СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра інформаційних технологій та програмування

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр

спеціальність *126 Інформаційні системи та технології*
(шифр і назва спеціальності)

на тему *Чат-бот у Telegram для автоматизації відповідей*

Виконав: студент групи ІСТ-21д	_____ (підпис)	М.В.Кухар (ініціали і прізвище)
Керівник	_____ (підпис)	В. Г. Іванов (ініціали і прізвище)
Завідувач кафедри	_____ (підпис)	О.І. Захожай (ініціали і прізвище)

Рецензент _____В.О. Лифар_____

Київ – 2025

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра інформаційних технологій та програмування
Освітній ступінь бакалавр
Спеціальність *126 Інформаційні системи та технології*
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Захожай О.І.
"___" _____ 2025 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ
Кухар Максим Володимирович
1. Тема роботи *Чат-бот у Telegram для автоматизації відповідей*

Керівник роботи *д.т.н., доцент, Іванов Віталій Геннадійович*,
(прізвище, ім'я, по-батькові, науковий ступінь, вчене звання)
Затверджений наказом університету від "*27*"*травня* 2025 року № *67/15.15-С*
2. Строк подання студентом роботи *14.06.2025 р.*
3. Вихідні дані роботи *Матеріали переддипломної практики, науково-методична література, дані інтернет мережі*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

4.1 Вступ

4.2 Аналітичний огляд питання.

4.3 Основна частина, де досліджено та проаналізовано методи для реалізації проекту, а також сформовано вирішення для розв'язку поставленої задачі.

4.4 Практична частина, в якій зроблено огляд технологій та інструментарію, які використовуються для реалізації проекту.

4.5 Висновки

4.6 Перелік використаних джерел.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників)

6. Консультанти розділів роботи

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

7. Дата видачі завдання *30.03.2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1.	Отримання завдання на виконання роботи	30.03.2025	
2.	Укладення та погодження з керівником плану і етапів виконання роботи	04.04.2025	
3.	Погодження з керівником оптимального шляху виконання завдання.	12.04.2025	
4.	Аналіз технічних засобів існуючих систем	21.04.2025	
5.	Реалізація практичної частини	15.05.2025	
6.	Написання, оформлення та узгодження пояснювальної записки з керівником	09.06.2025	
7.	Здача пояснювальної записки на кафедру	14.06.2025	
8.	Підготовка доповіді та презентації	16.06.2025	

Студент _____ *Кухар М. В.*
(підпис) (ініціали і прізвище)

Керівник роботи _____ *Іванов В. Г.*
(підпис) (ініціали і прізвище)

РЕФЕРАТ

Бакалаврська дипломна робота: 45 стор., 3 табл., 6 рис., 10 джерел.

Об’єкт дослідження – архітектура побудови чат-ботів.

Мета роботи – аналіз складових архітектури побудови автоматизованих ботів, для розробки власного чат-боту для Telegram.

Проаналізовано технології автоматизації, обробки та шифрування даних. Розроблений власний чат-бот на основі цих даних.

Для досягнення поставленої мети, дипломна робота поділена на такі завдання:

1) Дослідити предметну область;
2) Розглянути архітектуру побудови ботів;
3) Написати логіку бота, на основі загальної концепції побудови ботів;

ЗАПИТ, ЧАТ-БОТ, БОТ, PBKDF2, SQLITE, NLP, MVC, PYTHON, VISUAL STUDIO, AES-256, API, TELEGRAM, TELEBOT, АВТОМАТИЗАЦІЯ, BACKEND, СЕРВЕР, HTTP

ENGLISH VERSION

Bachelor's thesis: 45 pages, 3 tables, 6 figures, 10 pages.

Object of study – architecture of chatbots.

The aim of the thesis is to analyze the components of the architecture of automatized bot design in order to develop a proprietary chatbot for Telegram.

Technologies for automation, data processing, and encryption are analyzed. A proprietary chatbot is developed based on this data.

To achieve the set goal, the thesis is divided into the following tasks:

1) Research the subject area;
2) Consider the architecture of bot construction;
3) Write the bot's logic based on the general concept of bot building;

REQUEST, CHATBOT, BOT, PBKDF2, SQLITE, NLP, MVC, PYTHON, VISUAL STUDIO, AES-256, API, TELEGRAM, TELEBOT, AUTOMATION, BACKEND, SERVER, HTTP

ЗМІСТ

ВСТУП

Автоматизація процесів, це невід’ємна частина сучасного робочого циклу. Кожен підприємець, бажає отримати якомога більшу вигоду зі свого робочого процесу, витративши якомога менше зусиль. Оптимізація, слово яке вирішує багато проблем, та економить час. Саме автоматизація процесів, і дає змогу оптимізувати робочий процес, не вносячи кардинальних змін, в структуру виконання робочого процесу. Цей принцип, є фундаментальним напрямом науково-технічного прогресу.

