запуску у модулі start_bot.py.

```python
import logging

from aiogram.utils import executor

from create_bot import dp
from handlers import client, admin, plan_loop_handlers, catalog_handlers

admin.register_handlers_admin(dp)
client.register_handlers_client(dp)
plan_loop_handlers.register_handlers_schedule(dp)
catalog_handlers.register_handlers_catalog(dp)


logging.basicConfig(level=logging.INFO)
executor.start_polling(dp, skip_updates=True)
```

Рисунок 3.9 – Реєстрація хендлерів у start_bot.py

Виконавши команду /start користувач отримає доступ до головного меню, якщо у нього є доступ до користування ботом (рисунок 3.10).

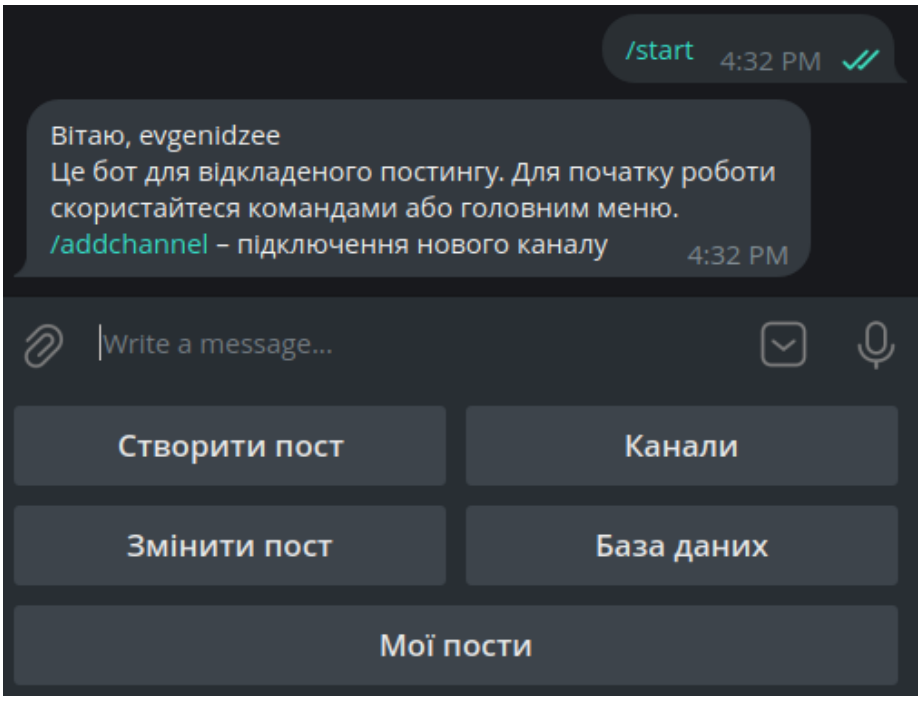


Рисунок 3.10 – Головне меню паблішера

Доступ користувача перевіряється за допомогою класу AuthMiddleware (рисунок 3.11). Клас AuthMiddleware є middleware для телеграм бота, написаного за допомогою бібліотеки aiogram. Його основна функція полягає в

перевірці, чи має користувач доступ до бота, перед обробкою кожного запиту. Якщо користувач не має доступу, запит скасовується, і користувач отримує відповідне повідомлення.

```
class AuthMiddleware(BaseMiddleware):
    async def on_pre_process_update(self, update: types.Update, data: dict):
        if not isinstance(update.message, type(None)):
            if update.callback_query:
                user = await get_user(update.callback_query.from_user.id)
                if not user:
                    await update.callback_query.message.answer(
                        text='У вас немає доступу до бота. Після отримання дос
                        reply_markup=ReplyKeyboardRemove())
                    raise CancelHandler()
            else:
                user = await get_user(update.message.from_user.id)
                if not user:
                    await update.message.answer(
                        text='У вас немає доступу до бота. Після отримання дос
                        reply_markup=ReplyKeyboardRemove())
                    raise CancelHandler()
```

Рисунок 3.11 – Middleware для авторизації

Ця функція виконується перед обробкою кожного оновлення (update), яке надходить до бота. Вона виконує перевірку наявності доступу користувача до бота.

Після натискання кнопки “Створити пост” у меню, з’являться нові повідомлення. Спочатку бот запросить обрати з бази або ввести текси, також можна створити пост без тексту. Далі, якщо обрати опцію “Додати медіа”, то бот буде очікувати на медіафайли (рисунок 3.12).

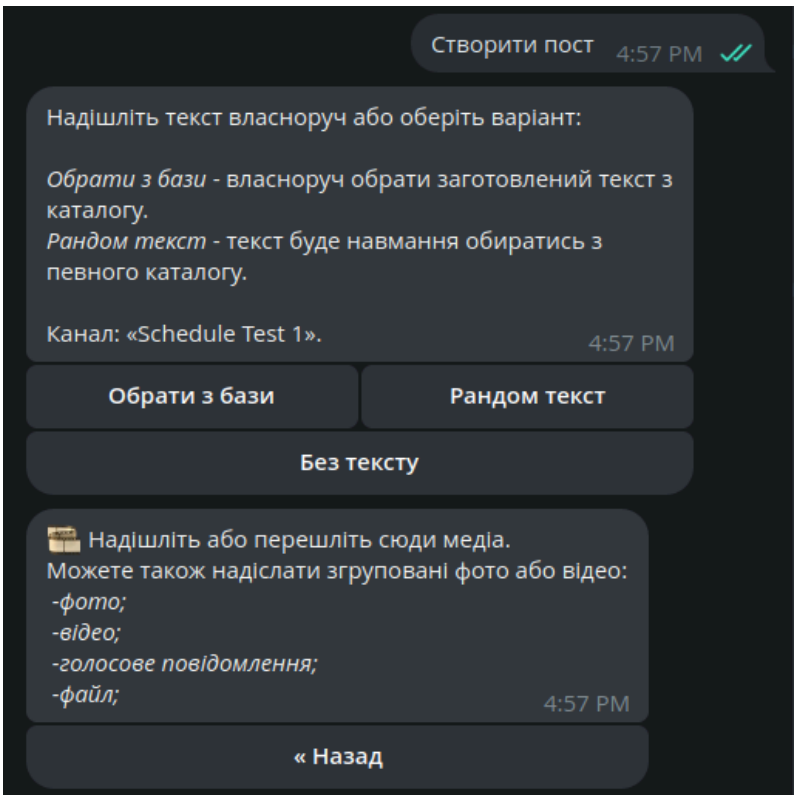


Рисунок 3.12 – Запит медіа та тексту

На цьому етапі виникає перша проблема. Часто в одному пості потрібно прикріпити більше ніж одне фото або відео. Це називається медіагрупа (також альбом). Якщо спробувати надіслати звичайному хендлеру aiogram медіагрупу, він відпрацює на кожне вкладення. Тобто при відправці трьох картинок однією групою, функція відпрацює тричі, сприймаючи кожну фотографію як окреме повідомлення. Через це у ботів-аналогів можна надсилати фотографії лише по черзі.

