ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
СТВОРЕННЯ ЗАСТОСУНКУ ДЛЯ ЕФЕКТИВНОГО КЕРУВАННЯ ТА АНАЛІЗУ ПОВІДОМЛЕНЬ У МЕСЕНДЖЕРАХ

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістра

Виконавець роботи Кіпятков Володимир Сергійович
_____ «____»_____ 2025 р.
(підпис)

Науковий керівник к.ф.-м.н., доцент, Ольховська Олена Володимирівна
_____ «____»_____ 2025 р.
(підпис)

Рецензент

ПОЛТАВА 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 2023 р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Створення застосунку для ефективного керування та аналізу повідомлень у месенджерах**»**
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістр
Прізвище, ім'я, по батькові Кіпятков Володимир Сергійович
Затверджена наказом ректора № 206-Н від «27» жовтня 2023 р.
Термін подання студентом роботи «___»_____ 2024 р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки застосунків для керування повідомленнями у месенджерах.

Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ
1.1. Огляд сучасних месенджерів та їх функцій
1.2. Дослідження проблеми керування та аналізу повідомлень у месенджерах
1.3. Існуючі рішення та їхні обмеження
1.4. Вимоги до розробки програмного застосунку та формулювання основних завдань дослідження
РОЗДІЛ 2. АРХІТЕКТУРА ТА ТЕХНОЛОГІЇ РОЗРОБКИ ЗАСТОСУНКУ
2.1. Вибір архітектурного підходу та його реалізація
2.2. Огляд і вибір технологій
2.3. Структура та модулі застосунку
2.4. Опис методології створення програмного забезпечення застосунку
РОЗДІЛ 3. Розробка програмного застосунку
3.2. Інтеграція з API Telegram
3.3. Програмна реалізація функціональних модулів застосунку
3.4. Реалізація алгоритму аналізу повідомлень у чатах Telegram
3.5. Побудова діаграми класів розробленого ПЗ застосунку
РОЗДІЛ 4. АНАЛІЗ ТА ОЦІНКА ЕФЕКТИВНОСТІ ЗАСТОСУНКУ
4.1. Порівняльний аналіз із існуючими рішеннями
4.2. Інструкція користувача та функціональні можливості застосунку
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Перелік графічного матеріалу: 1 аркуш діаграм, 21 ілюстрація

Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Ольховська О.В.		
Інформаційний огляд	Ольховська О.В.		
Теоретична частина	Ольховська О.В.		
Практична частина	Ольховська О.В.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
Вступ		
Вивчення методичних рекомендацій та стандартів та звіт керівнику		
Постановка задачі		
Інформаційний огляд джерел бібліотек та інтернету		
Теоретична частина		
Практична частина		
Закінчення оформлення		
Доповідь студента на кафедрі		
Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____ 2023 р.

Здобувач вищої освіти _____ Кіпятков Володимир Сергійович
(підпис)

Науковий керівник _____ к.ф.-м.н., доц. Ольховська О.В.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2025 р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 2023 р.

Погоджено
Науковий керівник _____
к.пед.н., Олена ОЛЬХОВСЬКА
«____» _____ 2023 р.

План

дипломного проекту з фаху
спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
на тему «Створення застосунку для ефективного керування та аналізу повідомлень у месенджерах»
Прізвище, ім'я, по батькові Кіпятков Володимир Сергійович

ВСТУП
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ
1.1. Огляд сучасних месенджерів та їх функцій
1.2. Дослідження проблеми керування та аналізу повідомлень у месенджерах
1.3. Існуючі рішення та їхні обмеження
1.4. Вимоги до розробки програмного застосунку та формулювання основних завдань дослідження
РОЗДІЛ 2. АРХІТЕКТУРА ТА ТЕХНОЛОГІЇ РОЗРОБКИ ЗАСТОСУНКУ
2.1. Вибір архітектурного підходу та його реалізація
2.2. Огляд і вибір технологій
2.3. Структура та модулі застосунку
2.4. Опис методології створення програмного забезпечення застосунку
РОЗДІЛ 3. Розробка програмного застосунку
3.2. Інтеграція з API Telegram
3.3. Програмна реалізація функціональних модулів застосунку
3.4. Реалізація алгоритму аналізу повідомлень у чатах Telegram
3.5. Побудова діаграми класів розробленого ПЗ застосунку
РОЗДІЛ 4. АНАЛІЗ ТА ОЦІНКА ЕФЕКТИВНОСТІ ЗАСТОСУНКУ
4.1. Порівняльний аналіз із існуючими рішеннями
4.2. Інструкція користувача та функціональні можливості застосунку
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Здобувач вищої освіти _____ В. С. Кіпятков
(підпис)
«____» _____ 2023 р.

РЕФЕРАТ

Записка: 73 сторінки, 21 рисунок, 1 додаток, 14 літературних джерел.

Мета дипломної роботи є розробка програмного застосунку для ефективного керування та аналізу повідомлень у месенджерах.

Об'єкт дослідження - програмне забезпечення застосунку для керування та аналізу повідомлень у месенджері Telegram.

Робота базується на використанні методів системного аналізу, проєктування програмного забезпечення та сучасних інструментів для розробки веб- та мобільних додатків. Перелік методів включає також використання при розробці модульної архітектури, яка передбачає розподіл функціональності на окремі незалежні компоненти. Застосунок розроблений із використанням Python та PyCharm як основні інструменти розробки, доповнені використанням асинхронного програмування, об'єктно-орієнтованого підходу, і інтеграції з бібліотекою Telethon для роботи з Telegram API; середовище розробки PyCharm.

У ході виконання даного дослідження було розглянуто сучасні проблеми управління та аналізу повідомлень у месенджерах, розроблено програмний застосунок для вирішення цих завдань; сформульовано основні вимоги до розробки застосунку. Розроблений застосунок дозволяє здійснювати автентифікацію користувачів за допомогою api_id, api_hash, номеру телефону та коду підтвердження. Забезпечено функцію зчитування списку всіх чатів користувача, включаючи особисті повідомлення, групові чати та канали. Дані кожного чату автоматично зберігаються у окремі текстові файли з відповідною назвою, що містять базову інформацію та останні 50 повідомлень. Застосунок реалізує функціонал автоматичного створення папок для збереження даних, а також їх авто-видалення відповідно до налаштувань користувача. Використання мови програмування Python та середовища розробки PyCharm дозволило реалізувати ефективне рішення з використанням сучасних підходів,

таких як асинхронне програмування, об’єктно-орієнтоване проєктування та інтеграція з Telegram API через бібліотеку Telethon.

ЗМІСТ

ВСТУП

Актуальність. Сучасний світ характеризується стрімким зростанням обсягу інформації, що потребує ефективного керування та аналізу. Месенджери стали невід'ємною частиною повсякденного життя, забезпечуючи оперативний обмін інформацією, обмін файлами та координацію діяльності у реальному часі. Однак сучасні рішення не завжди забезпечують достатній рівень організації та аналітики, що підкреслює необхідність подальшого дослідження в цій галузі.

Робота має на меті розробку програмного застосунку для ефективного керування та аналізу повідомлень у месенджерах.

Обєкт дослідження – програмне забезпечення застосунку для керування та аналізу повідомлень у месенджері Telegram.

Предметом дослідження є методи та технології аналізу і оптимізації процесів керування повідомленнями у сучасних месенджерах (на прикладі Telegram). Робота базується на використанні методів системного аналізу, проєктування програмного забезпечення та сучасних інструментів для розробки веб- та мобільних додатків.

Новизна дослідження полягає у створенні програмного рішення, яке інтегрує кілька чатів у месенджері в єдиний інтерфейс із можливістю аналітичного оброблення повідомлень. Розроблений застосунок дозволяє здійснювати автентифікацію користувачів за допомогою api_id, api_hash, номеру телефону та коду підтвердження. Забезпечено функцію зчитування списку всіх чатів користувача, включаючи особисті повідомлення, групові чати та канали. Дані кожного чату автоматично зберігаються у окремі текстові файли з відповідною назвою, що містять базову інформацію та останні 50 повідомлень.

Застосунок також реалізує функціонал автоматичного створення папок для збереження даних, а також їх авто-видалення відповідно до налаштувань

користувача. Це дозволяє ефективно керувати обсягами збережених даних та уникнути перевантаження системи.

Практичне значення роботи полягає у підвищенні ефективності використання месенджерів як для особистих, так і для корпоративних потреб.

Перелік методів включає також використання при розробці модульної архітектури, яка передбачає розподіл функціональності на окремі незалежні компоненти. Застосунок розроблений із використанням Python та PyCharm як основні інструменти розробки, доповнені використанням асинхронного програмування, об’єктно-орієнтованого підходу, і інтеграції з бібліотекою Telethon для роботи з Telegram API; середовище розробки PyCharm.

Пояснювальна записка містить чотири основних розділи: аналіз предметної обслаті та постановка задачі; архітектура та технології розробки застосунку; розробка програмного застосунку; аналіз та оцінка ефективності застосунку. Структура роботи орієнтована на логічне представлення матеріалу та всебічне розкриття теми дослідження.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Огляд сучасних месенджерів та їх функцій

Сучасний ринок месенджерів представлений широким спектром платформ, що забезпечують користувачам можливість миттєвого обміну повідомленнями, мультимедійним контентом та організації групової взаємодії. Найбільш популярними месенджерами є такі, як Telegram, WhatsApp, Facebook Messenger, Viber та Signal. Вони надають користувачам широкий функціонал, включаючи обмін текстовими повідомленнями, аудіо- та відеодзвінки, можливість створення групових чатів, інтеграцію з іншими сервісами, а також високий рівень захисту даних завдяки шифруванню.

Основні функції, що пропонуються сучасними месенджерами, включають підтримку різних форматів даних, можливість здійснення голосових та відеодзвінків, інтеграцію з іншими платформами, такими як CRM-системи, а також використання чат-ботів для автоматизації бізнес-процесів. Окрім цього, користувачам доступні інструменти для пошуку повідомлень, архівації даних, персоналізації інтерфейсу та взаємодії через десктопні та мобільні версії додатків.

Значну увагу розробники месенджерів приділяють питанням безпеки, забезпечуючи наскрізне шифрування повідомлень, автентифікацію користувачів за допомогою двофакторної авторизації та контроль доступу до особистих даних. Розвиток технологій штучного інтелекту дозволяє впроваджувати автоматичне сортування повідомлень, розпізнавання емоційного забарвлення текстів та інтеграцію з голосовими помічниками.

Загалом, сучасні месенджери не лише виконують функцію засобу комунікації, а й стають потужним інструментом для організації робочих процесів та взаємодії в рамках бізнесу.

1.2. Дослідження проблеми керування та аналізу повідомлень у месенджерах

Зі збільшенням обсягу повідомлень, що обробляються сучасними месенджерами, виникають значні труднощі у їх ефективному керуванні та аналізі. Основними проблемами є велика кількість несистематизованої інформації, ускладненість пошуку конкретних повідомлень, необхідність моніторингу важливих даних та забезпечення конфіденційності. Використання традиційних методів зберігання та обробки даних часто не відповідає потребам сучасних користувачів, оскільки не дозволяє швидко отримувати необхідну інформацію та проводити її аналітичну обробку.

Важливою проблемою є також безпека повідомлень, зокрема захист від несанкціонованого доступу, витоку конфіденційних даних та використання алгоритмів шифрування, які можуть бути вразливими до атак. Крім того, існує необхідність у розробці ефективних механізмів автоматизації аналізу повідомлень, таких як класифікація, сортування та виявлення тенденцій на основі штучного інтелекту.

Таким чином, вирішення проблеми ефективного керування та аналізу повідомлень потребує розробки сучасних програмних рішень, що поєднують інструменти автоматичної обробки текстів, забезпечення безпеки та зручного користувацького інтерфейсу.

1.3. Існуючі рішення та їхні обмеження

На сьогоднішній день існує ряд програмних рішень для керування та аналізу повідомлень у месенджерах. Серед найбільш відомих на світовому ринку можна виділити такі платформи, як Hootsuite, Sprout Social, Buffer, Zoho Social та Chatbase. Ці рішення пропонують можливості інтеграції з популярними месенджерами, зокрема Facebook Messenger, WhatsApp, Telegram,

дозволяючи користувачам здійснювати моніторинг, аналітику та автоматизацію обробки повідомлень [13, 14].

Інструменти Hootsuite, Sprout Social, Buffer, Zoho Social та Chatbase є провідними рішеннями для управління соціальними мережами та аналізу комунікаційних даних, що широко застосовуються в бізнес-середовищі для автоматизації, планування та моніторингу активностей у соціальних платформах. Кожне з цих рішень має свої особливості, що визначають їх конкурентоспроможність, зручність використання та технологічний стек, що забезпечує їхню функціональність.

Hootsuite є одним із найпопулярніших інструментів для управління соціальними медіа, пропонуючи широкий спектр функцій, таких як планування публікацій, аналітика ефективності та інтеграція з багатьма соціальними платформами, зокрема Facebook, Twitter, LinkedIn та Instagram. Платформа надає можливість централізованого управління контентом, співпраці між членами команди та автоматизації процесів публікації.

Рисунок 1.1. Лого Hootsuite

Серед основних недоліків Hootsuite можна виокремити високу вартість підписки для корпоративного сегменту та складність освоєння інтерфейсу для нових користувачів через велику кількість доступних опцій.

Для розробки Hootsuite використовуються такі технології, як JavaScript (зокрема фреймворки React.js для фронтенду), Node.js для бекенду, а також MongoDB для зберігання даних. Інструмент також інтегрується з RESTful API соціальних платформ.

Sprout Social надає користувачам глибоку аналітику, розширену візуалізацію даних та інтеграцію з CRM-системами, що дозволяє ефективно

взаємодіяти з аудиторією. Завдяки інструментам для аналізу соціальних медіа Sprout Social допомагає підприємствам оцінювати вплив своїх комунікаційних стратегій і адаптувати маркетингові кампанії.

Рисунок 1.2. Лого Sprout Social

Висока вартість підписки та обмеження щодо підтримки менш популярних платформ є основними недоліками Sprout Social. Крім того, деякі користувачі відзначають потребу в глибшій кастомізації аналітичних звітів.

Sprout Social розроблений з використанням Ruby on Rails для бекенду, що забезпечує швидкість розробки та масштабованість. Фронтенд реалізований на основі React.js, а для аналітики використовується PostgreSQL як основна база даних.