Дивлячись на історію автоматизації, то першим прикладом глобально-налаштованих процесів автоматизації є прядильна фабрика Річарда Аркрайта, що була побудована в 1771 році. Вона приводилася до дії – гідроенергію. Серед більш сучасних винахідників, можна беззаперечно вважати Генрі Форда. У 1913-ому році він починав роботи зі створення та встановлення в цехах свого підприємства безперервної лінії по збірці автомобілів моделі «Т». Конвеєрна лінія на одному з заводів Форда, у штаті Мічиган, стало початком індустріального перевороту, що затвердило назавжди в світовій практиці застосування автоматизації робочого процесу.

В свою чергу, дипломна робота має наметі чітко показати оптимізацію повсякденного робочого процесу, за допомогою програмного забезпечення у вигляді чат-боту. Актуальність теми, підкреслюється тим, що наразі дуже популярна концепція чат-ботів, наприклад на сайтах магазинів замість реальних менеджерів виступають чат-боти, або ж спілкування з AI про ходить через таку концепцію. Фінальний концепт, можна використовувати вже зараз.

Підсумовуючи, головна мета дипломної роботи, це розгляд видів автоматизації робочих процесів, далі вибір мови програмування та побудова алгоритму роботи чат-боту з імплементацією кешування повторюваної інформації у базах даних MySQL.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Автоматизація: основні принципи та застосування в інформаційних системах

Автоматизація є процесом використання технологій для виконання завдань із мінімальним втручанням людини. Згідно з визначенням, автоматизація охоплює використання комп'ютерних систем, програмного забезпечення та інших технічних засобів для оптимізації процесів у різних галузях, включаючи промисловість, бізнес і сферу послуг. У контексті інформаційних систем автоматизація спрямована на підвищення ефективності, зменшення помилок і скорочення часу виконання завдань.

Основні принципи автоматизації включають:

- **Механізація процесів:** заміна ручної праці програмними засобами;
- **Стандартизація:** створення уніфікованих процедур для обробки даних;
- **Інтеграція:** об'єднання різних систем для безперебійного обміну інформацією;
- **Адаптивність:** здатність системи реагувати на зміни умов або вимог;

У розробці чат-ботів автоматизація відіграє ключову роль, дозволяючи обробляти велику кількість запитів без участі оператора. Наприклад, чат-боти автоматизують відповіді на типові запитання, обробку замовлень або підтримку клієнтів. Використання Python для створення таких систем забезпечує гнучкість завдяки бібліотекам, таким як asyncio для асинхронної обробки запитів, або pandas для аналізу даних. [1]

Автоматизація також передбачає використання баз даних, таких як MySQL, для зберігання та швидкого доступу до інформації. Це дозволяє чат-ботам оперативно отримувати дані, необхідні для генерації відповідей, що є важливим для масштабування систем. Наприклад, автоматизована система кешування зменшує час обробки запитів шляхом збереження результатів у базі даних. [2]

1.2 Чат-боти: визначення, класифікація та основні принципи роботи

Чат-боти є програмними системами, що імітують людське спілкування через текстові або голосові інтерфейси. Вони використовуються для автоматизації взаємодії з користувачами, що дозволяє оптимізувати бізнес-процеси, зменшувати витрати на підтримку клієнтів та підвищувати ефективність комунікації. Згідно з визначенням Рассела та Норвіга, чат-боти належать до систем штучного інтелекту, які базуються на обробці природної мови (NLP) для інтерпретації запитів та генерації відповідей. [3] Чат-боти можна класифікувати за типом взаємодії та складністю обробки даних.