Для того, щоб можна було надіслати медіагрупу потрібно, щоб функція сприйняла вкладення один раз, помістивши всі медіа у масив. Український розробник під ніком deptyped у github [17] опублікував рішення цієї проблеми. Це хендлер media_group_handler, який може сприймати медіагрупи і збирати всі медіафайли в один масив (рисунок 3.13). Цю фунцію можна використати для свого хендлера як декоратор, тоді цей хендлер також зможе приймати декілька вкладень одразу в одному масиві.

```python
@media_group_handler
async def load_media_file(messages: List[types.Message], state: FSMContext):
    if await pressed_back_button(messages[0]):
        await state.reset_state(with_data=False)
        await choose_or_self_media(call=messages[0], state=state)
        return
    else:
        data = await state.get_data()
        job_id = data.get('job_id')
        if job_id:
            job = scheduler.get_job(job_id)
            data = scheduler.get_job(job_id).kwargs.get('data')
```

Рисунок 3.13 – Застосування декоратора media_group_handler

Просто використати це рішення було недостатньо, так як у разі надсилання 1 медіафайлу, тобто не альбому, функція видавала помилку ValueError("Not a media group message") рисунок(3.14). Тому треба трішки змінити її і надсилати у такому разі масив з одного елементу (рисунок 3.15)

```python
def decorator(handler):
    @wraps(handler)
    async def wrapper(message: types.Message, *args, **kwargs):

        if only_album and message.media_group_id is None:
            raise ValueError("Not a media group message")
        elif message.media_group_id is None:
            return await handler([message], *args, **kwargs)
```

Рисунок 3.14 – Варіант deptyped

```python
def decorator(handler):
    @wraps(handler)
    async def wrapper(message: types.Message, *args, **kwargs):
        if isinstance(message, types.Message):
            if only_album and message.media_group_id is None:
                return await handler([message], *args, **kwargs)
            elif message.media_group_id is None:
                return await handler([message], *args, **kwargs)
        else:
            return await handler([message], *args, **kwargs)
```

Рисунок 3.15 – Виправлений варіант

Після часткового редагування декоратора бот має змогу приймати як один медіафайл, так і групу медіафайлів (рисунок 3.16).

Рисунок 3.16 – Бот приймає зберігає одразу альбом

Процес планування відбувається за допомогою планувальника APScheduler та Redis (рисунок 3.17). Потрібно створити сам scheduler та вказати, що дані потрібно зберігати у Redis.

```
job_stores = {
    "default": RedisJobStore(
        jobs_key="dispatched_trips_jobs", run_times_key="dispatched_trips_running",
        host="localhost", port=6379
    )
}

scheduler = ContextSchedulerDecorator(AsyncIOScheduler(jobstores=job_stores))
scheduler.start()
```

Рисунок 3.17 – Створення планувальника

Коли користувач обрав тип планування зациклення, на back-end стороні буде створюватись job за допомогою scheduler.add_job() (рисунок 3.18). У метод add_job потрібно передати параметр func, trigger, days, start_date, kwargs{‘data’: data}. Параметр func має приймати функцію яка буде виконуватись в запланований час, цю функцію потрібно створити власноруч.

```
scheduler.add_job(send_message_cron,
                  trigger='interval',
                  days=int(days_skip) + 1,
                  start_date=str(new_date),
                  kwargs={'data': data})
```

Рисунок 3.18 – Створення планування

Функція send_message_cron – це функція-тригер, яку буде викликати scheduler у запланований час (рисунок 3.19). У цій функції буде створюватись та публікуватись пост на основі тих даних, які були передані scheduler при плануванні.

```python
async def send_message_cron(data):
    kb_inline = data.get('inline_kb')
    skip_minutes_loop = data.get('skip_minutes_loop')
    post_text = await create_text(data)
    kb = await create_kb(kb_inline)
    media_files = await create_random_media(data)

    if data.get('video_note'):
        post_video_note = data.get('video_note')
    else:
        post_video_note = data.get('random_v_notes_id')
        if post_video_note:
            post_video_note = post_video_note[0]

    if skip_minutes_loop:
        if skip_minutes_loop == '0':
            random_number = 0
        else:
            random_minutes = data.get('skip_minutes_loop').split('-')
            from_minute = int(random_minutes[0])
            to_minute = int(random_minutes[1])
            random_number = random.randint(from_minute, to_minute)
    else:
        random_number = 4
    print(f"post in {random_number} minutes")
    await asyncio.sleep(random_number * 60)
    await send_post_to_channel(post_media_files=media_files, post_text=post_text, bot_instance=bot,
                               channel_id=data.get('channel_id'), post_voice=data.get('voice'),
                               post_video_note=post_video_note,
                               inline_kb=kb, data=data)
```

Рисунок 3.19 – Функція send_message_cron

Функція “Змінити пост” є однією з найбільш важливих та часто використовуваних можливостей телеграм бота. Коли користувач хоче змінити пост, він може зробити це кількома способами. По-перше, можна редагувати текст повідомлення. По-друге, користувач може змінити медіа-файли, що додаються до посту. Також є можливість налаштувати або змінити інлайн-кнопки у існуючому пості. Це дозволяє змінити посилання на ресурси, додати нові кнопки або видалити непотрібні.

Окрім цього, функція “Змінити пост” дозволяє редагувати планування публікації, це включає зміну дати та часу виходу посту, зміну проміжків часу між публікаціями, а також затримки перед постингом публікації.

Після натискання “Змінити пост” потрібно обрати канал, після цього потрібно перевірити, чи є у Redis записи в id яких міститься id телеграм каналу (рисунок 3.20):

```python
async def edit_post_list(message: types.CallbackQuery, state: FSMContext):
    data = await state.get_data()
    ...
    jobs = scheduler.get_jobs()
    edit_kb = InlineKeyboardMarkup()
    posts = []
    jobs = sorted(jobs, key=lambda job: job.next_run_time)
    for job in jobs:
        job_data = job.kwargs.get('data')
        if job_data.get('channel_id') == channel_id:
            posts.append(job)
...
    posts = list(post_chunks[page_num - 1])
    if posts:
        await add_posts_to_kb(jobs=posts, edit_kb=edit_kb)
        edit_kb.add(back_edit_post_inline)
        await paginate(edit_kb)
        edit_kb.inline_keyboard[-1][-2].text = page_num
        try:
            await message.answer()
            await message.message.edit_text( text: f'Ваші заплановані та зациклені пости.\n'
                                             'Оберіть потрібний вам:', reply_markup=edit_kb)
        except Exception as err:
            logging.info(f'ERROR: {err}; {edit_kb}')
        await FSMClient.job_id.set()
    else:
        await message.answer()
        try:
            await message.message.edit_text( text: 'У вас немає запланованих або зациклених постів.',
                                             reply_markup=create_post_inline_kb)
```

Рисунок 3.20 – Хендлер відображення списку постів конкретного каналу

Якщо у каналі не буде запланованих постів, користувач отримає про це повідомлення (рисунок 3.20).

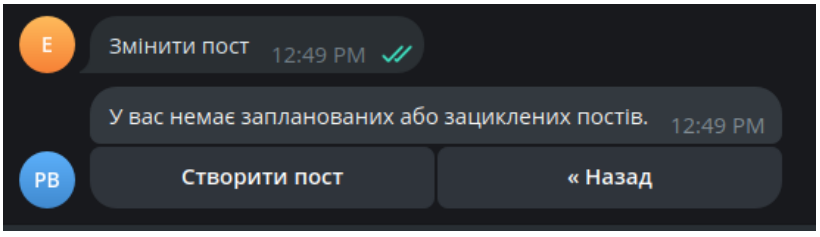


Рисунок 3.21 – Меню при відсутності постів у каналі

У разі, якщо пости присутні для обраного каналу, користувач отримає їх список. Пости, які заплановані у режимі циклу позначаються emoji 🌀 (рисунок 3.22):