Основною перевагою Buffer є простота у використанні, що робить його популярним серед малого бізнесу та незалежних маркетологів. Інструмент забезпечує легке планування та управління контентом у соціальних мережах, а також має прозору цінову політику з гнучкими тарифними планами.

Рисунок 1.3. Лого Buffer

Buffer має обмежений функціонал у порівнянні з конкурентами, зокрема відсутність комплексної аналітики, менш потужні функції командної роботи та обмежені можливості інтеграції з CRM-системами.

Buffer використовує Node.js для серверної частини, а фронтенд побудований на базі Vue.js. Для зберігання даних використовуються MySQL та Redis, що забезпечують швидкий доступ до ключової інформації.

Zoho Social забезпечує інтеграцію з екосистемою Zoho, що дозволяє бізнесу використовувати єдину платформу для управління соціальними мережами, продажами та клієнтськими комунікаціями. Інструмент пропонує розширені можливості автоматизації публікацій, інструменти моніторингу соціальних мереж та аналізу ефективності кампаній.

Рисунок 1.4. Лого Zoho Social

Попри інтеграцію з іншими продуктами Zoho, деякі користувачі відзначають, що інтерфейс може бути перевантаженим і складним для початківців. Крім того, підтримка деяких специфічних соціальних платформ є обмеженою.

Zoho Social побудований на платформі Java, що забезпечує стабільність і продуктивність системи. У фронтенд-розробці використовуються AngularJS, а для збереження даних застосовуються Oracle DB та NoSQL-сховища, що забезпечують гнучкість у роботі з великими обсягами інформації.

Chatbase є потужним інструментом для аналізу ефективності чат-ботів і взаємодії з користувачами. Платформа забезпечує відстеження тенденцій, побудову воронок спілкування та визначення потенційних проблем у роботі чат-ботів. Однією з ключових переваг є можливість інтеграції з популярними месенджерами, такими як Telegram і Facebook Messenger.

Основними обмеженнями Chatbase є залежність від платформ, на яких працюють чат-боти, а також обмежений набір функцій для кастомізації

аналітики. Крім того, система може потребувати значних ресурсів для обробки великих обсягів даних.

Chatbase розроблений за допомогою Python для аналізу даних, а серверна частина працює на Google Cloud Platform, використовуючи BigQuery для зберігання та обробки великих обсягів даних. У фронтенді застосовується React.js, що забезпечує гнучкий інтерфейс користувача.

Кожен із розглянутих інструментів має свої унікальні особливості та переваги, які визначають його доцільність використання в різних бізнес-сценаріях. Hootsuite і Sprout Social є потужними корпоративними рішеннями з розширеними аналітичними можливостями та підтримкою командної роботи, тоді як Buffer і Zoho Social пропонують більш прості рішення, орієнтовані на малий та середній бізнес. Chatbase, у свою чергу, спеціалізується на аналізі взаємодії з користувачами чат-ботів, що робить його ідеальним вибором для компаній, які прагнуть покращити автоматизовану взаємодію з клієнтами. Технологічний стек кожного з цих рішень містить сучасні інструменти, що забезпечують високу продуктивність, масштабованість та інтеграцію з іншими сервісами.

Зокрема, Hootsuite та Sprout Social орієнтовані на корпоративних користувачів і пропонують розширені аналітичні функції, такі як аналіз залученості, відстеження ключових показників продуктивності та візуалізація даних. Chatbase, своєю чергою, забезпечує аналіз чат-ботів у месенджерах, дозволяючи оцінювати ефективність взаємодії з користувачами.

Основними обмеженнями існуючих рішень є висока вартість ліцензій, обмеження у підтримці різних платформ одночасно, складність інтеграції з кастомними рішеннями, а також недостатній рівень персоналізації аналізу для індивідуальних користувачів.

Таким чином, існуючі програмні рішення покривають широкий спектр потреб бізнесу та користувачів, однак все ще залишаються певні виклики, пов’язані з кастомізацією, безпекою та комплексністю впровадження.

1.4. Вимоги до розробки програмного застосунку та формулювання основних завдань дослідження

Розробка програмного застосунку для ефективного керування та аналізу повідомлень у месенджерах вимагає врахування ряду критично важливих аспектів. Насамперед, застосунок повинен підтримувати інтеграцію з кількома платформами одночасно, такими як Telegram, WhatsApp, Facebook Messenger та Signal, забезпечуючи єдиний інтерфейс для обробки повідомлень. Важливим є впровадження механізмів автоматичної обробки даних, таких як сортування повідомлень за категоріями, аналіз тональності та виявлення ключових тем.

Крім того, необхідно врахувати вимоги щодо безпеки та конфіденційності, включаючи наскрізне шифрування, контроль доступу та відповідність нормативним стандартам захисту персональних даних. Важливим критерієм є масштабованість системи, що дозволить обробляти великі обсяги інформації без зниження продуктивності.

Окремої уваги потребує зручність інтерфейсу користувача, що має забезпечувати інтуїтивно зрозумілу навігацію та персоналізовані налаштування для різних категорій користувачів. Висока швидкодія, ефективний механізм пошуку та підтримка мобільних пристроїв є ключовими вимогами для забезпечення комфортного використання застосунку.

Основними завданнями дослідження є розробка програмного застосунку, який забезпечить ефективне керування повідомленнями у месенджерах (на прикладі Telegram). Насамперед необхідно реалізувати механізм автентифікації користувачів за допомогою api_id, api_hash, номеру телефону та коду підтвердження. Важливим завданням є зчитування списку всіх чатів користувача, включаючи особисті повідомлення, групові чати та канали, для їх подальшого аналізу.

Значну увагу необхідно приділити збереженню отриманих даних, зокрема реалізації функції автоматичного створення окремих текстових файлів для кожного чату, назва яких відповідатиме назві чату. Файли повинні містити базову інформацію про чат та до 50 останніх повідомлень, що дозволить користувачам швидко отримати доступ до необхідної інформації.

Крім того, дослідження передбачає реалізацію функціоналу автоматичного створення папок для збереження даних та механізмів їх авто-видалення відповідно до налаштувань користувача, що дозволить ефективно керувати обсягами збережених даних та уникнути перевантаження системи.

РОЗДІЛ 2. АРХІТЕКТУРА ТА ТЕХНОЛОГІЇ РОЗРОБКИ ЗАСТОСУНКУ

2.1. Вибір архітектурного підходу та його реалізація

У сучасній розробці програмного забезпечення для застосунків, які одночасно взаємодіють із зовнішніми сервісами (такими як Telegram та Google Sheets) і базою даних, можна виокремити кілька домінантних архітектурних підходів. Одним із найпоширеніших рішень є багатошарова (Layered) архітектура, коли застосунок ділиться на логічні рівні: від обробки запитів і відображення даних до бізнес-логіки та шару доступу до даних. Така модель зручна для невеликих і середніх проєктів, оскільки чітко розподіляє відповідальність між шарами, однак у великих проєктах кожен шар може стати громіздким і важче керованим.

Більш структурованою та гнучкою вважається Hexagonal архітектура, також відома як «Ports and Adapters», де центральну частину (ядро) становить доменна логіка, а доступ до зовнішніх систем реалізується через порти й адаптери. Цей підхід не прив’язує ключову логіку до конкретних протоколів або сервісів і полегшує тестування, оскільки адаптери легко підмінити. Цибулинна (Onion) архітектура дотримується того ж самого принципу суворого розділення ядра від інфраструктурних деталей, завдяки чому її часто називають одним із варіантів «чистої» архітектури. У ній центральною частиною так само є доменна модель, яку послідовно «обгортають» інші шари, що відповідають за взаємодію з БД та зовнішніми сервісами.

Подієва (Event-driven) архітектура зазвичай вибудовується навколо підходу, в межах якого ключову роль відіграють події, які генеруються системою або надходять іззовні. Застосунок реагує на такі події через асинхронні повідомлення або брокери подій на кшталт Kafka чи RabbitMQ. Це сприяє високій масштабованості та дозволяє легко розширювати функціонал за

рахунок нових сервісів або мікросервісів, що підписуються на відповідні події.

Альтернативою може виступати мікросервісна модель, коли великий проєкт поділяється на незалежні сервіси з власними базами даних, орієнтовані на конкретні бізнес-завдання. Такий метод дає можливість кожній команді працювати над окремим сервісом з автономним циклом розгортання та оновлення, проте збільшує складність координації, а також висуває підвищені вимоги до моніторингу й узгодження взаємодії між сервісами. У протилежність цьому підходу монолітна структура з чіткою внутрішньою модульністю залишається найпростішим способом організації застосунку. Для відносно невеликих систем можна зберігати єдину кодову базу, розклавши її по пакетах або директоріях відповідно до виконуваних функцій, що спрощує розгортання й базове підтримання коду. Однак у міру зростання обсягу функціоналу й команди моноліт може стати важким для масштабування.

Окремо також можна згадати serverless-підходи, коли основна логіка виокремлена у вигляді автономних функцій, що виконуються в хмарному середовищі (AWS Lambda, Google Cloud Functions тощо). Така модель вимагає мінімальних інфраструктурних зусиль, адже провайдер забезпечує динамічне масштабування, проте може ускладнити роботу з підключенням до баз даних через особливості мережевих обмежень і потребує ретельного планування викликів, щоб уникати затримок, пов’язаних із «холодним стартом». Вибір конкретного архітектурного підходу залежить від масштабу проєкту, очікуваних навантажень, можливостей команди розробників та вимог до швидкості й зручності розгортання.

Зважаючи на функціональні особливості нашого застосунку розробка буде базуватися на використанні модульної архітектури, яка передбачає розподіл функціональності на окремі незалежні компоненти. Основними інструментами розробки буде мова програмування Python та середовище розробки PyCharm. При розробці застосунку буде використано асинхронне програмування для забезпечення ефективної обробки запитів та взаємодії з

Telegram API за допомогою бібліотеки Telethon. Об’єктно-орієнтований підхід дозволить структурувати код, спрощуючи його підтримку та розширення.

У нашому програмному забезпеченні розробленого застосунку для керування та аналізом повідомлень у месенджерах (на прикладі Telegram) реалізовано наступну архітектуру:

1. Модульність і розподіл відповідальності (Separation of Concerns).

- Код розділений на кілька логічних частин (робота з БД, робота з Telegram API, інтеграція з Google Sheets), кожна з яких виконує свою роль.
- Клас DataBaseUseCase відповідає за з’єднання та взаємодію з базою даних, зокрема через SSH-тунель.
- Клас TelegramParseChat відповідає за обробку даних із Telegram (отримання повідомлень і чатів).
- Окремі функції для роботи з Google Sheets (update_google_sheets_, format_data_for_sheets) утворюють ще один «модуль».

Такий поділ відображає принцип єдиної відповідальності (Single Responsibility Principle) зі «SOLID» та дає змогу легше підтримувати й розвивати код.

2. Шаровість (Layered Architecture) або багатошарова архітектура

У коді проглядається щонайменше три умовні «шари» (хоч формально вони не виокремлені в окремі пакети):

1) Data/Infrastructure layer:
   - DataBaseUseCase (з’єднання з БД, доступ до даних).
   - Функції для доступу до Google Sheets (запис даних).
2) Application/Logic layer - логіка збирання даних із Telegram (TelegramParseChat) та виклику методів БД (вставка/вибірка даних) і Google Sheets (оновлення).

   Presentation/Entry point - головний блок if __name__ == ‘__main__’:, який викликає upload_data_google_sheet тощо.

Така структуризація часто застосовується в багатошаровій архітектурі, коли кожен шар виконує свою роль і взаємодіє з іншими шарами.

3. Об’єктно-орієнтований підхід

- Код використовує класи DataBaseUseCase, TelegramParseChat, MessageModel, ChatModel.
- Кожен клас інкапсулює дані та методи, що пов’язані з певною сутністю (для прикладу, DataBaseUseCase зберігає стан SSH-тунелю й з’єднання, а TelegramParseChat має методи для отримання повідомлень і чатів).

Це свідчить про використання ООП (об’єктно-орієнтованого підходу) з метою зробити код більш гнучким і зрозумілим.

4. Agile розробка

У моєму проекті я вирішив дотримуватися Agile моделі проєктування, яка відома своєю гнучкістю та здатністю адаптуватися до змін протягом усього процесу розробки. Agile підхід дозволяє швидко реагувати на нові вимоги та ітеративно вдосконалювати продукт:

- Є базовий функціонал (вставка даних у БД).
- Згодом додано новий функціонал (перевірка й оновлення Google Sheets).
- Далі інтегровано TelegramParseChat для завантаження даних із Telegram.

В реальних умовах це зазвичай відбувається невеликими ітераціями, де кожен новий модуль (чи можливість) поступово додається й інтегрується в існуючу систему.

5. Принципи KISS та DRY

- KISS (Keep It Simple, Stupid) — код відносно простий: без зайвих фреймворків, логіка викладена явно, без складних патернів.
- DRY (Don’t Repeat Yourself) — замість дублювання запитів до БД чи логіки доступу, усе зведено до методів _insert_chat, _insert_message, _select_all_chats_by_type у DataBaseUseCase.

6. Патерн доступу до даних (Data Access Object / DAO).

Методи _insert_message, _insert_chat, _select_all_chats_by_type реалізують

доступ до таблиць через єдину «точку входу» — DataBaseUseCase. Це можна вважати спрощеним застосуванням патерну DAO (Data Access Object), де всі операції з даними зосереджені у відповідному класі (об’єкті).

7. Конфігурація та винос секретів (Configuration Management).

- Усі конфігурації (API-ключі, паролі) винесені в окремі файли (private_config.py, config.py).
- Така практика відповідає принципам безпеки, оскільки уникає «хардкоду» секретних даних у головному коді.

2.2. Огляд і вибір технологій

Для реалізації поставлених завдань нами було обрано наступні інструменти та технології:

1. Python. Python підтримує різні стилі програмування, включаючи процедурний, об’єктно-орієнтований та функціональний, дозволяючи мені вибирати найбільш підходящий підхід для кожного завдання. Широкий спектр доступних бібліотек, таких як Telethon для взаємодії з Telegram, значно прискорює розробку та зменшує кількість необхідного коду.

2. PyCharm. Автоматичне завершення коду, перевірка синтаксису та рефакторинг у PyCharm покращують ефективність моєї роботи та допомагають уникати помилок. Вбудовані інструменти для налагодження дозволяють легко знаходити та усувати помилки, що робить процес розробки більш гладким та менш стресовим.