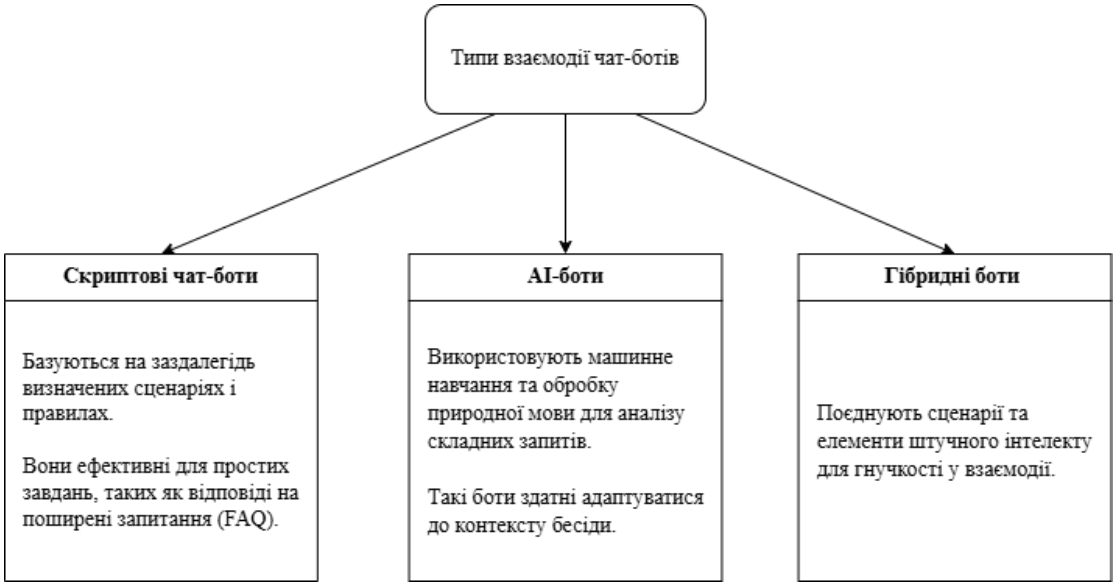

Рисунок 1.2.1 Класифікація чат-ботів, за типом взаємодії

Основні принципи роботи чат-ботів включають:

- **Обробку вхідних даних:** аналіз тексту або голосу за допомогою NLP-технік;
- **Генерацію відповідей:** використання баз знань, шаблонів або моделей машинного навчання;
- **Підтримка контексту:** збереження історії діалогу для підтримки логічної послідовності.

Для реалізації чат-ботів на Python часто використовуються бібліотеки, такі як NLTK, spaCy або Rasa, які забезпечують інструменти для обробки тексту та створення діалогових систем. [4]

1.3 Технології автоматизації відповідей у чат-ботах

Автоматизація відповідей у чат-ботах залежить від технологій обробки природної мови, баз знань та алгоритмів пошуку відповідей. Основними компонентами є:

- **Розпізнавання намірів (*Intent Recognition*):** визначення мети запиту користувача. Наприклад, бібліотека Rasa використовує моделі машинного навчання для класифікації намірів.
- **Генерація відповідей:** створення відповідей на основі шаблонів, баз знань або нейронних мереж. Наприклад, моделі на основі трансформерів, такі як BERT, дозволяють генерувати контекстно-залежні відповіді.
- **Управління діалогами:** забезпечення логічної послідовності бесіди. Це досягається за допомогою кінцевих автоматів або систем управління станами (Dialogue State Tracking).

Для автоматизації відповідей важливим є використання баз знань, які можуть бути структурованими (наприклад, у вигляді баз даних) або неструктурованими (текстові документи). У контексті чат-ботів на Python бібліотеки, такі як ChatterBot, дозволяють створювати прості системи автоматизації відповідей, тоді як фреймворки на кшталт Dialogflow надають хмарні рішення для складних діалогових систем.

Кешування відповідей є ключовим елементом для підвищення продуктивності. Воно дозволяє зберігати результати попередніх запитів, зменшуючи час обробки повторюваних запитів. У Python це може бути реалізовано через бібліотеки, такі як sqlite3 для роботи з SQLite або використання інших локальних баз даних.

1.4 Роль баз даних SQLite у системах кешування для чат-ботів

Бази даних SQLite є популярним вибором для зберігання даних у чат-ботах завдяки своїй легкості, портативності та вбудованій підтримці в Python через бібліотеку sqlite3. SQLite є без серверною базою даних, що не вимагає окремого серверного процесу, що робить її ідеальною для локальних застосунків, таких як чат-боти з системами кешування. У контексті чат-ботів SQLite використовується для:

- Зберігання історії діалогів: збереження контексту бесід для підтримки послідовності взаємодії.
- Кешування відповідей: збереження результатів обробки запитів для швидкого доступу при повторних запитах.
- Управління базами знань: зберігання структурованих даних, таких як шаблони відповідей або інформація про запити.

Система кешування в чат-ботах зменшує час обробки запитів шляхом збереження часто використовуваних відповідей у базі даних. Наприклад, якщо користувач повторно запитує переклад тексту, прогноз погоди або новину, чат-бот може отримати кешовану відповідь із SQLite замість повторного звернення до зовнішнього API. SQLite забезпечує швидкий доступ до даних завдяки своїй легкій архітектурі та підтримці індексів, що оптимізують пошук. [5]. В поточному чат-боті інтегрована стандартна бібліотека SQLite в Python, і прикладом однієї з таблиць для кешування даних, можна подивитися нижче, в Таблиці 1.4.1 .

Таблиця 1.4.1 Приклад структури SQLite таблиці

Поле	**Тип даних**	**Опис**
id	INTEGER	Унікальний ідентифікатор (автоінкремент)
source_text	TEXT	Вихідний текст для перекладу
target_lang	TEXT	Мова перекладу
translation_result	TEXT	Результат перекладу

Ці таблиці дозволяють чат-боту ефективно кешувати різноманітні типи даних, такі як переклади, новини та прогнози погоди, що зменшує кількість запитів до зовнішніх сервісів і підвищує швидкість відповідей. Оптимізація пошуку може бути досягнута шляхом створення індексів на часто використовуваних полях, таких як source_text або city, що прискорює виконання SQL-запитів.