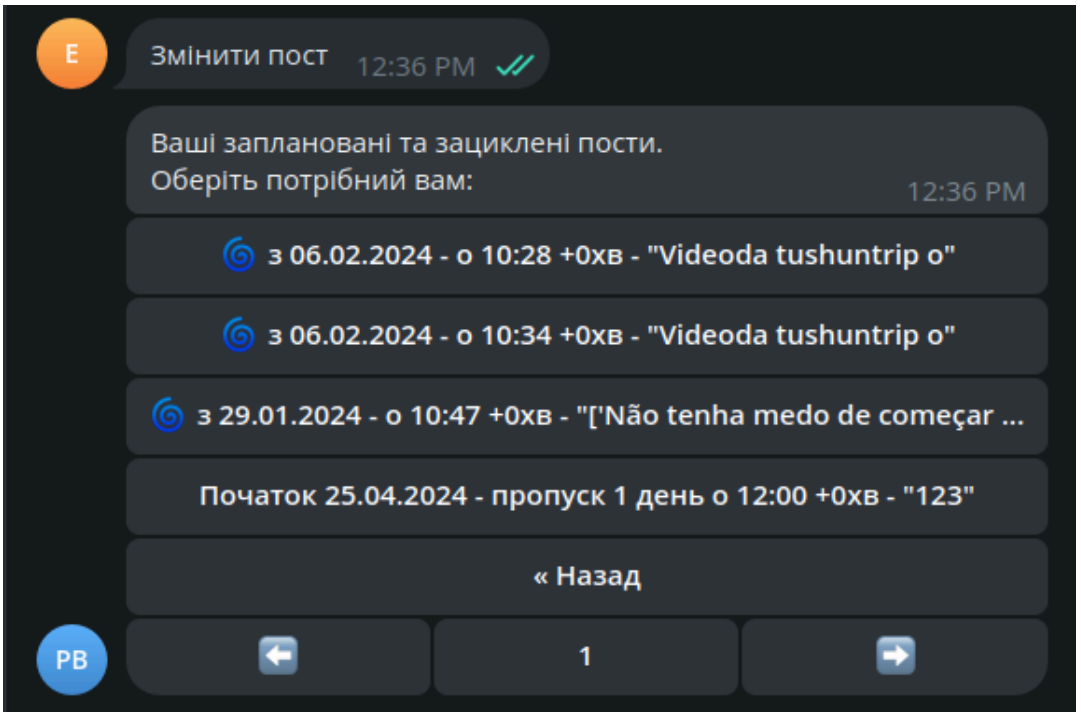


Рисунок 3.22 – Список постів каналу

Після натискання на пост, з’явиться меню схоже на те, що й при створенні посту, але додається нова кнопка “❌ Видалити пост” (рисунок 3.23):

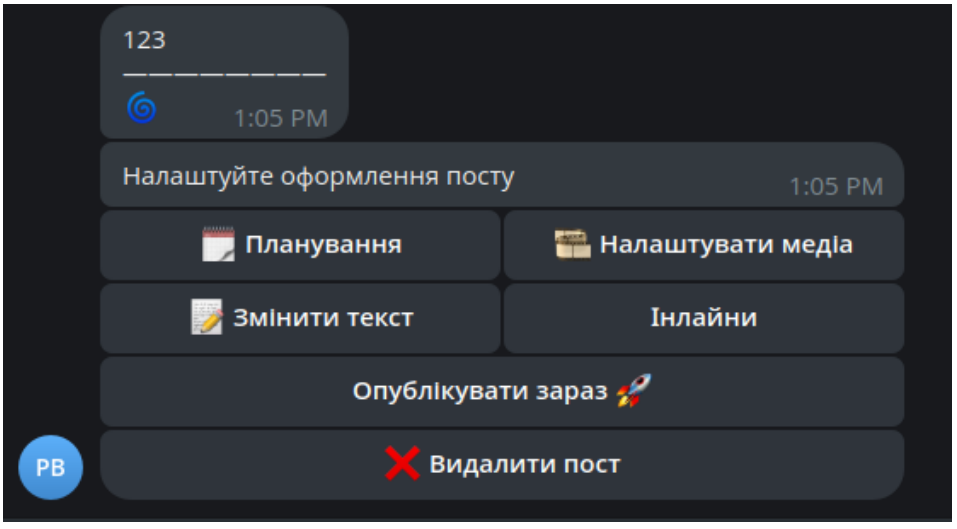


Рисунок 3.23 – Меню редагування існуючого поста

При створенні та зміні посту медіафайли можна не тільки завантажувати з локального пристрою, але й з бази медіафайлів. Натиснувши в головному мненю на кнопку “База даних” користувач отримає панель управління базою медіа (рисунок 3.24):

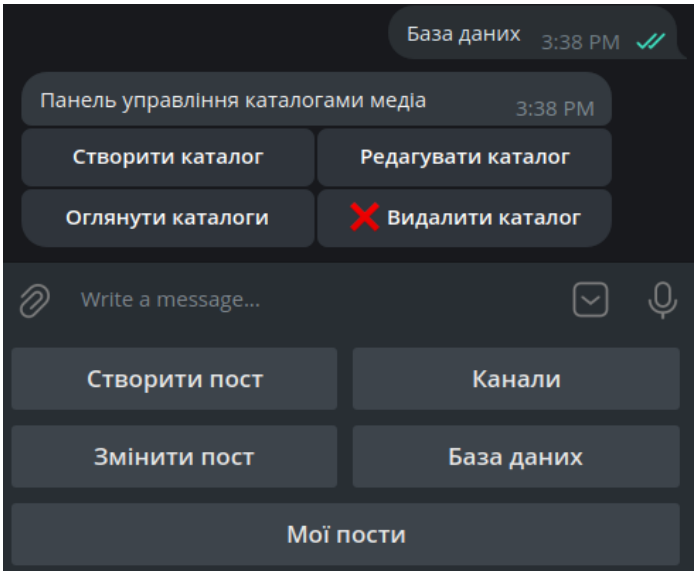


Рисунок 3.24 – Панель управління базою медіа

Усі каталоги будуть зберігатись у базі даних MySQL. Для зручної взаємодії з таблицями у коді, їх можна представити у вигляді класів які називаються моделями (рисунок 3.25).

```
10 usages
class Catalog(Base):
    __tablename__ = 'Catalog'
    id: Mapped[int] = mapped_column(primary_key=True, nullable=False, autoincrement=True)
    name: Mapped[str] = mapped_column(type_=String(255), nullable=False, unique=False)

    def __repr__(self):
        return f'<Catalog {self.name}>'
```

Рисунок 3.25 – Модель таблиці Catalog

Після того, як користувач натиснув на “Створити каталог” та ввів назву нового каталогу, виконується хендлер load_cat_name у якому викликається функція save_cat (рисунок 3.26). save_cat безпосередньо робить запит на створення нового запису у таблиці Catalog (рисунок 3.27).

```python
async def load_cat_name(message, state: FSMContext):
    if await pressed_back_button(message=message):
        await state.reset_state(with_data=False)
        await media_base_panel(message=message, state=state)
        return
    if await cat_name_exist(cat_name=message.text):
        await message.answer(text='Такий каталог вже існує.\n'
                                  'Введіть іншу назву:')
    else:
        await save_cat(message.text)
        await state.update_data(cat_name=message.text)
        await message.answer(f'Каталог "{message.text}" створено.\n'
                             f'📁 Надішліть або перешліть сюди медіа.\n'
                             'Можете також надіслати згруповані фото або відео:\n'
                             '\t<i>-фото;</i>\n'
                             '\t<i>-відео;</i>\n'
                             '\t<i>-голосове повідомлення;</i>\n'
                             '\t<i>-файл;</i>\n'
                             '\t<i>-текст</i>', parse_mode='html', reply_markup=back_kb)
        await state.reset_state(with_data=False)
        from handlers.client import FSMClient
        await FSMClient.loaded_catalog_file.set()
```

Рисунок 3.26 – Хендлер, який спрацьовує після введення назви каталогу

```python
async def save_cat(cat_name):
    async with async_session() as session:
        query = insert(Catalog).values(name=cat_name)
        await session.execute(query)
        await session.commit()
```

Рисунок 3.27 – Функція створення нового запису у таблиці Catalog

Отже, третій розділ присвячений безпосередній реалізації телеграм бота. Розпочато з проектування логіки взаємодії з користувачем, де детально розглянуто процеси авторизації та створення постів. Блок-схеми процесів допомагають краще зрозуміти послідовність дій та взаємодію компонентів. Створення бази медіа та адміністрування каналів забезпечують зручність та ефективність роботи бота. Використання MySQL для зберігання структурованих даних та Redis для кешування та зберігання планувань дозволяє оптимізувати продуктивність системи. Цей розділ закладає основу для подальшої програмної реалізації бота, забезпечуючи чітку структуру та логіку роботи.