3. Асинхронне програмування. Використання асинхронних функцій і `await` дозволяє моєму скрипту виконувати мережеві запити та інші I/O операції без блокування, зберігаючи високу продуктивність. Асинхронне програмування надає кращий контроль над потоком виконання програми, що

спрощує обробку одночасних завдань.

4. Об’єктно-орієнтоване програмування (ООП). Завдяки використанню класів і об’єктів ми можемо структурувати свій проект таким чином, щоб кожна частина була ізольованою та відповідала за свою конкретну функцію. Інкапсуляція та абстракція - ці принципи ООП дозволяють мені приховувати складні деталі реалізації всередині класів та представляти об’єкти на більш високому, абстрактному рівні.

5. Telethon і Telegram API. Telethon надає простий і ефективний спосіб взаємодії з Telegram API, дозволяючи легко отримувати доступ до чатів, повідомлень та інших даних. Використання цієї бібліотеки дозволяє автоматизувати процес збору даних з моїх чатів у Telegram, значно спрощуючи завдання, яке вручну було б дуже часомістким.

Розглянемо більш докладно кожен із обраних нами інструментів.

Telegram API (Application Program Interface)

Telegram API є комплексним інструментом, який надає розробникам можливість взаємодіяти з різними компонентами месенджера Telegram для створення, управління та розширення функціоналу чат-ботів і сторонніх додатків. Він розроблений для забезпечення широких можливостей автоматизації та інтеграції, дозволяючи створювати високоефективні інтерактивні рішення. Завдяки цьому API розробники можуть реалізовувати автоматизовану взаємодію з користувачами, забезпечуючи обмін повідомленнями, обробку подій та управління даними у межах екосистеми Telegram [2].

Рисунок 2.1. Лого Telegram API.

Однією з ключових переваг Telegram API є його висока продуктивність і надійність, що дає змогу розробникам створювати ботів, здатних ефективно обробляти значний обсяг запитів від користувачів, а також гарантує оперативну доставку повідомлень. API пропонує широкий набір функціональних можливостей, які дозволяють адаптувати ботів до специфічних завдань і потреб користувачів. Завдяки підтримці інтеграції з іншими сервісами через механізм webhooks, можливості створення інтерактивних клавіатур та вбудованих методів для обробки мультимедійного контенту, таких як фото, аудіо та відео, Telegram API забезпечує гнучкість у розробці [2].

Одним із важливих аспектів використання Telegram API є дотримання високих стандартів конфіденційності та безпеки, що гарантує надійний захист особистих даних користувачів і приватності їхніх розмов. Завдяки цим характеристикам Telegram API створює сприятливі умови для розробників, які прагнуть створювати функціональні та зручні чат-боти в межах екосистеми месенджера. Поєднання гнучкості, продуктивності та надійності дозволяє реалізовувати різноманітні ідеї та забезпечувати якісне задоволення потреб користувачів [2].

Мова програмування Python

Python - це універсальна високорівнева мова програмування, яка поєднує простоту синтаксису, високу продуктивність і гнучкість. Вона була розроблена

наприкінці 1980-х років Гвідо ван Россумом і з часом набула широкого визнання серед розробників завдяки інтуїтивності та легкості у використанні. Основним принципом мови є зручність читання коду, що сприяє швидкому освоєнню та ефективній роботі з програмним забезпеченням. [3, 4].

Рисунок 2.2. Лого мови програмування Python.

Однією з визначальних характеристик мови програмування Python є її висока читабельність і лаконічність, що значно спрощує процес написання та розуміння коду. Завдяки цим властивостям Python є зручним для швидкого засвоєння, що сприяє ефективному розвитку програмних продуктів. Розгалужена стандартна бібліотека мови забезпечує розробників широким спектром інструментів для виконання різноманітних завдань, зокрема у сферах веб-розробки, аналізу даних та наукових обчислень.

Python підтримує різноманітні парадигми програмування, включаючи процедурний, об’єктно-орієнтований та функціональний підходи. Така гнучкість робить його універсальним засобом для вирішення широкого кола завдань у різних галузях. Значною перевагою є велика та активна спільнота користувачів, яка надає доступ до численних бібліотек, фреймворків та інструментів, що полегшують процес розробки та усунення можливих технічних труднощів.

У сфері розробки чат-ботів Python демонструє особливу ефективність завдяки своїй здатності легко інтегруватися з численними API та базами даних. Його продуктивність та масштабованість дозволяють створювати рішення, здатні адаптуватися до потреб різних бізнес-середовищ та забезпечувати

стабільну роботу ботів незалежно від обсягів взаємодії з користувачами [3, 4].

Таким чином, Python є потужним інструментом для сучасних розробників, забезпечуючи можливість створення швидких, ефективних і масштабованих додатків. Поєднання гнучкості, інтуїтивного синтаксису та широкого функціоналу робить його оптимальним вибором для розробки чат-ботів, здатних забезпечити користувачам високоякісний досвід взаємодії.

Бібліотека Telethon

Telethon - це асинхронна бібліотека для Python, яка забезпечує зручний доступ до API месенджера Telegram. Вона дозволяє взаємодіяти з платформою Telegram, надаючи широкий спектр функцій для автоматизації та інтеграції різноманітних сервісів. Завдяки підтримці асинхронного програмування Telethon забезпечує ефективне використання ресурсів, дозволяючи виконувати велику кількість запитів до API без блокування основного потоку виконання програми [5].

Рисунок 2.3. Логотип бібліотеки Telethon.

Основою роботи з Telethon є використання клієнта, який встановлює з’єднання з сервером Telegram і забезпечує доступ до всіх доступних методів API. Бібліотека підтримує автентифікацію користувача за допомогою телефонного номера або токена бота, що дозволяє реалізовувати як користувацькі, так і автоматизовані сценарії. Telethon надає розширені можливості для роботи з повідомленнями, контактами, групами, каналами та медіафайлами, дозволяючи здійснювати надсилання та отримання повідомлень,

завантаження та збереження мультимедійних даних, а також керування підписками та учасниками спільнот.

Однією з ключових особливостей бібліотеки є можливість прослуховування подій у реальному часі. Завдяки цьому користувач може реагувати на вхідні повідомлення, зміни статусу користувачів та інші події у чатах. Такий підхід робить Telethon корисним інструментом для створення чат-ботів, аналітичних систем та інструментів для автоматизації роботи в Telegram.

Архітектура Telethon спроектована таким чином, щоб забезпечити високу продуктивність та масштабованість. Бібліотека використовує сучасні підходи до організації роботи з мережею, що дозволяє ефективно працювати навіть за умов обмеженої пропускної здатності каналу зв’язку. Важливою особливістю є можливість здійснення пакетної обробки запитів, що дозволяє значно зменшити кількість викликів до API та підвищити загальну ефективність взаємодії з Telegram.

Бібліотека активно розвивається і підтримує актуальні версії Telegram API, що гарантує її сумісність із новими функціями месенджера. Вона має детальну документацію та активну спільноту користувачів, що спрощує процес вивчення та інтеграції у проекти різної складності. Завдяки своїй гнучкості, потужності та простоті використання, Telethon є популярним вибором серед розробників, які працюють із Telegram API [2, 5].

Середовище розробки (IDE) PyCharm

PyCharm – це професійне інтегроване середовище розробки (IDE), яке призначено специфічно для програмування на мові Python. Розроблене компанією JetBrains [6-9].

Рисунок 2.4. Лого середовища розробки PyCharm.

PyCharm є сучасним інтегрованим середовищем розробки, яке надає розробникам широкий спектр потужних інструментів для ефективного написання, діагностики та оптимізації програмного коду на мові Python. Завдяки розширеному функціоналу, PyCharm дозволяє значно підвищити продуктивність розробки та забезпечує високу якість програмного забезпечення.

Однією з ключових особливостей PyCharm є його інтелектуальні механізми виявлення помилок, які дозволяють автоматично аналізувати код і знаходити потенційні проблеми ще на ранніх етапах розробки. Автодоповнення коду та потужні засоби навігації по проєкту сприяють швидкому і зручному написанню програм, тоді як можливості рефакторингу дозволяють ефективно вдосконалювати та оптимізувати кодову базу.

PyCharm містить вбудований інструмент для налагодження програмного забезпечення, що забезпечує зручність у процесі тестування і виявлення помилок. Завдяки підтримці точок переривань, відстеженню значень змінних та покроковому виконанню коду, розробники можуть детально аналізувати процес виконання програми та оперативно вносити необхідні корективи [6-9].

Інтеграція PyCharm із системами контролю версій, такими як Git та Mercurial, забезпечує повноцінне управління змінами в коді безпосередньо з інтерфейсу середовища розробки. Ця функція дозволяє легко виконувати операції з комітами, об’єднанням змін і відстеженням історії проєкту, що

робить розробку більш організованою та контрольованою [6-9].

Таким чином, PyCharm є універсальним і гнучким інструментом для професійних розробників, які працюють з Python. Завдяки розширеним можливостям аналізу коду, глибокій інтеграції з інструментами контролю версій та зручним засобам налагодження, PyCharm дозволяє ефективно розробляти високоякісні програмні продукти, оптимізуючи робочий процес і підвищуючи продуктивність розробників.

Ураховуючи усе вище описане розробка застосунку здійснювалася за допомогою мови програмування Python, яка забезпечує гнучкість, масштабованість та підтримку великої кількості бібліотек. Основним середовищем розробки є PyCharm, що дозволяє ефективно працювати з проєктом завдяки вбудованим інструментам для налагодження та тестування коду. Інтеграція з Telegram API здійснюється за допомогою бібліотеки Telethon, яка надає широкий набір функцій для роботи з чатами, повідомленнями та користувачами.

У розробці застосовано асинхронне програмування, що дозволяє ефективно обробляти запити та зменшувати затримки під час взаємодії з сервером. Також використовується об’єктно-орієнтований підхід, який дозволяє структурувати код, полегшуючи його підтримку та масштабування.

Для збереження отриманих даних застосовуються файлові операції, де кожен чат записується у окремий текстовий файл, що містить базову інформацію про чат та останні 50 повідомлень. Реалізовано механізми автоматичного створення файлів і папок, а також авто-видалення для оптимального керування ресурсами сховища.

2.3. Структура та модулі застосунку

Структура програмного застосунку для обробки даних з месенджера Telegram організована таким чином, щоб забезпечити ефективну взаємодію між ключовими компонентами, включаючи клієнт Telegram, базу даних та зовнішні сервіси, такі як Google Sheets. Основними модулями цього застосунку є модуль обробки повідомлень з Telegram, модуль роботи з базою даних та модуль інтеграції з Google Sheets. Кожен з цих модулів виконує певні функції, які в сукупності забезпечують повний цикл обробки даних.

Модуль взаємодії з Telegram відповідає за підключення до облікового запису користувача через API Telegram за допомогою бібліотеки Telethon. Він забезпечує автентифікацію за допомогою API-ключів, отримання повідомлень із заданих чатів та збереження їх у структурованому вигляді. Модуль використовує асинхронні методи для підвищення продуктивності, що дозволяє обробляти великі обсяги повідомлень без блокування виконання інших операцій. Додатково цей модуль включає механізми обробки та трансформації отриманих даних для подальшого використання.

Модуль роботи з базою даних відповідає за збереження та вибірку інформації про чати та повідомлення. Він реалізований на основі використання бібліотеки pymysql для підключення до MySQL бази даних через SSH-тунель, що забезпечує безпечну передачу даних. Цей модуль включає функціональність для створення підключення, виконання SQL-запитів на вставку та вибірку даних, а також закриття підключення після завершення операцій. Передбачена обробка можливих винятків, пов’язаних із доступністю бази даних або помилками під час виконання запитів, що підвищує надійність роботи застосунку.

Модуль інтеграції з Google Sheets забезпечує передачу отриманих з Telegram даних у вигляді електронних таблиць за допомогою API Google Sheets. Цей модуль реалізує механізми аутентифікації через OAuth 2.0, використовуючи збережені облікові дані для доступу до таблиць. Дані, отримані з бази даних, форматуються відповідно до вимог API Google Sheets,

після чого відправляються у вказаний діапазон таблиці. Реалізована можливість збереження токенів авторизації для подальшого використання, що дозволяє уникнути повторної аутентифікації та покращує швидкість виконання операцій.

Архітектура застосунку базується на розділенні логіки між зазначеними модулями, що дозволяє легко масштабувати та розширювати функціональність при необхідності інтеграції з новими сервісами або додавання додаткових обробок даних. Взаємодія між модулями здійснюється через чітко визначені інтерфейси, що спрощує тестування та підтримку коду. Також застосунок включає механізми логування та обробки помилок, що дозволяє вчасно виявляти та усувати потенційні збої у роботі.

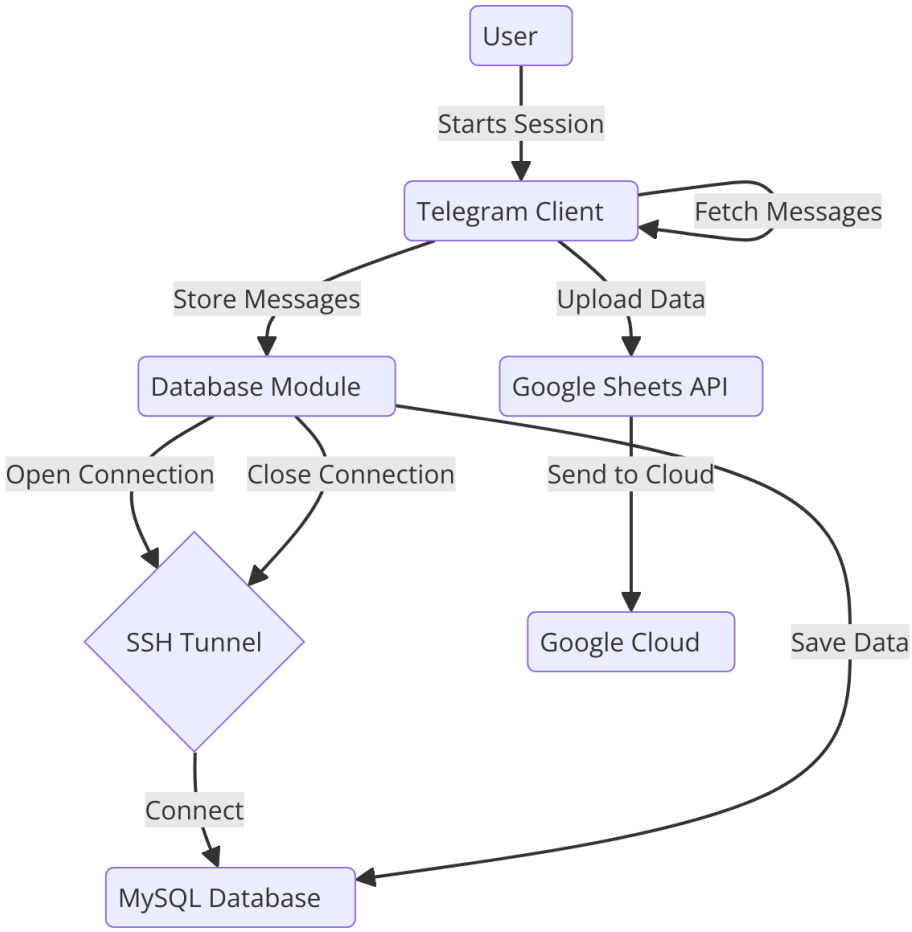

Рисунок 2.5. Структура та модулі застосунку

Таким чином, загальна структура застосунку забезпечує високий рівень

модульності, що сприяє гнучкості та надійності системи. Оптимізація процесів взаємодії між модулями дозволяє ефективно обробляти великі обсяги інформації, зберігаючи продуктивність та мінімізуючи час обробки даних.