1.5 Висновки розділу

Аналіз предметної області показав, що автоматизація є основою для створення ефективних чат-ботів, дозволяючи зменшити людське втручання та оптимізувати обробку запитів. Чат-боти, як частина інформаційних систем, використовують принципи автоматизації для обробки природної мови, управління діалогами та швидкого доступу до даних. Технології, такі як розпізнавання намірів і генерація відповідей, забезпечують гнучкість і адаптивність систем.

Бази даних SQLite відіграють ключову роль у реалізації систем кешування, що підвищує продуктивність чат-ботів за рахунок швидкого доступу до збережених даних. Використання Python як основної мови програмування забезпечує доступ до широкого спектра бібліотек для обробки тексту, інтеграції з базами даних і створення діалогових систем. Подальша розробка чат-бота потребує детального аналізу вимог до кешування, оптимізації запитів до бази даних та інтеграції автоматизованих процесів.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ПОБУДОВИ БОТІВ

2.1 Архітектура ботів

Архітектура ботів базується на сукупності компонентів, що забезпечують обробку природної мови, логіку відповідей та інтеграцію з зовнішніми системами. Дослідження архітектури дозволяє зрозуміти основні принципи їхньої побудови та адаптації до різних завдань, зокрема в сфері освіти, медицини та комерції.

Ефективна побудова ботів базується на кількох ключових принципах:

- **Модульність:** Розділення на незалежні компоненти для спрощення розробки та оновлення.
- **Масштабованість:** Можливість додавання нових команд без значних змін у коді.
- **Безпека:** Захист даних користувачів через шифрування та автентифікацію.
- **Швидкодія:** Оптимізація алгоритмів для миттєвих відповідей.
- **Простота:** Використання чітких команд для взаємодії з користувачами

Боти з такою побудовою базується на простій, але надійній структурі, що включає інтеграцію з зовнішніми сервісами через API, обробку команд користувачів і забезпечення безпеки даних. А що ж таке в свою чергу API? API (Application Programming Interface) – це інтерфейс який дозволяє різним програмам взаємодіяти між собою. Іншими словами посередник між програмами який задає правила взаємодії.

В підтвердження вище написаного, дослідження показують [6], що проста архітектура зменшує час розробки, швидкість обробки запитів, навантаження на сервер, імовірність збою, та витрати на ресурси – може доходити приблизно до 40-50% порівняно зі складними системами з NLP. (Див. Табл. 2.1.1)

Таблиця 2.1.1 Порівняння ефективності архітектур ботів

Метрика	**Проста архітектура (API)**	**Складна архітектура (NLP)**	**Відсоток переваги простої архітектури**
Час розробки (*години*)	50-70	80–100	20–30%
Швидкість обробки (мс)	100–150	250–300	50–70%
Навантаження на сервер (%)	10–15%	30–40%	15–25%
Імовірність збою (%)	5–10%	15–20%	50%

API технологій, як стало вже зрозуміло, є ефективним способом для автоматизації завдань у реальному часі. Найчастіше потрібно зекономити ресурси в побудові багатофункціонального програмного забезпечення, чим і допомагають різні API. Але потрібно зазначити, що не під кожну задачу раціонально використовувати архітектуру з API, можливо десь ефективніше та доцільніше, використати власну розробку адаптовану під конкретні потреби.

В контексті дипломної роботи, backend-код боту опирається на вище загадані принципи ефективності побудови: за швидкодію, і простоту відповідають API-бібліотеки, за безпеку криптографічні методи шифрування та SQLite база даних, а за модульність і масштабність – Python, як інтерпретована мова програмування, що дає змогу простіше відноситись до написання програмного коду у менш формалізованому але більш гнучкому стилі.

Після чіткої побудови структури програми, її можна зобразити у вигляді блок-схеми, що зображена нижче на Рисунку 2.1.1