ВИСНОВКИ

У рамках бакалаврської дипломної роботи було проведено глибоке, всебічне та комплексне дослідження предметної області, що стосується розробки телеграм ботів. Початковий етап роботи передбачав аналіз не лише актуальності цієї теми в сучасному світі, а й поглиблений огляд існуючих рішень та їхнього впливу на різні сфери діяльності. В результаті дослідження було встановлено, що телеграм боти стають все більш важливим інструментом у багатьох галузях, включаючи бізнес, освіту, медицину, маркетинг та інші, завдяки їхній унікальній здатності автоматизувати процеси комунікації та надавати широкий спектр послуг.

Після визначення актуальності теми та загального огляду існуючих рішень у сфері розробки телеграм ботів, наступним кроком був детальний аналіз методів розв'язання завдання створення такого бота. Цей аналіз включав в себе розгляд різних підходів та стратегій, які можуть бути використані для розробки бота, зокрема, архітектурних шаблонів, методів взаємодії з користувачем, а також огляд технічних аспектів, таких як зберігання та обробка даних.

Цей етап також передбачав вибір оптимальних технологій та інструментів для ефективної реалізації проекту. Важливою частиною було ретельне визначення мови програмування, яка найкраще підходила б для даного проекту, з урахуванням його специфіки та поставлених вимог. Також враховувалися середовища розробки, бібліотеки та інші інструменти, які могли б допомогти забезпечити швидку та ефективну розробку.

Під час аналізу було важливо врахувати специфіку завдання та потреби майбутнього користувача, щоб вибрати найбільш оптимальний набір технологій. Аналіз показав, що правильний вибір технологій є ключовим фактором для успішної реалізації проекту та досягнення його поставлених цілей.

Отже, результатом цього етапу було не лише визначення конкретних технологій та інструментів, а й глибоке розуміння їхнього впливу на процес розробки та функціональність майбутнього продукту.

У третьому розділі дипломної роботи було здійснено докладне проектування та програмну реалізацію телеграм бота. Цей процес передбачав розробку логіки взаємодії з користувачем створивши повну блок-схему у робочому просторі Miro, створення бази даних для зберігання необхідних даних та інформації, а також програмну реалізацію всієї необхідної функціональності бота. Кожен етап проектування та реалізації був ретельно промисловим, що дозволило врахувати всі технічні та функціональні вимоги до продукту.

Після завершення програмної реалізації було проведено широке тестування та оцінку результатів. Програма була піддана ретельному аналізу, щоб виявити та виправити всі можливі помилки та недоліки. Результати тестування показали, що програмний продукт відповідає всім вимогам та функціональності, встановленим на початку проекту та дійсно пришвидшив процес постингу.

Продемонстровано, що при потребі постингу 150 постів у 5 каналах протягом 30 днів знадобилось створити всього 9 постів замість 22500, що швидше рівно у 2500 разів.

Зрештою, у проекті не лише успішно було вирішено поставлені завдання, а й набуто цінного досвіду у розробці програмного забезпечення, який може бути успішно використаний у майбутніх проектах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Telegram Channels in Ukraine. 2024. URL: https://telemetr.io/en/catalog/ukraine (дата звернення: 20.05.2024)
2. Sajad Faramarzi, Hossein Heidari Tabrizi, Azizeh Chalak. Telegram an instant messaging application to assist distance language learning, 2019. 147 с.
3. Telegram канали. 2023. URL: https://www.site2b.ua/ua/web-blog-ua/telegram-kanal-oglyad-mozhlivostej-i-yak-jogo-rozkrutiti-z-nulya.html (дата звернення: 20.05.2024)
4. Що таке чат бот та як створити бота в телеграм. 2022. URL: https://marchenko.marketing/scho-take-chat-bot-ta-yak-stvoriti-bota-v-telegram/ (дата звернення: 20.05.2024)
5. Dan Morley. Netmark's 2016 Guide to the 6 fundamentals of Digital Marketing, 2016. 28 с.
6. Статистика власного телеграм каналу. 2024. URL: https://uk.tgstat.com/en (дата звернення 20.05.2024)
7. Про компанію SendPulse. 2021. URL: https://sendpulse.ua/about (дата звернення 20.05.2024)
8. Як створити чат-бот у Telegram. 2024. URL: https://sendpulse.ua/knowledge-base/chatbot/telegram/create-telegram-chatbot (дата звернення: 20.05.2024)
9. Bot API Library Examples. 2024. URL: https://core.telegram.org/bots/samples (дата звернення: 20.05.2024)
10. Aiogram 2.25.1 2023. URL: https://docs.aiogram.dev/en/v2.25.1/install.html (дата звернення: 20.05.2024)
11. Top 10 Python Libraries to Create Your Telegram Bot Easily. 2021. URL: https://blog.finxter.com/top-10-python-libraries-to-create-your-telegram-bot-easily-github/ (дата звернення: 20.05.2024)

12. Building Powerful Telegram Bots with Telethon in Python. 2023. URL: https://medium.com/@joloiuy/building-powerful-telegram-bots-with-telethon-in-python-76aef25902e9 (дата звернення: 20.05.2024)
13. Marino Miculan. Automated verification of Telegram's MTProto 2.0 in the symbolic model. Computers & Security. Вип. 126.
14. BaslayBot. 2022. URL: https://teletype.in/@elekosual_13/W8E2ePZvrCk (дата звернення: 20.05.2024)
15. Awesome Telegram bot for channel owners that helps you create scheduled posts, view stats and more. 2017. URL: https://telegra.ph/What-is-Controller-Bot-08-05 (дата звернення: 20.05.2024)
16. Delayed posting in Telegram using Notepost. 2022. URL: https://vlada-rykova.com/en/otlozhennyj-posting-v-telegram-s-pomoshhyu-notepost/ (дата звернення: 20.05.2024)
17. aiogram-media-group https://github.com/deptyped/aiogram-media-group
18. What is an IDE (Integrated Development Environment)? https://aws.amazon.com/what-is/ide/ (дата звернення: 10.06.2024)
19. What is MySQL? 2024. https://www.geeksforgeeks.org/what-is-mysql/ (дата звернення: 10.06.2024)
20. Josiah L. Carlson. Redis in Action. 2013. 320c.

ДОДАТКИ

Додаток А

Технічне завдання до бакалаврської дипломної роботи
Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

ЗАТВЕРДЖУЮ
завідувач кафедри АІІТ
_____ д.т.н., проф. Бісікало О. В.
«____» _____ 2024 року

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврської дипломної роботи
«Телеграм-бот паблішер на aiogram для автоматизації керування контентом у телеграм каналах»

Науковий керівник: к.т.н., доцент кафедри АІІТ
_____ Кабачій В. В
Студент групи 1АКІТ-20б
_____ Сметанюк Є.О.

ВНТУ 2024

1. Підстава для виконання бакалаврської дипломної роботи (БДР)

1.1 В сучасних умовах швидкого розвитку соціальних мереж та потреби в оперативному управлінні контентом, розробка інструментів, які автоматизують цей процес, є вкрай актуальною. Telegram є однією з найпопулярніших платформ для ведення каналів, а можливість планування необмеженої кількості постів дозволить значно підвищити ефективність роботи адміністраторів та редакторів.

1.2 Наказ про затвердження теми БДР.

2. Мета БДР і призначення розробки

2.1 Мета роботи — розробка телеграм бота паблішера, який забезпечить зручне та ефективне управління контентом у Telegram-каналах, долаючи обмеження на кількість запланованих повідомлень.