2.4. Опис методології створення програмного забезпечення застосунку

Для роботи із ПЗ застосунку для керування та аналізу повідомлень у месенджері Telegram було обрано Agile модель проєктування, яка відома своєю гнучкістю та здатністю адаптуватися до змін протягом усього процесу розробки. Agile підхід дозволяє швидко реагувати на нові вимоги та ітеративно вдосконалювати продукт.

Методологія Agile – це гнучкий підхід до розробки програмного забезпечення, що акцентує увагу на ітеративному нарощуванні функціоналу, швидкому отриманні зворотного зв’язку від замовників та здатності оперативно реагувати на зміни. Основу Agile складають короткі цикли розробки, які називають ітераціями або спринтами, протягом яких команда отримує працюючий прототип, що постійно вдосконалюється та доповнюється новими функціями. Цей процес передбачає тісну комунікацію між усіма учасниками проєкту, часті перевірки результатів та наступну корекцію курсу, якщо вимоги чи пріоритети змінюються [10].

Такі принципи, як взаємодія людей важливіша за процеси та інструменти, робоче програмне забезпечення важливіше за детальну документацію, співпраця із замовником важливіша за умови контракту та готовність до змін важливіша за дотримання плану, закріплено в «Маніфесті Agile». Завдяки їм команди можуть досягати високої швидкості розробки без втрати якості продукту, уникаючи надмірної формалізації, складних бюрократичних процедур та затяжного погодження специфікацій [10].

Ось основні принципи та підходи Agile, які ми застосували у своєму проекті:

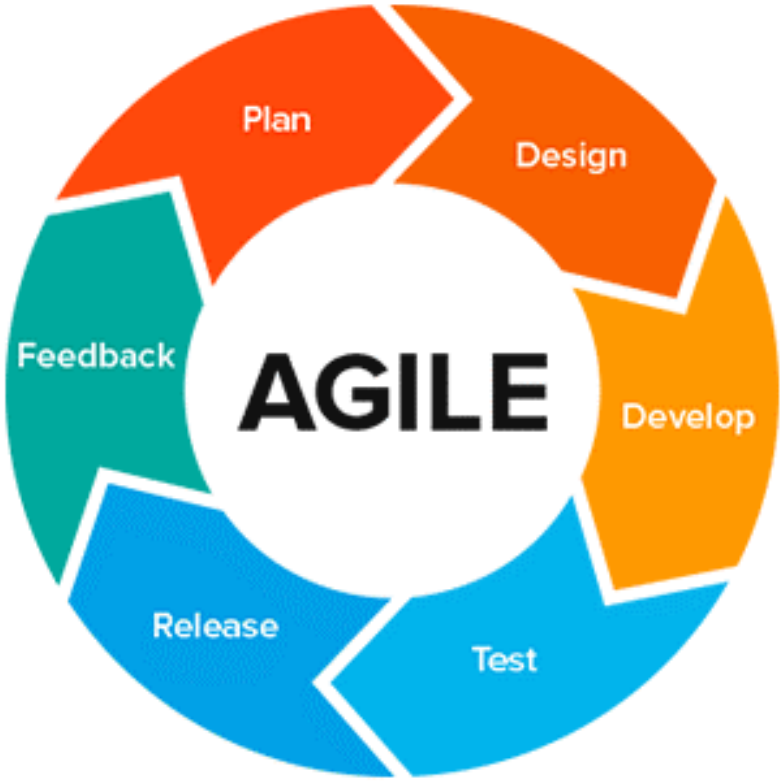

Рисунок 2.6. Методологія Agile

У межах розробки застосунку, що використовує Telethon для взаємодії з Telegram та MySQL для зберігання даних, застосовано гнучкий підхід управління проєктом, притаманний методології Agile. Основна ідея цієї методології полягала в ітеративному та інкрементному нарощуванні функціональності: код поділявся на невеликі етапи, кожен із яких мав сформувати певний працюючий функціонал, що можна було перевірити й за потреби змінити або розширити вже на наступній ітерації.

Насамперед було визначено центральні вимоги, зокрема можливість встановлення захищеного з’єднання з базою даних через SSH-тунель і ініціація підключень до Telegram з метою збирання й обробки повідомлень та запису отриманих даних у MySQL. На початкових ітераціях розробник приділив увагу реалізації базових механізмів, а саме створив клас DataBaseUseCase, що відповідає за відкриття й закриття тунелю та транзакційні операції з базою, і перевірив стабільність цього модуля під час запуску в різних середовищах. На наступній ітерації було впроваджено клас TelegramParseChat, у якому реалізовано послідовну логіку збирання повідомлень і чатів, а також передбачено їх запис через методи DataBaseUseCase. Така поетапність надала можливість оперативно протестувати окремі функції та переконатися, що вони задовольняють визначені вимоги, перш ніж переходити до розширення функціоналу.

Коли критичний функціонал отримання та зберігання даних виявився працездатним, наступні ітерації присвячувалися інтеграції з Google Sheets через виклик update_google_sheets_. На цьому етапі було створено окремі функції форматування даних та їхньої передачі до сервісу. Завдяки Agile-підходу з’явилася можливість швидко реагувати на зворотний зв’язок користувачів, зокрема вносити корективи до структури таблиць або механізмів авторизації в Google API, не порушуючи при цьому раніше розроблені модулі. У такий спосіб код розвивався зі збереженням сумісності з уже існуючими частинами системи, адже методи на кшталт _insert_message та _insert_chat лишалися стабільними, тоді як бізнес-логіку аналізу й експорту даних адаптували під нові сценарії.

Реалізація принципів Agile також простежується у постійному залученні коротких «циклів зворотного зв’язку», коли результати роботи скрипта відразу тестувалися в реальному середовищі (наприклад, на тестовому обліковому записі Telegram або в аркуші Google Sheets), після чого приймалися рішення щодо пріоритетності наступних завдань. У ході кожної ітерації проводилося так

зване «рефакторинг-завершення», коли окремі процедури — наприклад, авторизація через token.pickle чи логування винятків у методах асинхронного збирання повідомлень - уточнювалися й оптимізувалися на підставі отриманих результатів. Усе це відповідало ідеї безперервного вдосконалення, властивій гнучким методологіям.

Такий процес дав змогу переходити від мінімально працюючого продукту – скрипта, що просто встановлює SSH-тунель і звертається до Telegram API, - до системи, яка передбачає повноцінну інтеграцію з кількома зовнішніми сервісами й надає можливість масштабувати або модифікувати будь-який із її компонентів. Завдяки поступовому нарощуванню функціоналу, регулярній перевірці результатів і тісному зворотному зв’язку, розробник забезпечив стабільність проміжних версій коду й уникнув накопичення технічного боргу на пізніх стадіях. У підсумку це дозволило в рамках коротких спринтів ітеративно доопрацювати програми, досягнути відповідності технічним і функціональним вимогам, а також легко інтегрувати нові ідеї та зміни в уже розгорнуту систему.

РОЗДІЛ 3. РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ

3.1. Розробка та опис діаграми послідовностей застосунку

Діаграма послідовностей відображає процес обробки даних із месенджера Telegram із подальшим збереженням їх у базі даних та передачею до Google Sheets. Взаємодія здійснюється між чотирма основними учасниками: користувачем (User), клієнтом Telegram (TelegramClient), базою даних (DataBaseUseCase) та сервісом Google Sheets (GoogleSheetsAPI). Нижче наведено докладний опис етапів цього процесу.

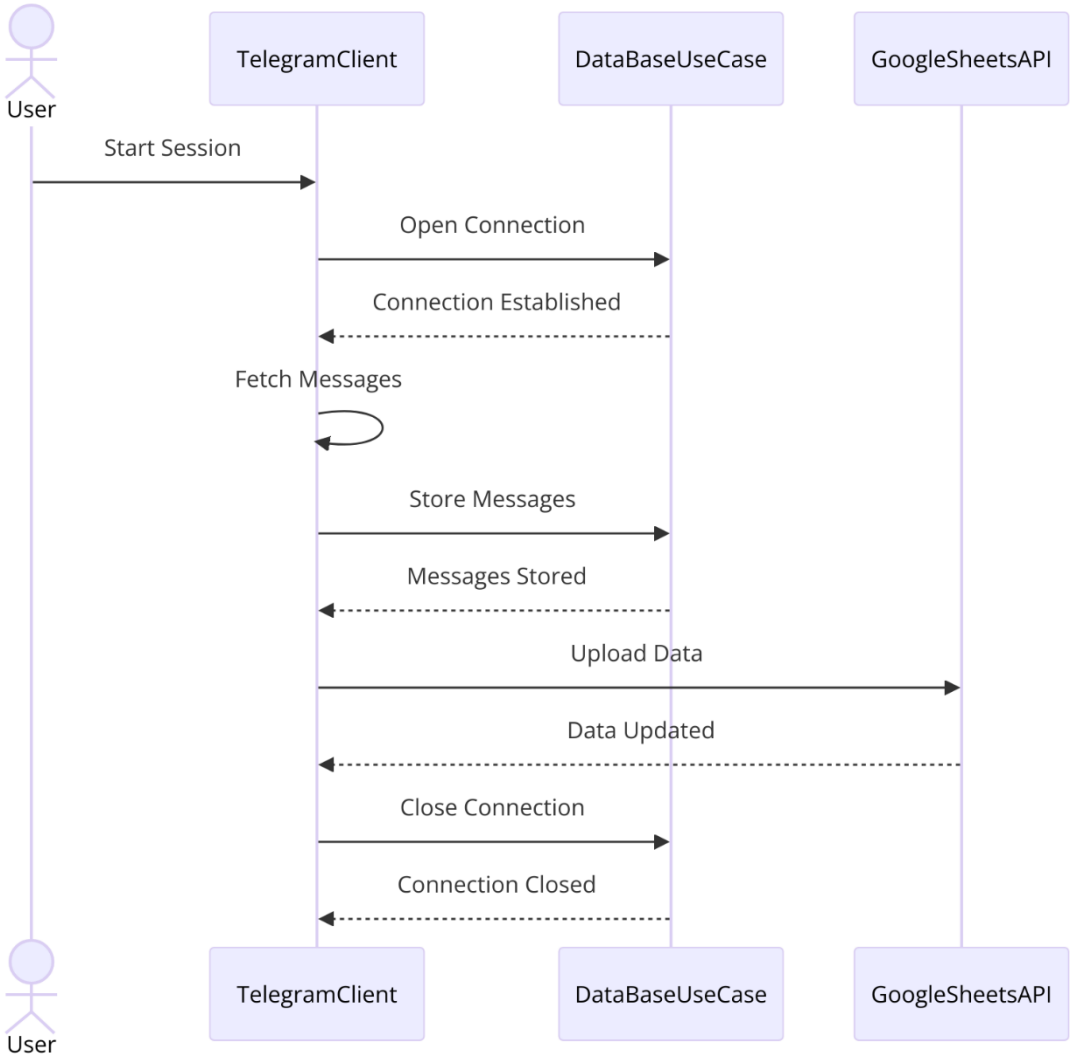

Рисунок 3.1. Діаграма послідовностей обробки даних Telegram

1. Ініціація сесії користувачем.

Процес починається із запиту користувача на підключення до сервісу. Учасник User надсилає запит до TelegramClient, ініціюючи початок роботи клієнта Telegram шляхом виконання операції Start Session. Ця операція передбачає автентифікацію користувача та підготовку клієнта до роботи із сервісом Telegram.

2. Встановлення підключення до бази даних.

Після успішного запуску сесії клієнт Telegram надсилає запит на встановлення з'єднання до бази даних через клас DataBaseUseCase, викликаючи метод Open Connection. У відповідь база даних підтверджує встановлення з'єднання, передаючи відповідь Connection Established. На цьому етапі клієнт отримує можливість виконувати операції з базою даних, такі як збереження та вибірка інформації.

3. Отримання повідомлень із Telegram.

Наступний крок передбачає збирання інформації про повідомлення з каналів або чатів Telegram. TelegramClient виконує операцію Fetch Messages, отримуючи доступ до відповідних чатів, та здійснює вибірку повідомлень із зазначеними критеріями, такими як кількість повідомлень та часовий діапазон.

4. Збереження повідомлень у базі даних.

Отримані повідомлення передаються на збереження до бази даних шляхом виклику функції Store Messages. Клієнт Telegram передає інформацію про повідомлення у структурованому вигляді, яка включає унікальні ідентифікатори, час відправлення, текст повідомлення тощо. У відповідь від DataBaseUseCase надходить підтвердження успішного збереження даних (Messages Stored).

5. Оновлення даних у Google Sheets.

Після успішного збереження даних у базі вони передаються до хмарного сервісу Google Sheets для подальшого аналізу та обробки. TelegramClient

надсилає запит до GoogleSheetsAPI, ініціюючи оновлення даних через виклик операції Upload Data. У відповідь сервіс Google Sheets повертає підтвердження про успішне оновлення (Data Updated), що вказує на те, що дані були успішно збережені у таблиці.

6. Закриття підключення до бази даних.

Останнім кроком є завершення роботи клієнта Telegram та закриття підключення до бази даних для забезпечення безпеки та ефективного використання ресурсів. TelegramClient викликає операцію Close Connection, після чого DataBaseUseCase підтверджує успішне завершення підключення (Connection Closed).