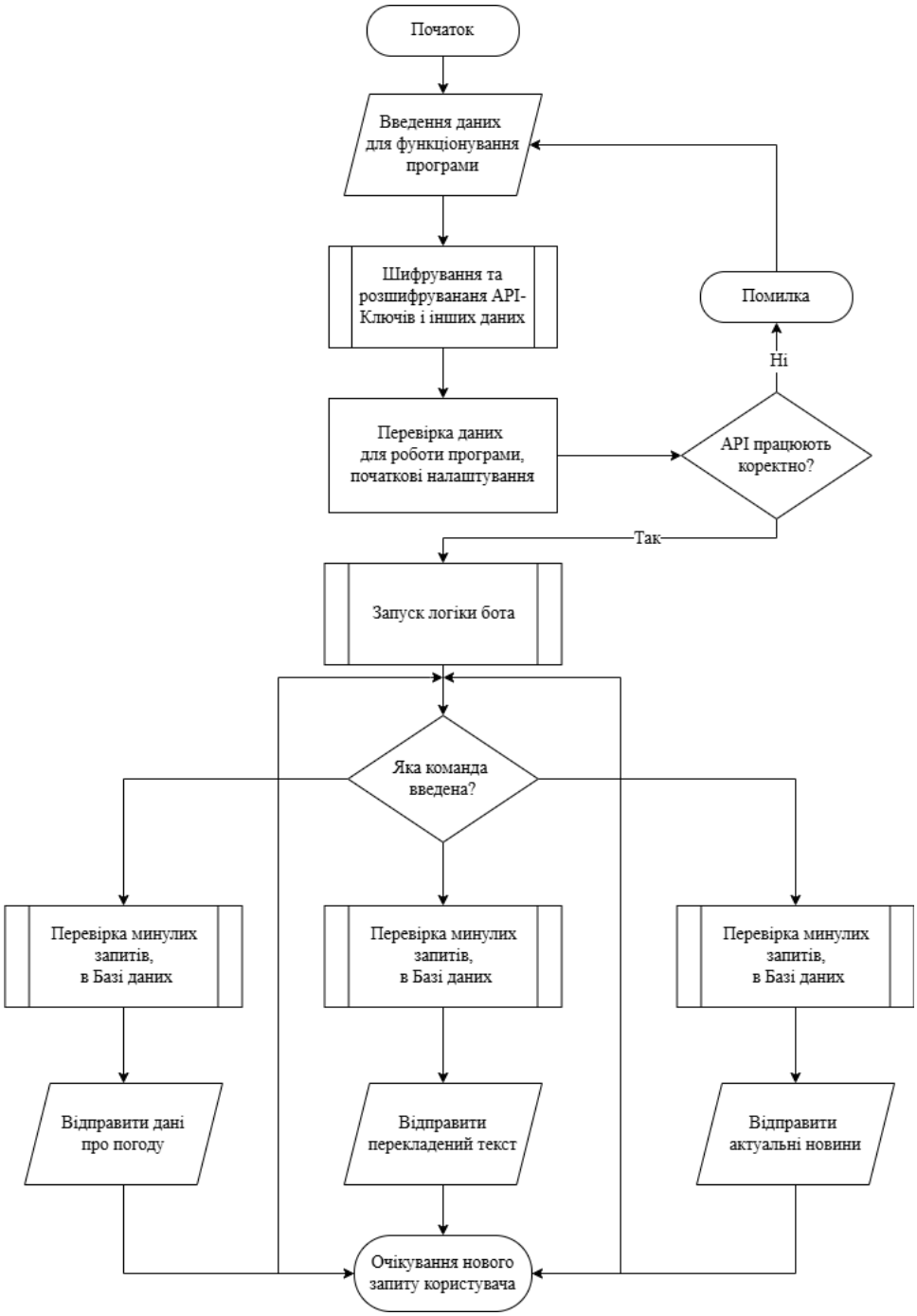

Рисунок 2.1.1 Блок схема логіки роботи бота

Як видно на блок-схемі, логіка починається з вводу даних, для функціонування API-Бібліотек і Telegram API: api-ключ телеграм-боту, id-чату для роботи бота в певному чаті(в своєму власному або в групі), та унікальний пароль який потрібно ввести для шифрування цих даних. Далі, йде перевірка введення цих даних, при якій усі бібліотеки роблять тестовий запит, причому в разі того якщо один з цих запитів невдалий, повертаємось до кроку з введенням даних.

Якщо все добре з даними, починається повноцінний запуск Telegram API, як основної API. Нескінченний цикл(polling), починає очікувати на запит команди від користувача. І останнім кроком можна вважати відповідь на команду, із запитом до сторонніх API які надають актуальну інформацію. Також для пришвидшення роботи бота, використовуються база даних SQLite, яка зберігає всередині себе останні запити за кожною командою. Алгоритм, кожен раз перед видачою відповіді користувачеві, перевіряє чи не було цих даних в БД – якщо ж вони були, то замість нового звернення до API-бібліотек, і економії часу і ресурсів серверу, видається повідомлення з БД, яка є таким собі буфером даних на постійній основі.

Підсумовуючи, архітектура цього телеграм-бота побудована за ефективними та раціональними принципами. В наступних підрозділах, розглянемо окремі частини, такі як: api-бібліотеки, api для взаємодії з telegram, та шифрування даних.

2.2 API Бібліотеки, для взаємодії з зовнішніми ресурсами

Для вирішення поставленої задачі, потрібно ефективно отримувати дані з агрегаторів новин або погоди, чи наприклад використовувати словники для перекладу тексту. В цьому випадку допоможуть API-бібліотеки, які при не складній задачі швидко вирішують, як інтерфейс, взаємодію зі сторонніми ресурсами.