2.2 Призначення розробки — розробка телеграм бота призначена для автоматизації процесів управління контентом у Telegram-каналах, що значно спрощує роботу контент та SMM менеджерів.

3. Вихідні дані для виконання БДР

3.1 Аналіз потреб в автоматизації контент-менеджменту для Telegram-каналів, включаючи аналіз існуючих обмежень на кількість запланованих повідомлень.

3.2 Порівняльний огляд поточних рішень для управління контентом у Telegram, акцентуючи увагу на технічних реалізаціях, функціональних можливостях і обмеженнях щодо планування постів.

3.3 Розробка архітектури телеграм бота паблішера, яка включає інтеграцію з базою даних для зберігання медіа контенту, підключення до

Telegram API та використання Redis для оптимізації кешування і обробки задач.

3.4 Створення детальних алгоритмів функціонування бота, що охоплюють процеси створення, редагування, планування та публікації постів, а також управління кількома Telegram-каналами без обмежень на кількість запланованих постів.

3.5 Програмна реалізація телеграм бота паблішера на основі розробленої архітектури та алгоритмів, з акцентом на автоматизацію процесів управління контентом у Telegram-каналах.

4. Вимоги до виконання БДР

Головна вимога — створення програмного засобу, який забезпечить безлімітне планування та управління постами у Telegram-каналах, подолавши стандартне обмеження на кількість запланованих повідомлень.

5. Етапи БДР та очікувані результати

Етапи роботи та очікувані результати наведено у Таблиці А.1.

Таблиця А.1 — Етапи БДР

№	Назва та зміст етапу	Строк виконання етапів роботи	Очікуванні результати
1.	Аналіз предметної області	04.01-08.01.2024	Розділ 1
2.	Аналіз телеграм ботів паблішерів аналогів	09.01-11.01.2024	Розділ 1
3.	Розробка архітектури взаємодії бота з користувачем	12.01-20.01.2024	Розділ 2
4.	Створення схеми бази даних	21.01-23.01.2024	Розділ 2

5.	Програмна реалізація телеграм бота	24.01-10.02.2024	Розділ 3
6.	Оформлення пояснювальної записки, доповіді та презентації	14.05-10.06.2024	Записка, доповідь, презентація.
7.	Оформлення супроводжуючих документів	11.06-14.06.2024	Оформлені документи

6. Матеріали, що подаються до захисту БДР

На захист бакалаврської дипломної роботи необхідно представити ряд документів, включаючи пояснювальну записку, матеріали з ілюстраціями, висновок наукового керівника, анотації до роботи українською та іноземними мовами, і довідку, що засвідчує відповідність оформлення БДР вимогам.

7. Порядок контролю виконання та захисту БДР

Науковий керівник відстежує виконання графічної та розрахункової документації БДР відповідно до встановлених термінів. Захист роботи проводиться на засіданні Екзаменаційної комісії, яка має офіційний статус затверджений ректором.

8. Вимоги до оформлювання та порядок виконання БДР

8.1 При оформлювання БКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 – «Комп'ютерна інженерія» (освітня програма «Комп'ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2023;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання БКР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21»