Розглянута діаграма послідовностей ілюструє логічний ланцюг обробки даних із Telegram, починаючи від ініціації користувачем сеансу, збору та збереження повідомлень, до їхньої обробки і відправлення у Google Sheets. Така архітектура дозволяє ефективно обробляти великі обсяги даних та забезпечувати їхню доступність для подальшого аналізу.

3.2. Інтеграція з API Telegram

Процес інтеграції даного програмного застосунку з API Telegram здійснюється через використання асинхронної бібліотеки Telethon, яка забезпечує ефективну взаємодію з месенджером Telegram на рівні протоколу Telegram MTProto. Інтеграція починається з ініціалізації клієнта Telegram, що передбачає автентифікацію за допомогою унікальних ідентифікаторів API, таких як api_id та api_hash, які отримуються з особистого облікового запису користувача на платформі Telegram. На етапі автентифікації застосунок ініціює запит до серверів Telegram для отримання тимчасового токена доступу, який дозволяє виконувати операції з акаунтом користувача. У випадку, якщо облікові дані були використані раніше, застосунок використовує збережену

сесію, що значно зменшує час на повторну автентифікацію та забезпечує безперервність роботи.

Після успішного встановлення з’єднання клієнт Telegram починає отримання даних із месенджера шляхом виклику відповідних методів API, зокрема для отримання списку чатів та повідомлень. Застосунок використовує метод get_dialogs для отримання інформації про наявні чати, а для витягування повідомлень використовується метод get_messages, який дозволяє виконувати фільтрацію за різними критеріями, такими як кількість повідомлень, часовий проміжок та тип повідомлень. Отримані дані надалі обробляються та нормалізуються для подальшого збереження у базі даних. З метою мінімізації затримок та забезпечення ефективності, операції отримання даних виконуються асинхронно, що дозволяє застосунку паралельно обробляти інші завдання, такі як передача даних до Google Sheets або здійснення додаткових запитів.

Обробка отриманих повідомлень передбачає їхню структуризацію відповідно до заданих моделей, які включають ключові атрибути, такі як ідентифікатор повідомлення, ідентифікатор відправника, дата та текст повідомлення. Після обробки застосунок викликає методи для збереження цих даних у реляційній базі даних через попередньо встановлений SSH-тунель. Для забезпечення збереження цілісності даних перед виконанням операцій запису відбувається перевірка наявності записів у базі, що дозволяє уникнути дублювання інформації. Після завершення процесу обробки повідомлень клієнт Telegram закриває підключення до сервера, а сесія зберігається для подальшого використання.

Одним із важливих аспектів інтеграції є обробка можливих винятків та помилок, пов’язаних із мережею або обмеженнями, що накладаються API Telegram. Для цього застосунок використовує механізми повторного виконання запитів у разі виникнення тимчасових помилок та ведення журналу помилок для подальшого аналізу. Крім того, інтеграція передбачає обмеження на кількість запитів, що надсилаються до API Telegram, шляхом дотримання

встановлених Telegram обмежень на швидкість запитів, щоб уникнути блокування облікового запису.

Таким чином, процес інтеграції з API Telegram в розробленому нами застосунку реалізований з урахуванням принципів ефективності, безпеки та надійності, що дозволяє здійснювати стабільний обмін інформацією між клієнтом та сервером, забезпечуючи можливість масштабування та розширення функціональності у майбутньому.

3.3. Програмна реалізація функціональних модулів застосунку

Реалізація функціональних модулів у наведеній програмі ґрунтується на чіткому розмежуванні логіки взаємодії з базою даних, сервісом Telegram та Google Sheets, а також на застосуванні допоміжних сутностей, що спрощують відображення інформації та здійснення CRUD-операцій. Фрагмент коду, де описується зв’язок із базою даних, містить клас DataBaseUseCase, який забезпечує встановлення та закриття з’єднання з MySQL через SSH-тунель, а також містить методи для вставки й вибірки даних. Зокрема, у конструкторі цього класу ініціалізуються закриті поля self.__tunel і self.__connection, які відповідають за стан тунелю та підключення. Метод open_connection здійснює створення екземпляра SSHTunnelForwarder, де вказується віддалений SSH-хост, логін та пароль, а також відкриває безпосереднє з’єднання з MySQL шляхом виклику pymysql.connect. Як приклад, у методі _insert_message можна побачити виконання SQL-запиту INSERT, що записує ідентифікатор повідомлення, інформацію про відправника та текстове поле message:

_command = "INSERT INTO `messages` (`id_message`, `id_sender`, `date_text`, `date_`, `text_message`, `type_chat`) VALUES (%s, %s, %s, %s, %s, %s);"

cursor.execute(_command, (model.id_message, model.id_sender_message, model.date_message, model.date_message, model.text_message, type_chat))

Завдяки такому підходу вся операційна логіка для роботи з даними залишається інкапсульованою в межах одного класу, який забезпечує виконання транзакцій і закриття тунелю через метод close_connection.

Функціонал, що відповідає за отримання повідомлень і списку чатів із Telegram, реалізовано в класі TelegramParseChat. У конструкторі ініціалізується об’єкт TelegramClient, де через параметри api_id та api_hash налаштовується робота з Telegram API. Методи на кшталт get_all_message(limit_message, type_chat) звертаються до Telethon для отримання діалогів через await self.client.get_dialogs(), після чого для кожного діалогу викликають get_messages, щоб вибрати обмежену кількість повідомлень. Під час обробки кожне повідомлення приводиться до інстансу класу MessageModel із полями id_message, id_sender_message, date_message та text_message. Нижче наведено приклад, як виконується створення моделі повідомлення:

```
model = MessageModel(
    id_message=message.id,
    id_sender=message.peer_id.user_id,
    date_=message.date,
    text_message=message.message
)
```

Після цього викликається метод _insert_message зі стороннього об’єкта DataBaseUseCase, що дозволяє зберегти зміст повідомлення до бази даних.

Модуль взаємодії з Google Sheets зосереджено у функціях update_google_sheets_ і format_data_for_sheets, де перша функція забезпечує безпосереднє відправлення даних у вибраний діапазон робочої книги, а друга — попереднє форматування, яке додає заголовки та конвертує значення datetime у рядковий формат. Взаємодія з Google API відбувається за допомогою авторизаційного механізму, в межах якого після успішного зчитування токена з

файлу token.pickle створюється клієнтське під’єднання до Sheets API:

```
service = build(‘sheets’, ‘v4’, credentials=creds)
sheet = service.spreadsheets().values().update(
   spreadsheetId=spreadsheet_id,
   range=range_name,
   valueInputOption=‘RAW’,
   body=body
).execute()
```

Таким чином, уся процедура оновлення реалізується асинхронно від решти компонентів, а з’єднання з базою даних у цей момент уже ініційоване зовнішнім кодом. Завершальним кроком виконується виклик функції upload_data_google_sheet, яка відкриває з’єднання з базою, завантажує з неї дані за допомогою _select_all_chats_by_type(type_chat) і передає їх у Google Sheets, після чого з’єднання закривається. У блоці if name == ‘main’: уся логіка збирання та відправлення інформації виконується автоматично шляхом виклику відповідного методу, що вказує конкретні ідентифікатори таблиці й аркуша, а також типу чату, для якого здійснюється вибірка. Така структуризація коду забезпечує зрозумілу сегментацію логіки, дозволяє легко доповнювати застосунок новими можливостями й робить модульним сам процес розробки та налагодження.

3.4. Реалізація алгоритму аналізу повідомлень у чатах Telegram

Реалізація алгоритму аналізу повідомлень у розглянутій програмі базується на використанні асинхронного підходу з використанням бібліотеки Telethon та відповідного класу TelegramParseChat, де передбачено метод get_all_message. У цьому методі відбувається отримання списку діалогів через виклик await self.client.get_dialogs(), що повертає інформацію про всі доступні

канали, групи та приватні чати, пов’язані з обліковим записом, під яким було ініціалізовано сесію. Для кожного виявленого діалогу здійснюється додатковий виклик await self.client.get_messages(PeerChannel(chat.id), limit=limit_message), який ініціює вибірку обмеженої кількості повідомлень за вказаною кількістю limit_message.

Після отримання чергового повідомлення виконується перетворення його структурованих даних із формату, прийнятого Telethon, до внутрішнього представлення в межах об’єкта класу MessageModel. Конструктор MessageModel приймає ідентифікатор повідомлення, ідентифікатор відправника, часову мітку відправлення та власне текст повідомлення. Важливим моментом є збереження цієї інформації в базі даних, що виконується за допомогою виклику _insert_message, який належить до класу DataBaseUseCase. У цьому методі формується SQL-запит на кшталт INSERT INTO messages (...), у який передаються відповідні аргументи з моделі, після чого виконується cursor.execute, а результати фіксуються у межах транзакції шляхом self.__connection.commit().

Для більшої керованості процесом обробки повідомлень у коді передбачено окремий виняток для кожного кроку аналізу, що дає змогу відстежувати ймовірні помилки у випадку, коли певний діалог недоступний або коли сама бібліотека Telethon генерує виключення. Якщо метод _insert_message повертає успішний результат, у консолі друкується відповідне повідомлення з ідентифікатором вставленого запису, а також відображається вміст тексту, що дає можливість відстежувати поточний статус обробки. Кількісне обмеження виклику get_messages обумовлене параметром limit_message, який задає граничну кількість повідомлень для обробки в одному циклі, завдяки чому забезпечується контрольована інтенсивність аналізу та запобігання надмірному навантаженню або перевищенню квот з боку Telegram API. У разі потреби метод можна викликати з більшим або меншим значенням, залежно від мети збирання даних. Відповідно, алгоритм аналізу зводиться до послідовного

перебору діалогів, отримання повідомлень, формування внутрішніх моделей і їх запису в базу даних, що забезпечує систематизоване збирання й первинну підготовку даних для подальшого використання.

3.5. Побудова діаграми класів розробленого ПЗ застосунку

Діаграма класів відображає структуру програмного застосунку, що забезпечує інтеграцію з месенджером Telegram, управління базою даних та експорт інформації до сервісу Google Sheets. Основними класами, які формують архітектуру застосунку, є TelegramParseChat, DataBaseUseCase, GoogleSheetsUseCase, а також моделі MessageModel та ChatModel, що використовуються для представлення даних. Взаємозв’язки між класами забезпечують чітке розділення функціональності та сприяють підвищенню модульності застосунку.

Клас TelegramParseChat є центральним елементом, що відповідає за взаємодію з API Telegram. Він містить екземпляр клієнта Telegram, представлений атрибутом client, який ініціалізується за допомогою бібліотеки Telethon шляхом використання параметрів автентифікації api_id та api_hash. У класі реалізовані методи start_client(), який ініціює підключення до облікового запису, та get_all_message(limit_message: int, type_chat: str), що відповідає за отримання повідомлень з певного чату. Зокрема, метод здійснює виклик функції get_messages, яка повертає список об’єктів повідомлень, що далі передаються до бази даних. Метод get_all_chats(type_chat: str) забезпечує отримання списку доступних чатів із подальшою обробкою та збереженням інформації.

Клас DataBaseUseCase реалізує логіку роботи із реляційною базою даних MySQL через безпечне підключення за допомогою SSH-тунелю. Внутрішня структура класу включає атрибути __tunel та __connection, які представляють

відповідно об’єкт SSH-тунелю та з’єднання з базою даних, ініціалізовані в методі open_connection(). Функціональність класу передбачає виконання операцій додавання та вибірки даних. Метод _insert_message(model, type_chat) виконує SQL-запит на вставку нового повідомлення у таблицю messages, використовуючи параметри, передані у вигляді екземпляра класу MessageModel.

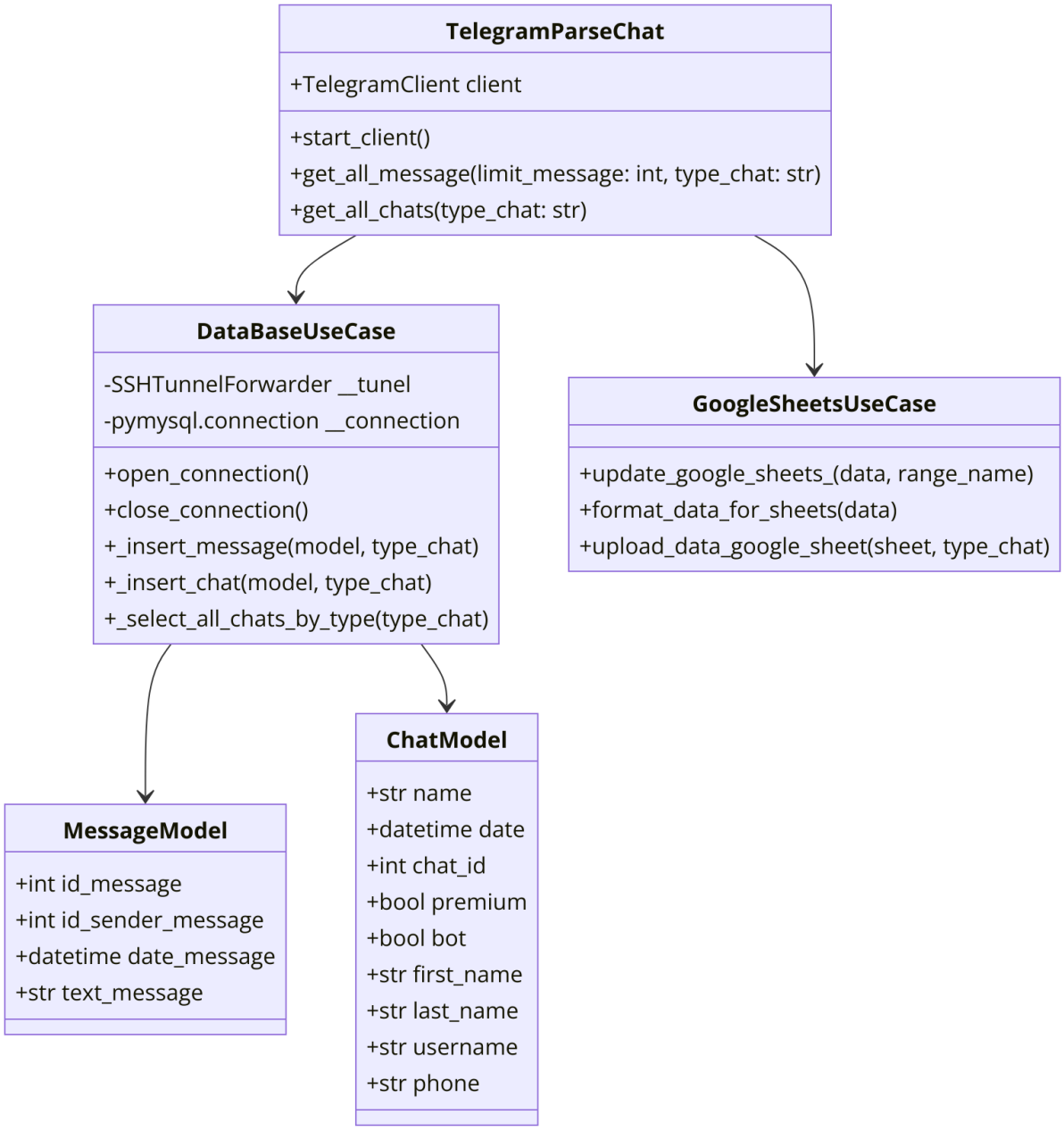

Рисунок 3.2. Діаграма класів розробленого ПЗ застосунку

Аналогічно, метод _insert_chat(model, type_chat) здійснює запис інформації про чати в таблицю chats, використовуючи об’єкт класу ChatModel. Функція _select_all_chats_by_type(type_chat) забезпечує вибірку даних з таблиці на основі переданого типу чату.