В якості інтерфейсу, для новинного ресурсу, використовується News API. Це простий HTTP Rest API (Representational State Transfer Application Programming Interface, або ж підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів), для пошуку та отримання опублікованих статей з усього Інтернету. News API відомий сервіс, який надає свої послуги таким великим компаніям як Samsung, Walmart, і феноменальну підтримку клієнтських бібліотек для таких сучасних мов програмування, як: Python, Node.js, Rubi, C#, Java і PHP.

Новини фільтруються за будь-якою комбінацією наступних критеріїв:

- **Ключове слово або фраза:** знайти всі статті, що містять слово «Microsoft».
- **Дата публікації:** знайти всі статті, опубліковані від Березня 2025 року.
- **Ім'я вихідного домену:** знайти всі статті, опубліковані на ukr.net
- **Мова:** знайти всі статті, написані англійською мовою.

Єдиний недолік який можна було б підкреслити, це те що дана API комерційна, тому звичайним клієнтам надана обмежена кількість запитів, на певний проміжок часу. Але це все не стосується наступного API для актуальної інформації щодо погоди.

Безкоштовний сервіс wttr.in , призначений для отримання погоди, який повертає дані у текстовому форматі через url-запити. Абсолютна безкоштовна бібліотека без обмежень по кількості запитів. Була створена приблизно в 2017 році, та спочатку мала призначення тільки обгортки (Це програмний код, що створюється навколо існуючого інструмента, бібліотеки чи API, щоб спростити використання), для іншої бібліотеки для отримання прогнозу погоди – wego.

На практиці, запит може включати багато критеріїв для отримання інформації, але в даному випадку використовуються саме такі позначки:

- {city} – Місто, або інший населений пункт в якому визначити погоду (наприклад, "Київ" або "Kyiv")
- %C — умова погоди (наприклад, "Partly cloudy"),
- %t – температура (наприклад, "+20°C"),
- %w – швидкість вітру (наприклад, "14km/h"),
- &lang=uk – встановлення української мови.

І останньою на черзі, є deep_translator – бібліотека для Python, що розроблена для спрощеного доступу до відомих перекладацьких сервісів, таких як DeepL, Microsoft Translator, Google Translator. Вона є open-source проектом(не комерційним), тому не немає обмежень в кількості запитів.

Бібліотека імітує запити до веб-версії перекладацького сервісу, використовуючи HTTP-запити (зазвичай через бібліотеку requests). Вона не потребує офіційного API-ключа, а замість цього аналізує HTML-відповіді або AJAX-запити сайту.

Дане API, має вбудований обробник винятків, що дозволяє більш точно налагоджувати та шукати помилки, за допомогою конструкції try-except. В контексті дипломної роботи використовуються такі параметри:

- source='auto' — автоматично визначає мову вхідного тексту.
- target=target_lang — вказує цільову мову (наприклад, "uk" для української).
- translate(text) — виконує переклад і повертає результат як рядок.

2.3 Telegram – сучасна платформа для роботи з ботами

Окремо потрібно зазначити «серце» чат-боту, яким є Telegram API, що є прямим інтерфейсом для роботи самого бота і взаємодії з користувачем, на комфортному рівні.

Telegram є одним із провідних месенджерів у світі, який поєднує в собі функції швидкого обміну повідомленнями з можливостями створення та інтеграції чат-ботів. Був запущений 14 серпня 2013 року як альтернатива іншим месенджерам, таким як WhatsApp, із акцентом на шифрування повідомлень і швидкість доставки. Використання власного протоколу MTProto забезпечує end-to-end шифрування в секретних чатах, тоді як звичайні чати захищені серверним шифруванням. Його архітектура базується на хмарних технологіях, що забезпечує доступність на різних пристроях, а підтримка ботів зробила його популярним інструментом для автоматизації завдань.

Станом на 2024 рік Telegram має понад 900 мільйонів активних користувачів щомісяця (Див. Рис. 2.3.1), що робить його однією з найбільших платформ для комунікації. Його відкритий API та підтримка ботів сприяли розвитку екосистеми, де користувачі можуть створювати власні автоматизовані сервіси.