Додаток Б

Лістинг частини файлу client.py

```
import datetime
import locale
import logging
import string
from copy import deepcopy
from typing import List
import aiogram.utils.exceptions
from aiogram import types, Dispatcher
from aiogram.dispatcher import FSMContext
from aiogram.dispatcher.filters import Text
from aiogram.dispatcher.filters.state import StatesGroup, State
from aiogram.types import Message, InlineKeyboardMarkup, InlineKeyboardButton
from aiogram_calendar import SimpleCalendar

from db_manage import Media
from utils.aiogram_media_group import media_group_handler
from create_bot import bot, scheduler
from handlers.catalog_handlers import media_type_from_cat, media_base_panel
from utils.utils import kb_channels, AuthMiddleware, send_voice_from_audio, restrict_media, set_caption, \
    send_post_to_channel, \
        pressed_back_button, sorting_key_jobs, show_post, job_list_by_channel, alert_vnote_text, paginate, \
    create_text, create_kb, create_random_media, catalog_paginate, update_page_num
from db_manage import get_all_channels, save_channel_json, remove_channel_id_from_json, catalog_list_db, \
                get_all_catalog_media, get_video_notes_by_cat, get_media_by_type
from keyboards.kb_client import (main_kb, kb_manage_channel_inline, cancel_kb, post_formatting_kb, add_channel_inline, \
```

```
                                        create_post_inline_kb, back_kb,
base_manage_panel_kb, enter_text_kb, del_voice_kb,
                                    add_posts_to_kb, take_from_db, \
                                      media_kb, inlines_menu_kb, back,
self_or_random_kb, del_post_inline, back_to_main_menu,
                                    back_edit_post_inline, \
                                                 change_create_post_kb,
create_post_inline, back_to_my_posts_inline,
                                               back_to_media_settings,
my_posts_inline, back_to_inlines)
    from aiogram_calendar import simple_cal_callback
    import random
    import numpy as np

    locale.setlocale(locale.LC_ALL, 'uk_UA.utf8')


    class FSMClient(StatesGroup):
        del_random_video_answer = State()
        skip_minutes_loop = State()
        new_inline_link = State()
        change_button_index = State()
        start_loop_date = State()
        skip_days_loop_vnotes = State()
        skip_days_loop = State()
        random_text_catalog = State()
        catalog_text_number = State()
        catalog_for_text = State()
        inline_to_delete = State()
        new_cat_name = State()
        time_random_video_notes = State()
        # random_v_notes_id = State()
        posts_by_data = State()
        all_posts_channel_id = State()
        inline_link = State()
        inline_text = State()
        channel_change_post = State()
```

```
        random_or_self = State()
        number_of_rand_video = State()
        number_of_rand_gifs = State()
        number_of_rand_photo = State()
        choose_catalog = State()
        user_name = State()
        post_text = State()
        remove_channel_id = State()
        channel_id = State()
        create_post_in_channel = State()
        date_planning = State()
        time_planning = State()
        time_loop = State()
        media_answer = State()

        create_cat_name = State()
        show_catalog = State()  # show catalog content
        edit_catalog = State()  # choose add or remove media
         add_delete_cat_media = State()  # if delete - pick a type if
add - load media
        catalog_media_type_remove = State()  # request media number
          del_cat_media_number = State()  # receive num of media and
remove
        loaded_catalog_file = State()
        catalog_for_post = State()
        del_catalog = State()
        media_type_add_from_cat = State()
        add_media_from_cat = State()

        loaded_post_files = State()
        voice = State()
        job_id = State()
        job_modify = State()
        del_voice_or_vnote_answer = State()
        del_media_answer = State()
```

```
    async def start_command(message: Message, state: FSMContext):
        await state.finish()
        if message.chat.type != types.ChatType.GROUP:
                                    await    message.answer(text=f'Вітаю,
{message.from_user.username}\n'
                                                f'Це бот для відкладеного
постингу. Для початку роботи '
                                            f'скористайтеся командами або
головним меню.\n'
                                                f'/addchannel – підключення
нового каналу',
                                    reply_markup=main_kb)


    async def main_menu(call: types.CallbackQuery, state: FSMContext):
        await state.finish()
        await call.answer()
                await    call.message.answer(text='Головне    меню    ⬇️',
reply_markup=main_kb)


    async   def   channel_manage_menu(message:   types.Message,   state:
FSMContext):
        await state.finish()
                 await    bot.send_message(chat_id=message.from_user.id,
text="Панель управління каналами",
                                reply_markup=kb_manage_channel_inline)


    async def deny_channel(message: types.Message):
        await FSMClient.remove_channel_id.set()
        all_channels = await get_all_channels(message.from_user.id)
                    channel_list     =     '\n'.join([f"{channel.name}
<code>{channel.channel_id}</code>" for channel in all_channels])

        await bot.send_message(chat_id=message.from_user.id,
```

```
                                    text=f'Надішліть id каналу, який хочете
відключити.\n\n'
                                                        f'{channel_list}',
reply_markup=cancel_kb, parse_mode="html")


    async def remove_channel_id(message: types.Message, state:
FSMContext):
        channel_id = str(message.text)
        if channel_id[1:].isdigit():
            await remove_channel_id_from_json(channel_id, message)
            await state.finish()
        else:
               await message.answer('id має складатись тільки з цифр,
надішліть ще раз.')


    async def add_channel(message, state: FSMContext):
        await state.finish()
        await FSMClient.channel_id.set()
        me = await bot.me
        bot_name = me.username

        if isinstance(message, types.CallbackQuery):
            await message.answer()
            await message.message.answer(
                        text="Щоб підключити канал, призначте бота його
адміністратором. Для цього:\n1. Перейдіть в канал.\n2. "
                             "Натисніть на назву каналу, щоб відкрити його
налаштування.\n3. Перейдіть в «Адміністратори» → «Додати "
                                    f"адміністратора».\n4. Введіть в пошуку
<code>@{bot_name}</code>. У результатах пошуку побачите вашого бота. "
                                     "Клікніть по ньому, потім натисніть
«Готово».\nПісля цього перешліть сюди будь-яке повідомлення з каналу.",
                    parse_mode='html', reply_markup=cancel_kb)
        else:
            await message.answer(
```

```
                    text="Щоб підключити канал, призначте бота його адміністратором. Для цього:\n1. Перейдіть в канал.\n2. "
                         "Натисніть на назву каналу, щоб відкрити його налаштування.\n3. Перейдіть в «Адміністратори» → «Додати "
                                f"адміністратора».\n4. Введіть в пошуку <code>@{bot_name}</code>. У результатах пошуку побачите вашого бота. "
                                    "Клікніть по ньому, потім натисніть «Готово».\nПісля цього перешліть сюди будь-яке повідомлення з каналу.",
                parse_mode='html', reply_markup=cancel_kb)


    async def load_channel_id(message: types.Message, state: FSMContext):
        if 'forward_from_chat' in message:
            channel_chat_id = str(message.forward_from_chat.id)
            me = await bot.get_me()
            bot_id = me.id
            bot_name = me.username
                                            chat_member = await bot.get_chat_member(chat_id=channel_chat_id, user_id=bot_id)
            bot_status = chat_member.status
            if bot_status == 'administrator':
                    await save_channel_json(channel_id=channel_chat_id, message=message)
                await state.finish()
            else:
                await state.finish()
                await message.answer(
                        text=f'⛔ Немає доступу. Переконайтеся, що бот <code>@{bot_name}</code> доданий до адміністраторів каналу '
                                                            f'<a href="{await message.forward_from_chat.get_url()}">{message.forward_from_chat.title}</a>.',
                    parse_mode='html')
        else:
             await message.answer(text='Потрібно переслати повідовлення саме з каналу.\n'
```

```
                                        'Спробуйте ще раз.')


    async def channel_list(call: types.CallbackQuery):
        await call.answer()
        all_channels = await get_all_channels(call.from_user.id)
        try:
            if all_channels:
                            channel_list  =  '\n'.join([f"{channel.name}
<code>{channel.channel_id}</code>" for channel in all_channels])

                        await  call.message.edit_text(text=channel_list,
parse_mode='html',

reply_markup=kb_manage_channel_inline)
            else:
                                                    add_channel_kb    =
InlineKeyboardMarkup().add(add_channel_inline)
                        await  call.message.edit_text(text='У  вас  немає
каналів', reply_markup=add_channel_kb)
        except:
            pass


    async    def    edit_create_post_channel_list(message,    state:
FSMContext):
        if isinstance(message, types.Message):
            await state.finish()
            if await get_all_channels(message.from_user.id):
                kb = await kb_channels(message, bot)
                kb.add(back_to_main_menu)
                if message.text == 'Створити пост':
                    await FSMClient.create_post_in_channel.set()
                      await message.answer(text='Оберіть канал, у якому
хочете створити пост:', reply_markup=kb,
                                           parse_mode='html')
                else:
```

```
                    await FSMClient.channel_change_post.set()
                      await message.answer(text='Оберіть канал, у якому
хочете змінити пост:', reply_markup=kb,
                                           parse_mode='html')
            else:
                await add_channel(message, state)
        elif isinstance(message, types.CallbackQuery):
            await message.answer()