Клас GoogleSheetsUseCase є відповідальним за інтеграцію із сервісом Google Sheets та містить методи для форматування та експорту даних. Основним методом класу є update_google_sheets_(data, range_name), який використовує API Google Sheets для оновлення діапазону комірок таблиці переданими даними. Форматування здійснюється за допомогою функції format_data_for_sheets(data), яка перетворює отримані записи у формат, що відповідає вимогам Google Sheets API. Процес експорту відбувається шляхом виклику методу upload_data_google_sheet(sheet, type_chat), що отримує дані із бази за допомогою класу DataBaseUseCase та передає їх у хмарний сервіс.

Дві допоміжні моделі, представлені класами MessageModel та ChatModel, використовуються для інкапсуляції даних, отриманих із Telegram. Клас MessageModel містить атрибути, що характеризують повідомлення, такі як ідентифікатор повідомлення id_message, ідентифікатор відправника id_sender_message, дата надсилання date_message та текст повідомлення text_message. Клас ChatModel визначає атрибути, пов’язані з інформацією про чати, включаючи ім’я чату name, дату створення date, унікальний ідентифікатор чату chat_id, а також додаткові атрибути, такі як статус преміум-акаунта, наявність бота та контактні дані.

Взаємодія між класами побудована на основі композиції та агрегації. Клас TelegramParseChat використовує функціональність класу DataBaseUseCase для збереження даних та класу GoogleSheetsUseCase для експорту інформації. У свою чергу, клас DataBaseUseCase взаємодіє з моделями MessageModel та ChatModel для обробки структурованих даних перед їхнім записом у базу. Така структура застосунку забезпечує розділення відповідальності між модулями, сприяє легкості модифікації та розширюваності функціоналу в майбутньому.

РОЗДІЛ 4. АНАЛІЗ ТА ОЦІНКА ЕФЕКТИВНОСТІ ЗАСТОСУНКУ

4.1. Порівняльний аналіз із існуючими рішеннями

У контексті порівняння розробленого застосунку з існуючими рішеннями, що забезпечують аналіз і керування повідомленнями у месенджерах, варто звернути увагу на кілька важливих параметрів, серед яких функціональне охоплення, гнучкість налаштувань, можливості розширення та підтримка інтеграцій. Насамперед застосунок, побудований із використанням Telethon і MySQL, є прикладом «легкої» індивідуальної розробки, коли вся логіка зберігання та отримання даних про повідомлення залишаються під повним контролем користувача. У цьому випадку реалізовано досить вузькоспеціалізовані функції: завантаження списку діалогів із Telegram, зчитування їхніх повідомлень і зберігання структурованих записів у базі даних, а також подальше відправлення зібраної інформації до Google Sheets, що забезпечує гнучкий механізм для її перегляду та аналізу.

Існують готові програмні рішення, що пропонують віддалений збір повідомлень у месенджерах разом із розширеними опціями аналізу, наприклад платформи моніторингу соціальних мереж, які інтегруються з Telegram на рівні ботів або API. Деякі з них спеціалізуються на аналізі тональності, виявленні ключових слів чи обчисленні статистичних метрик і можуть пропонувати користувачеві зручний вебінтерфейс та інструменти візуалізації даних. Комерційні системи цього типу здебільшого сфокусовані на обробці великих обсягів інформації й надають засоби для зберігання історії чатів у «хмарі», що зручніше для командного доступу, але водночас вимагає підписки й може мати обмеження стосовно налаштування та власноручного розміщення. Типовий підхід подібних сервісів полягає у використанні ботів, які залучаються безпосередньо до каналів і груп, а зібрані повідомлення в реальному часі потрапляють до хмарної платформи, де й зберігаються або аналізуються

автоматичними модулями.

Ще однією поширеною категорією програм є скрипти на основі Telegram CLI або офіційного Telegram Bot API, що дозволяють експортувати історію повідомлень локально, часто у форматі JSON чи CSV, для подальшої аналітики традиційними засобами обробки даних. Подібні рішення не завжди передбачають підтримку асинхронності чи розвиненої взаємодії з базою даних, проте вони є простішими у налаштуванні й орієнтованими на швидке отримання резервної копії чатів. Такий підхід може бути корисним, якщо першочерговим завданням є саме архівація даних, але він менш гнучкий для оперативної інтеграції з іншими сервісами на кшталт Google Sheets або для розширеного аналізу у режимі реального часу.

Застосунок, створений із використанням DataBaseUseCase і TelegramParseChat, виграє в тому, що допускає повноцінне керування всіма етапами: від процесу аутентифікації до формування власних запитів у базі даних, а також відкриття SSH-тунелю для безпечного доступу до MySQL, при цьому не покладається на сторонні обчислювальні сервіси чи закриті API з обмеженими можливостями. Така автономія дає змогу гнучко масштабувати рішення й доповнювати його власними модулями аналізу, наприклад інтегрувати кластеризацію або моделі обробки природної мови у вже наявний код. Однак за відсутності вбудованих алгоритмів аналізу чи розширених сценаріїв взаємодії з користувачами це рішення поступається розвинутим комерційним платформам або масовим сервісам аналітики, які пропонують візуалізацію, інструменти налаштування звітів та додаткові метрики. Головними перевагами індивідуальної розробки залишаються висока гнучкість, можливість допасування до специфічних бізнес-вимог та повний контроль над даними, тоді як готові платформи надають більш швидкий старт, але водночас містять статичний набір функцій і залежать від ліцензійних угод постачальників.

Таким чином, розроблений нами застосунок можна розглядати як основу

системи управління повідомленнями Telegram із наголосом на безпечному з’єднанні з базою даних, простоті розгортання й налаштуванні інтеграції зі сторонніми сервісами типу Google Sheets. Порівняно з уже існуючими масштабними інструментами, що спеціалізуються на комплексній аналітиці, алгоритмах кластеризації чи роботі з тональністю повідомлень, воно дозволяє глибше кастомізувати збір і використання даних і виконує функцію базового сховища та інтеграційної ланки для подальших розрахунків. У такий спосіб воно є привабливою опцією для розробників, яким важливо зберегти контроль над процесом і максимально налаштувати сценарії взаємодії з повідомленнями та іншими ресурсами.

4.2. Інструкція користувача та функціональні можливості застосунку

Розглянемо поетапно процес запуску розробленого програмного застосунку для керування повідомленнями у месендері Telegram та результати виконання програми.

Щоб запустити програму користувачеві, який буде використовувати застосунок, необхідно виконати наступні дії:

1. Завантажити всі програмні файли в одну папку на ПК. Перелік файлів представлено на рисунку 4.1.

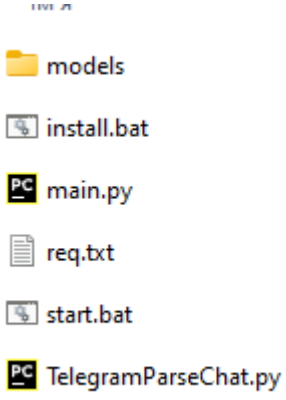

Рисунок 4.1. Файли програмного застосунку

2. Зареєструвати додаток в телеграмі за посиланням https://my.telegram.org/apps. Процес реєстрації представлено на рисунку 4.2.

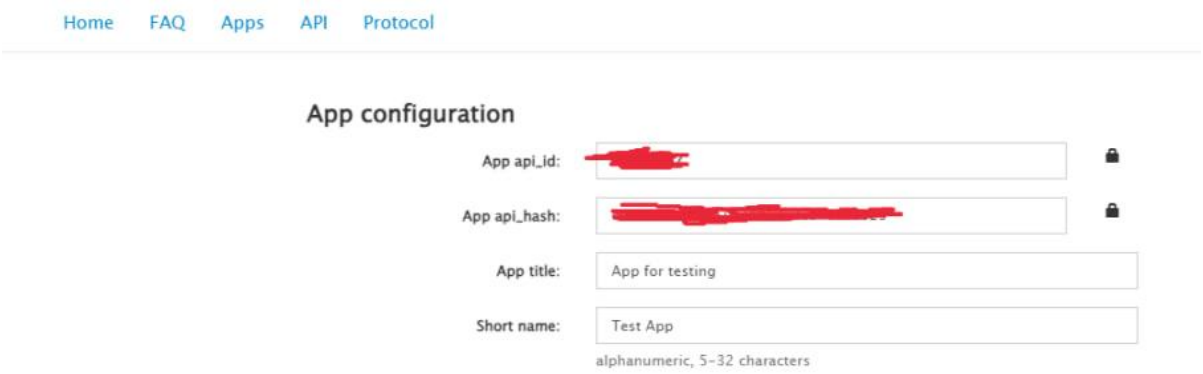

Рисунок 4.2. Процес реєстрації програмного застосунку у Telegram

3. Вимкнути двохетапну аунтифікацію (див. Рисунок 4.3.)

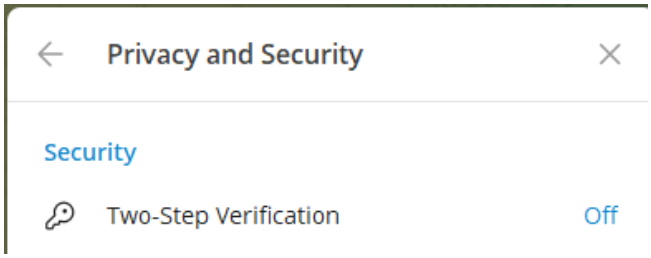

Рисунок 4.3. Налаштування аутентифікації

4. Встановити python за посиланням https://www.python.org/downloads/.

5. Запустити файл install.bat та дочекатися встановлення і автоматичного закриття консолі (один раз), з’явиться папка venv.

6. Запускати start.bat кожен раз коли потрібно запустити програму.

Тут вводимо api_id та api_hash з https://my.telegram.org/apps

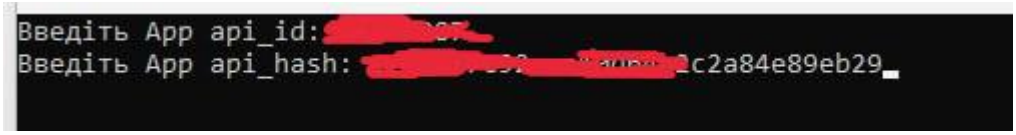

Рисунок 4.4. Процес введення api_id та api_hash

7. Далі вводимо телефон у форматі +380 та код який прийде на цей номер телеграму

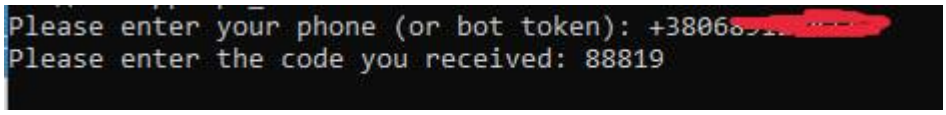

Рисунок 4.5. Введення номера телефону та коду безпеки під час роботи застосунку

Після реєстрації та правильного запуску програми застосунок почне почне витягувати чати, весь процес ви зможете побачити у консолі, при необхідності, програма також повідомить про виникнення помилки.

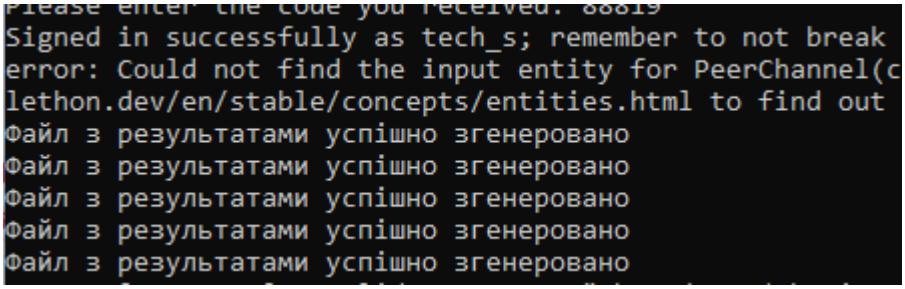

Рисунок 4.6. Процес запуску програми у консолі

Після виконання програми буде згенеровано декілька папок, представлених на рисунку 4.7.

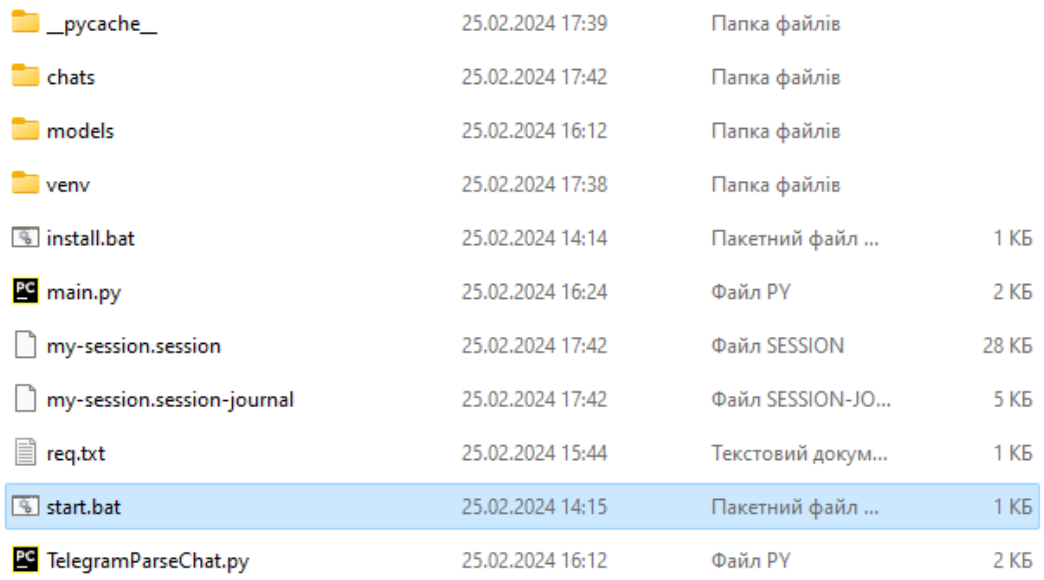

Рисунок 4.7. Результат парсингу чатів у Telegram

Для того щоб отримати доступ до певного чату відкриваємо папцу chats і бачимо всі наші чати.