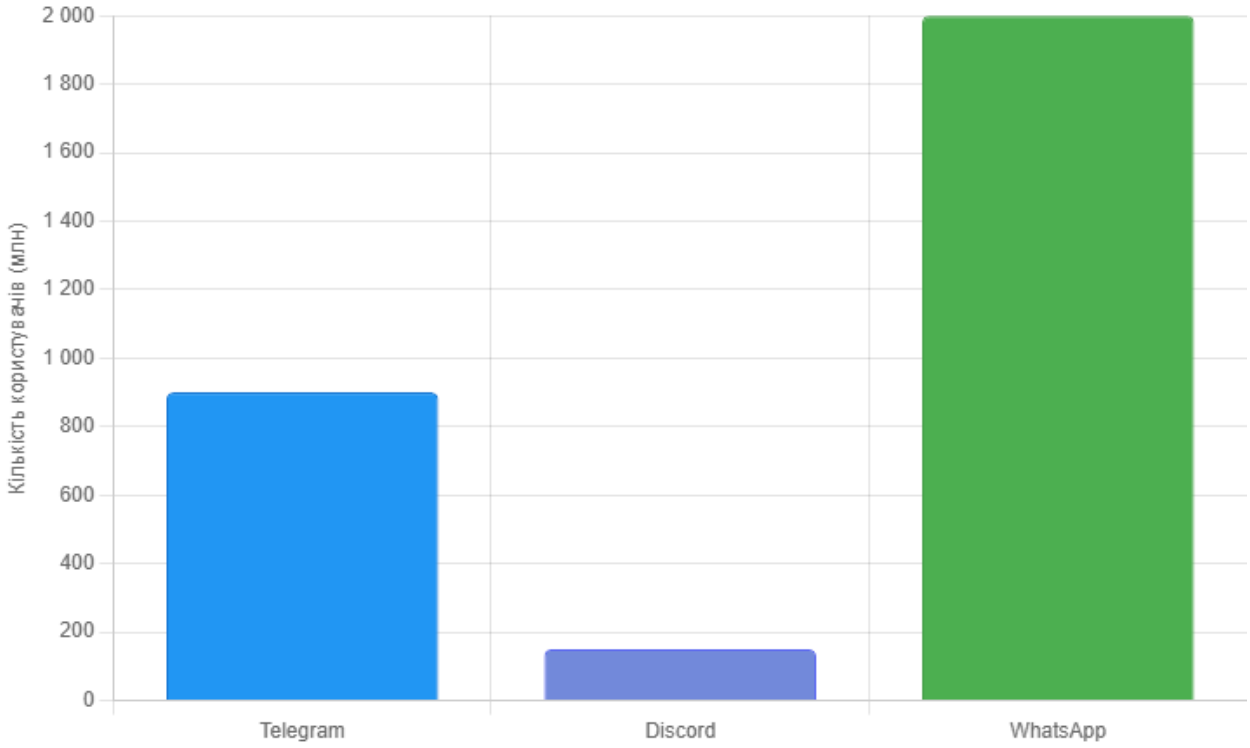

Рисунок 2.3.1 Кількість користувачів популярних месенджерів (млн.)

Telegram є однією з найпопулярніших платформ для роботи з ботами та розмовами. За даними журналу "Artificial Intelligence Review" (2023), понад 30% ботів, створених у 2022–2023 роках, були інтегровані з Telegram завдяки його відкритості та безпеці [7]. Книга "Building Chatbots with Python" (Smith, 2021) зазначає, що Telegram вирізняється серед конкурентів (наприклад, Discord) завдяки підтримці багатомільйонної аудиторії та зручному API [8]. Це робить його ідеальним для освітніх, комерційних та розважальних ботів.

Підтримка ботів у Telegram була введена 24 червня 2015 року з релізом Bot API, розробленим командою Telegram. Telegram Bot API — це RESTful API, яке надає набір методів для взаємодії з месенджером. Для використання бота необхідно отримати токен через бот @BotFather, який генерується Telegram. API підтримує такі ключові функції:

- Надсилання та отримання повідомлень (sendMessage, getUpdates).
- Обробка команд через веб-хуки або довгі опитування.
- Інтеграція з мультимедіа (зображення, аудіо) та інтерактивними елементами (кнопки).

У 2016 році з’явилася альтернативна бібліотека telebot, розроблена Етьєном Роба (Etienne Roba's), яка стала популярною завдяки простоті використання. Обидві бібліотеки дозволили Python-розробникам швидко інтегрувати ботів у Telegram, що сприяло зростанню їхньої популярності. (Див. Рис. 2.3.2)