            if await get_all_channels(message.from_user.id):
                kb = await kb_channels(message, bot)
                if message.data == 'Створити пост':
                    await state.finish()
                    await FSMClient.create_post_in_channel.set()
                    await message.message.answer(text='Оберіть канал, у
якому хочете створити пост:', reply_markup=kb,
                                                 parse_mode='html')
                elif message.data in ('Змінити пост', 'back'):
                    await FSMClient.channel_change_post.set()
                    try:
                          await message.message.edit_text(text='Оберіть
канал, у якому хочете змінити пост:', reply_markup=kb,

parse_mode='html')
                    except:
                        pass
                else:
                    try:
                        kb = InlineKeyboardMarkup()
                        kb.add(create_post_inline, my_posts_inline)
                             await message.message.answer(text='Бажаєте
створити новий пост або оглянути існуючі?',
                                                          reply_markup=kb)
                    except:
                        pass
            else:
                await add_channel(message.message, state)
```

```
    async def edit_post_list(message: types.CallbackQuery, state:
FSMContext):
        data = await state.get_data()
         if isinstance(message, types.CallbackQuery) and message.data
not in ('+', '-'):
            channel_id = message.data
            await state.update_data(edit_channel_id=channel_id)
        else:
            channel_id = data.get('edit_channel_id')
        if not data.get('page_num'):
            await state.update_data(page_num=1)
        data = await state.get_data()
        page_num = data.get('page_num')
        jobs = scheduler.get_jobs()
        edit_kb = InlineKeyboardMarkup()
        posts = []
        jobs = sorted(jobs, key=lambda job: job.next_run_time)
        for job in jobs:
            job_data = job.kwargs.get('data')
            if job_data.get('channel_id') == channel_id:
                posts.append(job)
        if len(posts) > 30:
            post_chunks = np.array_split(posts, len(posts) // 30)
        else:
            post_chunks = np.array_split(posts, 1)
        if page_num >= len(post_chunks):
            page_num = len(post_chunks)
            await state.update_data(page_num=page_num)
        posts = list(post_chunks[page_num - 1])
        if posts:
            await add_posts_to_kb(jobs=posts, edit_kb=edit_kb)
            edit_kb.add(back_edit_post_inline)
            await paginate(edit_kb)
            edit_kb.inline_keyboard[-1][-2].text = page_num
            try:
                await message.answer()
```

```
                    await message.message.edit_text(f'Ваші заплановані та
зациклені пости.\n'
                                                    'Оберіть потрібний
вам:', reply_markup=edit_kb)
            except Exception as err:
                logging.info(f'ERROR: {err}; {edit_kb}')
            await FSMClient.job_id.set()
        else:
            await message.answer()
            try:
                        await  message.message.edit_text('У  вас  немає
запланованих або зациклених постів.',

reply_markup=create_post_inline_kb)
            except:
                pass


    async    def    change_job(call:    types.CallbackQuery,    state:
FSMContext):
        await call.answer()
        data = await state.get_data()
        if call.data in ('+', "-"):
            await update_page_num(data, call, state)
            await edit_post_list(call, state)
            return
        job_id = call.data
        job = scheduler.get_job(job_id=job_id)
        if job:
            await state.update_data(job_id=job_id)
            await show_post(call, state)

            await state.reset_state(with_data=False)

            if job_id:
                kb_job = deepcopy(post_formatting_kb)
                del kb_job.inline_keyboard[-1][0]
```

```
                kb_job.add(del_post_inline)
                await call.message.answer(
                                text='Налаштуйте оформлення посту',
reply_markup=kb_job)
            else:
                await call.message.answer(
                                text='Налаштуйте оформлення посту',
reply_markup=post_formatting_kb)


    async def enter_change_text(message: types.CallbackQuery, state:
FSMContext):
        fsm_data = await state.get_data()
        job_id = fsm_data.get('job_id')
        if fsm_data.get('channel_id'):
            channel_id = fsm_data.get('channel_id')
        elif job_id:
            job_data = scheduler.get_job(job_id).kwargs.get('data')
            channel_id = job_data['channel_id']
        else:
            channel_id = message.data

        channel = await bot.get_chat(channel_id)
        channel_name = channel.title
        await state.update_data(channel_id=channel_id)

        await FSMClient.post_text.set()
        try:
            await message.message.edit_text(
                        text=f'Надішліть текст власноруч або оберіть
варіант:\n\n'
                             f'<i>Обрати з бази</i> - власноруч обрати
заготовлений текст з каталогу.\n'
                            f'<i>Рандом текст</i> - текст буде навмання
обиратись з певного каталогу.\n\n'
                        f'Канал: «{channel_name}».',
                reply_markup=enter_text_kb, parse_mode='html')
```

```
        except:
            pass
        await message.answer()


    async def load_changed_text(message, state: FSMContext):
        text = None
        if isinstance(message, types.Message):
            text = message.text
        else:
            await message.answer()
            if message.data in ('pick_text_from_db', 'random_text'):
                await pick_text_catalog(message, state)
                return
        data = await state.get_data()
        job_id = data.get('job_id')
        if job_id:
            job = scheduler.get_job(job_id)
            data = job.kwargs.get('data')
            data['post_text'] = text
            job.modify(kwargs={'data': data})
        else:
            await state.update_data(post_text=text)
        await show_post(message, state)

                await bot.send_message(chat_id=message.from_user.id,
text='Налаштуйте оформлення посту.',
                                       reply_markup=post_formatting_kb,
parse_mode='Markdown')
        await state.reset_state(with_data=False)


    async def pick_text_catalog(call: types.CallbackQuery, state:
FSMContext):
        catalogs_kb = await catalog_paginate(state)
                catalogs_kb.add(InlineKeyboardButton(text='«  Назад',
callback_data='Змінити текст'))
```

```
            await call.message.edit_text(text='Оберіть каталог, з якого бажаєте додати текст:', reply_markup=catalogs_kb)
        if call.data == 'pick_text_from_db':
            await FSMClient.catalog_for_text.set()
        elif call.data == 'random_text':
            await FSMClient.random_text_catalog.set()


    async def pick_text(call: types.CallbackQuery, state: FSMContext):
        await call.answer()
        data = await state.get_data()
        if call.data in ('+', "-"):
            await update_page_num(data, call, state)
            await pick_text_catalog(call, state)
            return
        cat_name = call.data
        texts = await get_media_by_type(cat_name, media_type='text')
        texts = [text.text for text in texts]
        catalogs_kb = await catalog_paginate(state)
                catalogs_kb.add(InlineKeyboardButton(text='«  Назад', callback_data='Змінити текст'))

        if texts:
            await state.update_data(catalog_for_text=cat_name)
            for text_num in range(len(texts)):
                                                message = await call.message.answer(text=texts[text_num])
                await message.reply(text=str(text_num + 1))
             await call.message.answer(text='Введіть номер тексту, який бажаєте обрати:')
            await FSMClient.catalog_text_number.set()
        else:
                  await call.message.edit_text('❌ У каталозі немає заготовлених текстів.', reply_markup=catalogs_kb)
```

```
    async def load_random_text(call: types.CallbackQuery, state: FSMContext):
        await call.answer()
        data = await state.get_data()
        if call.data in ('+', "-"):
            await update_page_num(data, call, state)
            await pick_text_catalog(call, state)
            return
        job_id = data.get('job_id')
        cat_name = call.data
        texts = await get_media_by_type(cat_name, media_type='text')
        texts = [text.text for text in texts]
        catalogs_kb = await catalog_paginate(state)
                catalogs_kb.add(InlineKeyboardButton(text='«  Назад', callback_data='Змінити текст'))

        if texts:
            if job_id:
                job = scheduler.get_job(job_id)
                job_data = job.kwargs.get('data')
                job_data['catalog_for_text'] = cat_name
            else:
                await state.update_data(catalog_for_text=cat_name)
        else:
                  await call.message.edit_text('❌ У каталозі немає заготовлених текстів.', reply_markup=catalogs_kb)
            return

        await show_post(call, state)
                  await  call.message.answer(text='Текст  обрано.', reply_markup=post_formatting_kb)
        await state.reset_state(with_data=False)


    async def set_text_in_post_from_db(message: types.Message, state: FSMContext):
        data = await state.get_data()
```

```
        job_id = data.get('job_id')
        catalog_for_text = data.get('catalog_for_text')
               texts   =   await   get_media_by_type(catalog_for_text,
media_type='text')
        texts = [text.text for text in texts]
        if message.text.isdigit() and int(message.text) <= len(texts):
            if job_id:
                job = scheduler.get_job(job_id)
                job_data = job.kwargs.get('data')
                job_data['text_index'] = message.text
                job_data['catalog_for_text'] = catalog_for_text
                job.modify(kwargs={'data': job_data})
            else:
                await state.update_data(text_index=message.text)
                await state.update_data(post_text=None)
            await show_post(message, state)
                        await   message.answer(text='Текст   обрано.',
reply_markup=post_formatting_kb)
            await state.reset_state(with_data=False)
        else:
            await message.answer(text='Невірне значення.\n'
                                      'Введіть номер тексту, який
бажаєте обрати:')
```