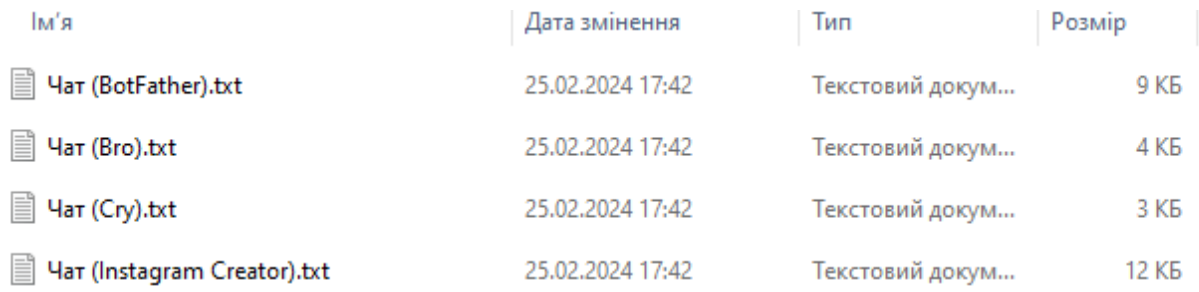

Рисунок 4.8. Список чатів отриманих із Telegram

Далі потрібно обрати необхідний вам чат і відкрити його.

Приклад вмісту файлу ви бачите на Рисунку 4.9.

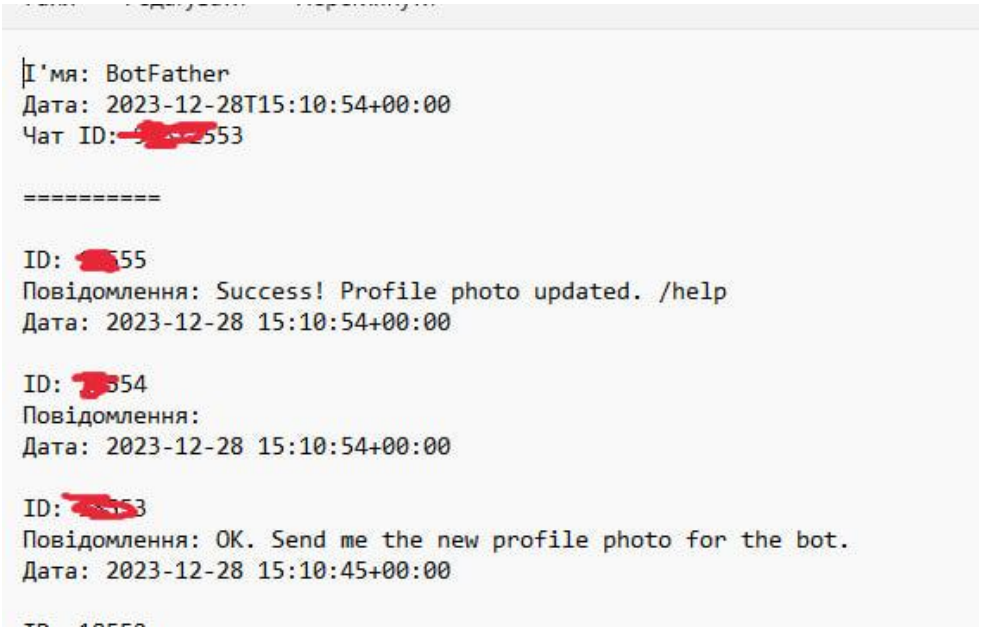

Рисунок 4.9. Приклад вмісту написаних повідомлень із обраного чату

Розроблений програмний застосунок для управління повідомленнями в Telegram забезпечує широкий спектр функціональних можливостей, спрямованих на автоматизацію процесів збору, обробки та зберігання комунікаційних даних. Основним завданням програмного забезпечення є інтеграція з Telegram API для отримання повідомлень з чатів і каналів, їх структуризація, а також збереження у реляційній базі даних з можливістю подальшого аналізу та експорту до зовнішніх сервісів, таких як Google Sheets.

Серед ключових функціональних можливостей застосунку можна виокремити автоматизоване підключення до Telegram за допомогою бібліотеки Telethon, що дозволяє отримувати дані в асинхронному режимі та обробляти великі обсяги інформації з мінімальними затримками. Реалізована функція аутентифікації через API-ключі забезпечує безпечний доступ до облікового запису користувача, а механізми управління сесіями дозволяють зберігати стан з’єднання та повторно використовувати його без необхідності повторної авторизації. У процесі роботи здійснюється отримання діалогів та повідомлень із зазначеною кількістю записів, що дозволяє оптимізувати завантаження даних відповідно до потреб користувача.

Програмний застосунок також передбачає модуль обробки повідомлень, у межах якого дані перетворюються у формат, зручний для збереження в базі даних. Відбувається структуризація отриманих повідомлень шляхом розбиття їх на ключові атрибути, такі як ідентифікатор повідомлення, відправник, час створення та текстове наповнення. Реалізовано механізм внесення цих даних до бази MySQL через захищене з'єднання, використовуючи SSH-тунель, що гарантує безпеку переданих даних і мінімізує ризики несанкціонованого доступу.

Значну увагу в застосунку приділено функціоналу інтеграції із зовнішніми сервісами. Розроблена система експорту отриманих даних у Google Sheets, що дає змогу здійснювати подальший аналіз комунікаційної активності за допомогою стандартних інструментів візуалізації та роботи з електронними таблицями. Додатково передбачено можливість автоматичного форматування перед експортом, включаючи перетворення часових міток та структурування повідомлень у табличному вигляді.

Важливою частиною застосунку є функціонал управління з'єднаннями та транзакціями в базі даних, що включає можливості встановлення та завершення сеансів роботи з MySQL, а також автоматичне управління помилками у разі виникнення збоїв у процесі запису чи отримання даних. Це дозволяє забезпечити надійність та стабільність роботи застосунку навіть при великих навантаженнях.

Таким чином, розроблене програмне забезпечення реалізує функціонал збору та обробки даних із Telegram, їх надійного збереження та подальшого експорту в зручний для користувача формат, що дозволяє ефективно управляти комунікаційними потоками, аналізувати отриману інформацію та автоматизувати робочі процеси, пов’язані з використанням месенджерів у бізнес- та організаційній діяльності.

ВИСНОВКИ

Актуальність розробки програмного забезпечення для управління повідомленнями у месенджерах, таких як Telegram, обумовлена зростаючою роллю цифрової комунікації в різних сферах діяльності, зокрема в бізнесі, освіті та державному управлінні. Сучасні компанії та організації все частіше використовують месенджери як основний канал комунікації зі своїми клієнтами та партнерами, що створює потребу в автоматизованих рішеннях для обробки великого обсягу інформації, структурованого зберігання даних та ефективного управління повідомленнями. Такі програмні продукти дозволяють систематизувати процеси взаємодії, підвищити рівень персоналізації обслуговування клієнтів та знизити операційні витрати за рахунок автоматизації рутинних завдань.

Іншою важливою складовою актуальності є необхідність аналітичної обробки даних, отриманих з месенджерів. В умовах високої конкуренції на ринку підприємства прагнуть отримати детальну інформацію про поведінку клієнтів, їхні уподобання та рівень залученості, що можливо завдяки впровадженню інструментів аналізу повідомлень. Крім того, стрімке зростання кількості користувачів месенджерів у всьому світі створює значні виклики щодо забезпечення конфіденційності та безпеки переданих даних. У цьому контексті важливою вимогою до розробки програмного забезпечення є використання сучасних методів шифрування та анонімізації даних, а також забезпечення надійного зберігання інформації з можливістю її подальшої обробки в захищених середовищах. Наявність подібного функціоналу дозволяє організаціям відповідати сучасним регуляторним вимогам та законодавчим нормам щодо захисту персональних даних.

Таким чином, розробка програмного забезпечення для парсингу та управління повідомленнями в месенджерах є актуальною через необхідність ефективного управління інформаційними потоками, підвищення рівня

аналітики комунікаційних процесів та забезпечення високого рівня безпеки даних. Впровадження таких рішень дозволяє не лише оптимізувати бізнес-процеси, а й отримати конкурентні переваги в умовах цифрової трансформації сучасного суспільства.

У ході виконання даного дослідження було розглянуто сучасні проблеми управління та аналізу повідомлень у месенджерах, а також розроблено програмний застосунок, що дозволяє ефективно вирішувати ці завдання. На основі проведеного аналізу сучасних рішень та визначення їхніх обмежень було сформульовано основні вимоги до розробки застосунку.

Розроблений застосунок дозволяє здійснювати автентифікацію користувачів за допомогою api_id, api_hash, номеру телефону та коду підтвердження. Забезпечено функцію зчитування списку всіх чатів користувача, включаючи особисті повідомлення, групові чати та канали. Дані кожного чату автоматично зберігаються у окремі текстові файли з відповідною назвою, що містять базову інформацію та останні 50 повідомлень.

Застосунок також реалізує функціонал автоматичного створення папок для збереження даних, а також їх авто-видалення відповідно до налаштувань користувача. Це дозволяє ефективно керувати обсягами збережених даних та уникнути перевантаження системи.

Використання мови програмування Python та середовища розробки PyCharm дозволило реалізувати ефективне рішення з використанням сучасних підходів, таких як асинхронне програмування, об’єктно-орієнтоване проєктування та інтеграція з Telegram API через бібліотеку Telethon.

Підсумовуючи, запропоноване рішення демонструє високу ефективність та зручність використання для організації та аналізу повідомлень у месенджерах. Подальший розвиток проєкту може включати розширення підтримки інших месенджерів, впровадження додаткових аналітичних інструментів та підвищення рівня безпеки обробки даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Черненко О. О. Методичні рекомендації щодо виконання кваліфікаційної роботи студентів спеціальності 122 Комп'ютерні науки освітня програма «Комп'ютерні науки» ступеня магістра / С. В. Гаркуша, О. В. Ольховська, О. О. Черненко. – Полтава : ПУЕТ, 2023. – 68 с. Режим доступу: URL: http://elib.puet.edu.ua/action.php?kt_path_info=ktcore.SecViewPlugin.actions.document&fDocumentId=824868 - Назва з екрану

2. Telegram API Docs. Режим доступу: https://core.telegram.org/ - Назва з екрану

3. Путівник мовою програмування Python Режим доступу: https://pythonguide.rozh2sch.org.ua/ - Назва з екрану

4. Що таке мова програмування Python? Режим доступу: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-jazik-programmirovanija-python/ - Назва з екрану

5. Telethon's Documentation. Режим доступу: https://docs.telethon.dev/en/stable/ - Назва з екрану

6. PyCharm — інтегроване середовище розробки для мови програмування Python Режим доступу: URL: https://technologies-school.blogspot.com/2018/03/pycharm-python.html - Назва з екрану

7. Середовище розробки для Python: що це, які вони бувають і як їх використовувати Режим доступу: URL: https://foxminded.ua/seredovyshche-rozrobky-python/ - Назва з екрану

8. Система програмування PyCharm Режим доступу: URL: https://www.miyklas.com.ua/p/informatica/8-klas/algoritmi-ta-programi-394917/suchasni-movi-programuvannia-osnovi-roboti-v-isr-394274/re-1c2610f7-fbec-428a-8b2c-e011c30a05d3 - Назва з екрану

9. Pycharm: середовище розробки для Python. Режим доступу: URL: https://blog.desdelinux.net/uk/pycharm-un-entorno-de-desarrollo-para-python/- Назва з екрану

10. Гнучка методологія розробки ПЗ – Agile. Режим доступу: URL: https://training.qatestlab.com/blog/technical-articles/flexible-software-development-methodology-agile/ - Назва з екрану

11. Ольховська, О. В., Олексійчук, Ю. Ф., Кошова, О. П., Черненко, О. О., & Бойко, О. А. (2024). РОЗРОБКА TELEGRAM ЧАТ-БОТА ДЛЯ НАДАННЯ ТЕХНІЧНОЇ ПІДТРИМКИ У ГАЛУЗІ ТУРИСТИЧНИХ ПОСЛУГ. *Таврійський науковий вісник. Серія: Технічні науки*, (6), 35-44. https://doi.org/10.32782/tnv-tech.2023.6.5

12. Ольховська, О. В., Черненко, О. О., Ананенко, І. В., та ін. (2023). РОЗРОБКА НАВЧАЛЬНОГО ТРЕНАЖЕРУ З СИСТЕМНОГО АНАЛІЗУ У ВИГЛЯДІ ЧАТ-БОТУ Вісник Херсонського національного технічного університету № 2(85). https://doi.org/10.35546/kntu2078-4481.2023.2.27

13. Compare : Buffer vs Hootsuite vs Sprout Social vs Later. Режим доступу: URL: https://www.saasworthy.com/compare/buffer-vs-hootsuite-vs-sprout-social-vs-later?pIds=197%2C647%2C800%2C2085&utm_source=chatgpt.com - Назва з екрану.

14. Hootsuite vs Sprout Social: A 2025 Detailed Comparison. Режим доступу: URL: https://www.hopperhq.com/blog/hootsuite-vs-sprout-social/ - Назва з екрану.

ДОДАТОК А
Код програми

```
import private_config

api_id = private_config.API_ID
api_hash = private_config.API_HASH

database_password = private_config.DATABASE_PASSWORD
ssh_password = private_config.SSH_PASSWORD
ssh_host = private_config.SSH_HOST
ssh_port = private_config.SSH_PORT
database_port = private_config.DATABASE_PORT

spreadsheet_id = private_config.SPREADSHEET_ID



import pymysql
from sshtunnel import SSHTunnelForwarder

from config import *



class DataBaseUseCase:
    def __init__(self):
        self.__tunel = None
        self.__connection = None

    def open_connection(self):
        if self.__tunel is None:
            self.__tunel = SSHTunnelForwarder(
                (ssh_host, ssh_port),
                ssh_username="root",
                ssh_password=ssh_password,
```

```
            remote_bind_address=("localhost", database_port)
        )
        self.__tunel.start()

    if self.__connection is None:
        self.__connection = pymysql.connect(
            host="localhost",
            user="root",
            port=self.__tunel.local_bind_port,
            password=database_password,
            db="telegram_chats_db",
            charset="utf8mb4",
            cursorclass=pymysql.cursors.DictCursor
        )

def close_connection(self):
    if self.__connection is not None:
        self.__connection.close()
        self.__connection = None

    if self.__tunel is not None:
        self.__tunel.stop()
        self.__tunel = None

def _insert_message(self, model, type_chat):
    try:
        with self.__connection.cursor() as cursor:
            _command = "INSERT INTO `messages` (`id_message`, `id_sender`, `date_text`, `date_`,
`text_message`, `type_chat`) VALUES (%s, %s, %s, %s, %s, %s);"
            cursor.execute(_command, (
                model.id_message,
                model.id_sender_message,
                model.date_message,
                model.date_message,
```

```
                model.text_message,
                type_chat
            ))

            self.__connection.commit()
            return True
        except Exception as e:
            print(f"_insert_message: {e}")
            return None

    def _insert_chat(self, model, type_chat):
        try:
            with self.__connection.cursor() as cursor:
                _command = "INSERT INTO `chats` (`name`, `date_`, `chat_id`, `premium`, `bot`, `first_name`, `last_name`, `username`, `phone`, `type_chat`) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);"
                cursor.execute(_command, (
                    model.name,
                    model.date,
                    model.chat_id,
                    model.premium,
                    model.bot,
                    model.first_name,
                    model.last_name,
                    model.username,
                    model.phone,
                    type_chat
                ))

            self.__connection.commit()
            return True
        except Exception as e:
            print(f"_insert_chat: {e}")
            return None
```

```
    def _select_all_chats_by_type(self, type_chat):
        try:
            with self.__connection.cursor() as cursor:
                _command = "SELECT * FROM `chats` WHERE `type_chat` = %s;"
                cursor.execute(_command, (type_chat,))
                return cursor.fetchall()
        except Exception as e:
            print(f"_select_all_chats_by_type: {e}")
            return None


import os
import pickle
from datetime import datetime

from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build

from config import spreadsheet_id
from data_base_usecase import DataBaseUseCase


def update_google_sheets_(data, range_name):

    scopes = ['https://www.googleapis.com/auth/spreadsheets']

    # Шлях до файлу облікових даних
    creds_filename = 'credential.json'
    token_filename = 'token.pickle'

    # Завантажуємо збережені облікові дані, якщо вони є
    if os.path.exists(token_filename):
        with open(token_filename, 'rb') as token:
```

```
            creds = pickle.load(token)
    else:
        # Якщо збережених облікових даних немає, виконуємо авторизацію
        flow = InstalledAppFlow.from_client_secrets_file(creds_filename, scopes)
        creds = flow.run_local_server(port=0)
        # Зберігаємо облікові дані для наступних запусків
        with open(token_filename, 'wb') as token:
            pickle.dump(creds, token)

    service = build('sheets', 'v4', credentials=creds)

    values = [list(row) for row in data]

    # Опції для запису
    body = {
        'values': values
    }

    # # Виконання запису
    sheet = service.spreadsheets().values().update(
        spreadsheetId=spreadsheet_id,
        range=range_name,
        valueInputOption='RAW',
        body=body
    ).execute()

    print(f"Updated {sheet.get('updatedCells')} cells.")


def format_data_for_sheets(data):
    if not data:
        return []

    # Отримуємо заголовки з ключів першого словника
```

```
    headers = list(data[0].keys())

    # Додаємо заголовки як перший рядок
    formatted_data = [headers]

    # Додаємо дані
    for row in data:
        formatted_row = []
        for key in headers:
            value = row[key]
            # Якщо значення є datetime, перетворюємо його у строку
            if isinstance(value, datetime):
                value = value.strftime("%Y-%m-%d %H:%M:%S")
            formatted_row.append(value)
        formatted_data.append(formatted_row)

    return formatted_data


def upload_data_google_sheet(sheet, type_chat):
    database = DataBaseUseCase()
    database.open_connection()

    result = database._select_all_chats_by_type(type_chat)

    if result is not None:
        print(result)
        update_google_sheets_(format_data_for_sheets(result), sheet)

    database.close_connection()


from google_sheets_usecase import upload_data_google_sheet
from telegram_parse_usecase import TelegramParseChat
```

```
clickbait_session = "clickbait_session"
mt_shop_session = "mt_shop_session"
recruiter_session = "recruiter_session"

clickbait = 'clickbait'
mt_shop = 'mt_shop'
recruiter = 'recruiter'

clickbait_sheet = "Clickbait!A1"
mt_shop_sheet = "MT_Shop!A1"
recruiter_sheet = "Recruiter!A1"


telegram_parse_chat = TelegramParseChat(name_session=recruiter_session)


async def main():
    # await telegram_parse_chat.start_client()
    # await telegram_parse_chat.get_all_message(9999, clickbait)
    # await telegram_parse_chat.get_all_chats(type_chat=recruiter)
    pass

if __name__ == '__main__':
    upload_data_google_sheet(sheet=recruiter_sheet, type_chat=recruiter)
    # telegram_parse_chat.client.loop.run_until_complete(main())


from telethon import TelegramClient
from telethon.tl.types import PeerChannel

from config import api_id, api_hash
from data_base_usecase import DataBaseUseCase
```

```
class MessageModel:
    def __init__(self, id_message, id_sender, date_, text_message):
        self.id_message = id_message
        self.id_sender_message = id_sender
        self.date_message = date_
        self.text_message = text_message


class ChatModel:
    def __init__(self, name, date_, chat_id, premium, bot, first_name, last_name, username, phone):
        self.name = name
        self.date = date_
        self.chat_id = chat_id
        self.premium = premium
        self.bot = bot
        self.first_name = first_name
        self.last_name = last_name
        self.username = username
        self.phone = phone


class TelegramParseChat:
    def __init__(self, name_session):
        self.client = TelegramClient(f"sessions/{name_session}", api_id, api_hash)

    async def start_client(self):
        self.client.parse_mode = 'html'
        await self.client.start()

    async def get_all_message(self, limit_message: int, type_chat: str):
        chats = await self.client.get_dialogs()
```

```
    database = DataBaseUseCase()
    database.open_connection()

    for chat in chats:
        # for chat in chats:
        try:
            messages = await self.client.get_messages(PeerChannel(chat.id), limit=limit_message)

            for message in messages:
                model = MessageModel(
                    id_message=message.id,
                    id_sender=message.peer_id.user_id,
                    date_=message.date,
                    text_message=message.message
                )

                result = database._insert_message(model, type_chat)

                print(f"{result}\n{model.text_message}\n\n")
        except Exception as e:
            print(f"error: {e}")

    database.close_connection()

async def get_all_chats(self, type_chat: str):
    chats = await self.client.get_dialogs()

    database = DataBaseUseCase()
    database.open_connection()

    for chat in chats:
        try:
            model = ChatModel(
                name=chat.name,
```

```
                date_=chat.date.isoformat(),
                chat_id=chat.id,
                premium=chat.entity.premium,
                bot=chat.entity.bot,
                first_name=chat.entity.first_name,
                last_name=chat.entity.last_name,
                username=chat.entity.username,
                phone=chat.entity.phone
            )

            result = database._insert_chat(model, type_chat)
            print(f"{result} | {model.name}\n\n")
        except Exception as e:
            print(f"error: {e}")

    database.close_connection()


# Message(
# id=104322,
# peer_id=PeerUser(user_id=777000),
# date=datetime.datetime(2024, 1, 3, 11, 52, 20, tzinfo=datetime.timezone.utc),

# message='Код для входу: 57734. Не повідомляйте його нікому, навіть якщо співрозмовник наз
# ивається представником Telegram!\n\nЗа допомогою цього коду можна ввійти до вашого акаунта
# Telegram. Ми більше ні для чого його не просимо.\n\nЯкщо ви не намагаєтесь увійти з іншого
# пристрою, проігноруйте це повідомлення.',

# out=False,
# mentioned=False,
# media_unread=False,
```

```
# silent=False,
# post=False,
# from_scheduled=False,
# legacy=False,
# edit_hide=False,
# pinned=False,
# noforwards=False,
# invert_media=False,
# from_id=None,
# fwd_from=None,
# via_bot_id=None,
# reply_to=None,
# media=None,
# reply_markup=None,
# entities=[MessageEntityBold(offset=0, length=14),
# MessageEntityBold(offset=22, length=90)],
# views=None,
# forwards=None,
# replies=None,
# edit_date=None,
# post_author=None,
# grouped_id=None,
# reactions=None,
# restriction_reason=[],
# ttl_period=None
# )
#


# Dialog(
# name='Telegram',
# date=datetime.datetime(2024, 1, 3, 11, 52, 20,
# tzinfo=datetime.timezone.utc),
```

message=Message(id=104322, peer_id=PeerUser(user_id=777000),
date=datetime.datetime(2024, 1, 3, 11, 52, 20, tzinfo=datetime.timezone.utc),
message='Код для входу: 57734. Не повідомляйте його нікому, навіть якщо співрозмовник н
азивається представником Telegram!\n\nЗа допомогою цього коду можна ввійти до вашого акаунта Telegram.
Ми більше ні для чого його не просимо.\n\nЯкщо ви не намагаєтесь увійти з іншого пристрою, проігноруйте
це повідомлення.',
out=False,
mentioned=False,
media_unread=False,
silent=False,
post=False,
from_scheduled=False,
legacy=False,
edit_hide=False,
pinned=False,
noforwards=False,
invert_media=False,
from_id=None,
fwd_from=None,
via_bot_id=None,
reply_to=None,
media=None,
reply_markup=None,
entities=[MessageEntityBold(offset=0, length=14), MessageEntityBold(offset=22, length=90)],
views=None,
forwards=None,
replies=None,
edit_date=None,
post_author=None,
grouped_id=None,
reactions=None,
restriction_reason=[],

```
# ttl_period=None),

# entity=User(
# id=777000,
# is_self=False,
# contact=False,
# mutual_contact=False,
# deleted=False,
# bot=False,
# bot_chat_history=False,
# bot_nochats=False,
# verified=True,
# restricted=False,
# min=False,
# bot_inline_geo=False,
# support=True,
# scam=False,
# apply_min_photo=True,
# fake=False,
# bot_attach_menu=False,
# premium=False,
# attach_menu_enabled=False,
# bot_can_edit=False,
# close_friend=False,
# stories_hidden=False,
# stories_unavailable=True,
# access_hash=728775571588486482,
# first_name='Telegram',
# last_name=None,
# username=None,
# phone='42777',
# photo=UserProfilePhoto(photo_id=3337190045231025,
# dc_id=1,
# has_video=False,
```

```
# personal=False,
# stripped_thumb=b'\x01\x08\x08\x90y~Y\xe4\xef\xa2\x8a+\xa6\xc7%\xcf'),
# status=UserStatusRecently(),
# bot_info_version=None,
# restriction_reason=[],
# bot_inline_placeholder=None,
# lang_code=None,
# emoji_status=None,
# usernames=[],
# stories_max_id=None,
# color=None,
# profile_color=None)
# )


def dialog_to_dict(dialog):
    dialog_dict = {
        "name": dialog.name,
        "date": dialog.date.isoformat(),
        "draft": str(dialog.draft),
        "message": message_to_dict(dialog.message),
        "entity": user_to_dict(dialog.entity)
    }
    return dialog_dict


def message_to_dict(message):
    return {
        "id": message.id,
        "peer_id": {
            "user_id": message.peer_id.user_id
        },
        "date": message.date.isoformat(),
        "message": message.message,
```

```
        "out": message.out,
        "mentioned": message.mentioned,
        "media_unread": message.media_unread,
        "silent": message.silent,
        "post": message.post,
        "from_scheduled": message.from_scheduled,
        "legacy": message.legacy,
        "edit_hide": message.edit_hide,
        "pinned": message.pinned,
        "noforwards": message.noforwards,
        "invert_media": message.invert_media,
        "from_id": message.from_id,
        "fwd_from": message.fwd_from,
        "via_bot_id": message.via_bot_id,
        "reply_to": message.reply_to,
        "media": str(message.media),
        "reply_markup": str(message.reply_markup),
        "entities": [str(entity) for entity in message.entities] if message.entities else [],
        "views": message.views,
        "forwards": message.forwards,
        "replies": message.replies,
        "edit_date": message.edit_date,
        "post_author": message.post_author,
        "grouped_id": message.grouped_id,
        "reactions": message.reactions,
        "restriction_reason": message.restriction_reason,
        "ttl_period": message.ttl_period
    }


def user_to_dict(user):
    return {
        "id": user.id,
        "is_self": user.is_self,
```

"contact": user.contact,
"mutual_contact": user.mutual_contact,
"deleted": user.deleted,
"bot": user.bot,
"bot_chat_history": user.bot_chat_history,
"bot_nochats": user.bot_nochats,
"verified": user.verified,
"restricted": user.restricted,
"min": user.min,
"bot_inline_geo": user.bot_inline_geo,
"support": user.support,
"scam": user.scam,
"apply_min_photo": user.apply_min_photo,
"fake": user.fake,
"bot_attach_menu": user.bot_attach_menu,
"premium": user.premium,
"attach_menu_enabled": user.attach_menu_enabled,
"bot_can_edit": user.bot_can_edit,
"close_friend": user.close_friend,
"stories_hidden": user.stories_hidden,
"stories_unavailable": user.stories_unavailable,
"access_hash": user.access_hash,
"first_name": user.first_name,
"last_name": user.last_name,
"username": user.username,
"phone": user.phone,
"photo": str(user.photo),
"status": str(user.status),
"bot_info_version": user.bot_info_version,
"restriction_reason": user.restriction_reason,
"bot_inline_placeholder": user.bot_inline_placeholder,
"lang_code": user.lang_code,
"emoji_status": user.emoji_status,
"usernames": user.usernames,

```
    "stories_max_id": user.stories_max_id,
    "color": user.color,
    "profile_color": user.profile_color
}
```