Додаток В

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

ТЕЛЕГРАМ-БОТ ПАБЛІШЕР НА AIOGRAM ДЛЯ АВТОМАТИЗАЦІЇ КЕРУВАННЯ КОНТЕНТОМ У ТЕЛЕГРАМ КАНАЛАХ

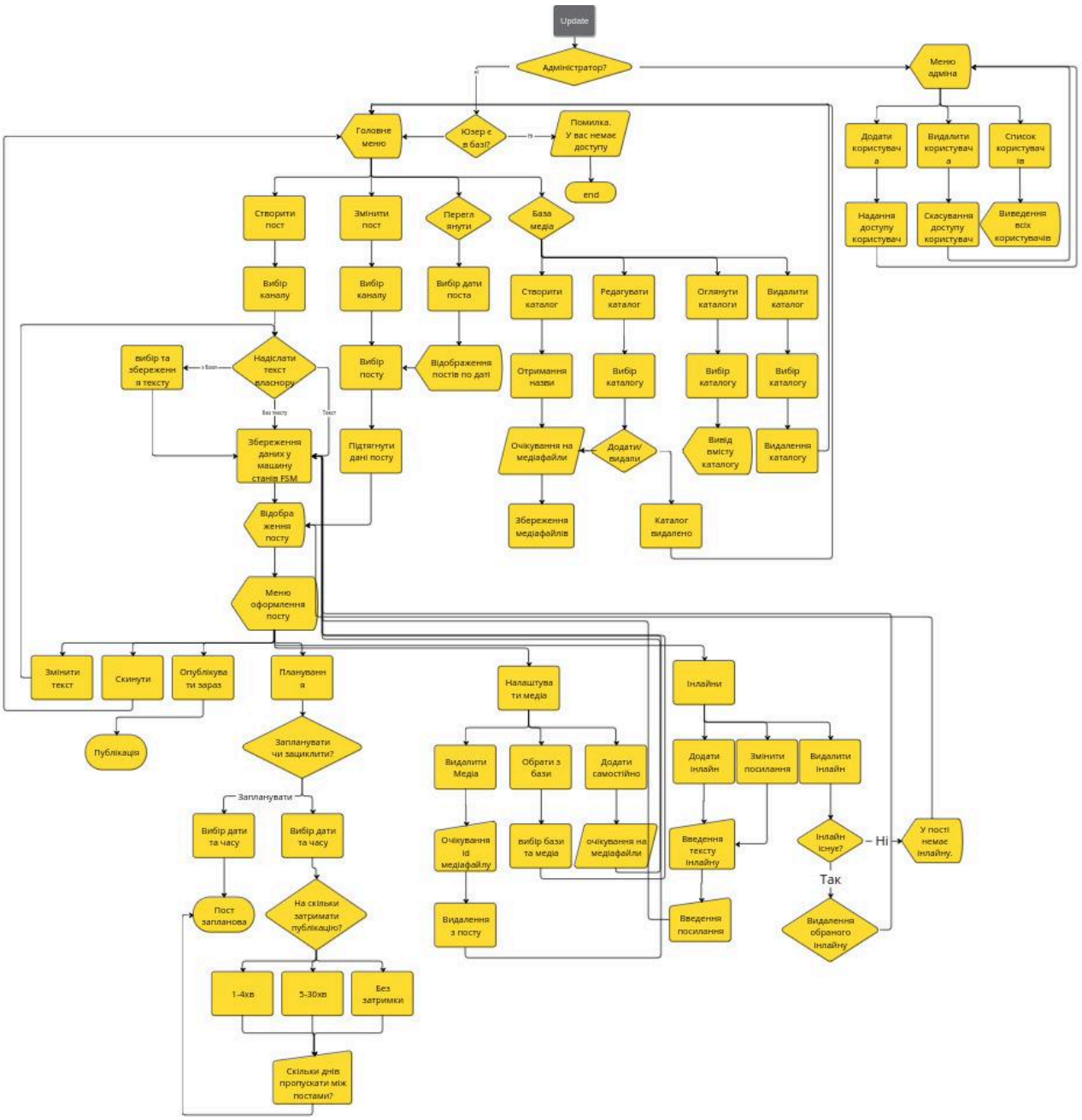


Рисунок В.1 – Блок схема логіки взаємодії користувача з ботом

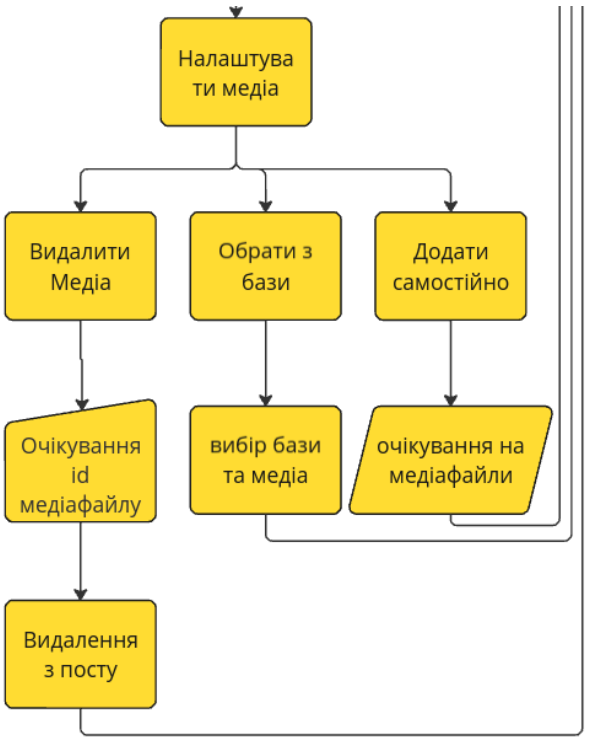


Рисунок В.2 – Блок схема налаштування медіа

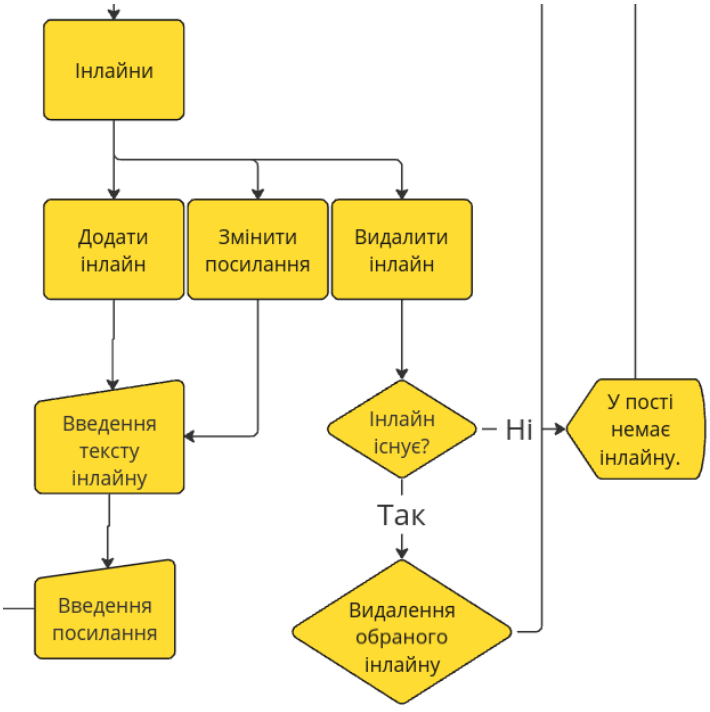


Рисунок В.3 – Блок схема налаштування інлайнових кнопок

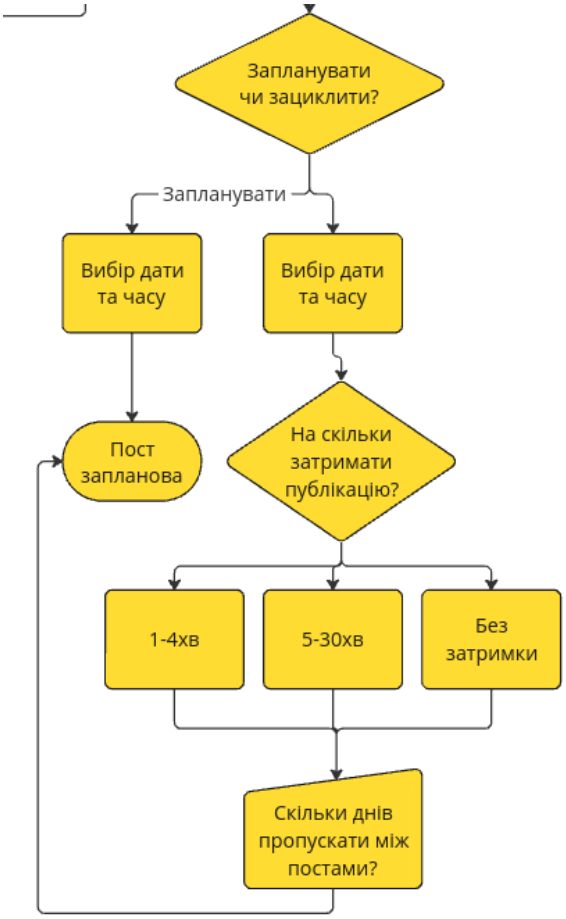


Рисунок В.4 – Блок схема налаштування планування публікації

Додаток Г

(обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ ДИПЛОМНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Телеграм-бот паблішер на aiogram для автоматизації керування контентом у телеграм каналах»

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ: кафедра АІІТ, ФІІТА, 1АКІТ-20б
(кафедра, факультет, навчальна група)

Науковий керівник: Кабачій В. В., доц. каф. АІІТ
(прізвище, ініціали, посада)

Показники звіту подібності Unicheck

Оригінальність 98.96% Схожість 1.04%

Аналіз звіту подібності (відмітити потрібне)

■ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень

Особа, відповідальна за перевірку _____ Овчинников К. В.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи

Автор _____ Сметанюк Є. О.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Кабачій В. В
(підпис) (прізвище, ініціали)