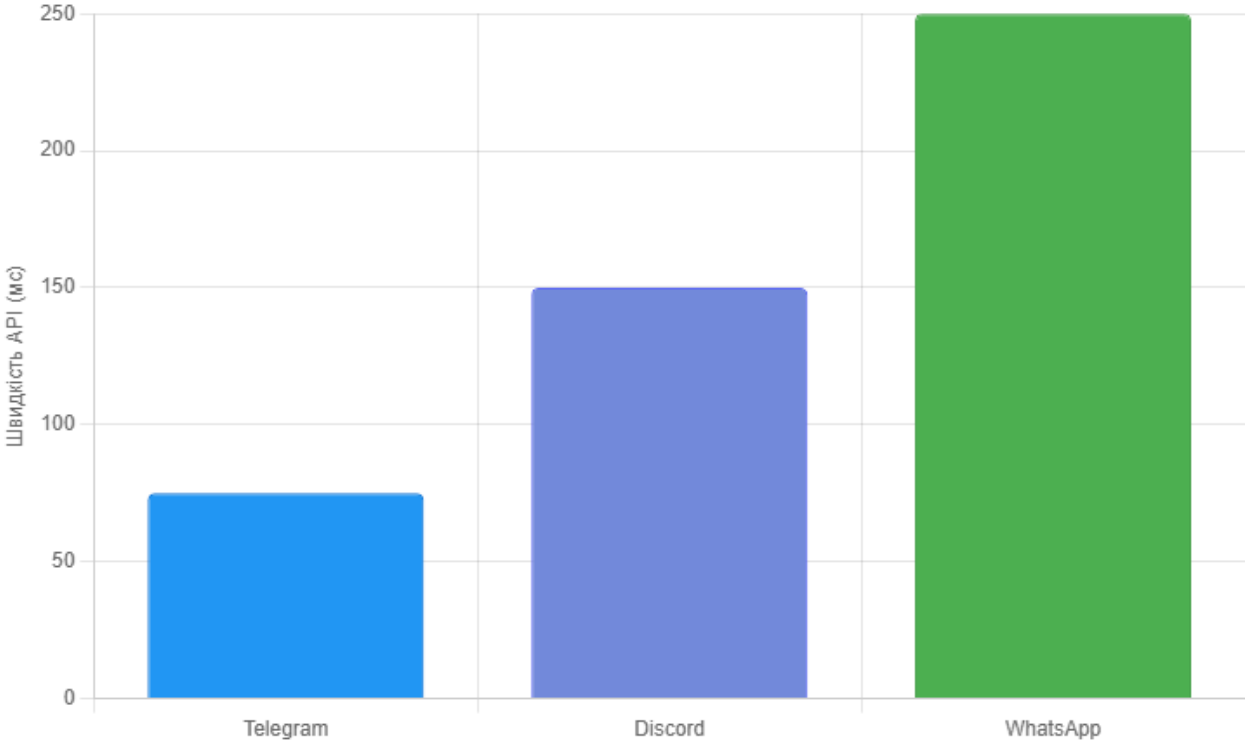

Рисунок 2.3.2 Швидкість роботи API (мс.)

Цей API дозволяє розробникам створювати ботів, які взаємодіють із користувачами через чати, надсилають повідомлення, обробляють команди та інтегруються з зовнішніми сервісами. Боти можуть бути додані до груп, каналів або використовуватися в особистих чатах, що розширює їхній функціонал. Популярність Telegram як платформи для ботів зумовлена низкою факторів:

- Безкоштовний доступ до API.
- Гнучкість у реалізації логіки через бібліотеки, такі як python-telegram-bot або telebot.
- Підтримка багатомовності та кастомізованих інтерфейсів (наприклад, клавіатур).

В дипломній роботі, використовується бібліотека для Python, така як telebot, для спрощення роботи з офіційним API від Telegram. Наприклад, декоратор @bot.message_handler(commands=['start']) дозволяє обробляти команду /start. Дослідження підтверджують [9], що Telegram Bot API забезпечує швидкість обробки до 1000 повідомлень за секунду на одного бота, що робить його придатним для масштабних проектів.

2.4 Шифрування даних

Не менш важливою частиною, побудови архітектури ботів, є захист секретних даних при роботі логіки бота. Оскільки можна вважати, що бот є клієнт-серверною програмою, то у вихідному коді не повинно міститися тих самих API-Ключів ітд. В цьому випадку потрібно тимчасово зберігати важливі дані, чим і допоможуть методи шифрування даних.

Гарним «тоном» шифрування даних вважаться, використання загально відомих методів і концепцій шифрування даних – оскільки найкращі методи вже придумані і масово використовуються, а написання своїх власних алгоритмів може призвести до серйозних прогалин в захисті. Але в цьому випадку, постає питання «Як користуватися загальновідомими методами шифрування, якщо вони такими є, і хтось вже знає як зламувати цей захист?», і тут можна апелювати тим що: «Все чим можна зарадити, це виграти якомога більше часу, якщо буде процес зламу цих даних». Тому тут допоможе комбінування різних методів шифрування даних.

Шифрування ключів, реалізоване за допомогою бібліотеки cryptography з алгоритмом PBKDF2(Password-Based Key Derivation Function 2), захищає доступ до сервісів, зберігаючи ключі у зашифрованому вигляді. Цей підхід дозволяє:

- Захищати ключі від несанкціонованого доступу.
- Зберігати їх у зашифрованому вигляді на диску.
- Забезпечувати повторне використання ключа лише з правильним паролем.

Алгоритм реалізує симетричне шифрування AES-256 у режимі CBC із використанням PBKDF2 для генерації ключа. Дані шифруються з додаванням випадкової солі та ініціалізаційного вектору (IV), що забезпечує унікальність кожного шифрування навіть для однакових вхідних даних. Розшифрування відновлює оригінальні дані за допомогою того ж пароля. Уточнимо, що кожен з елементів значить в цьому складеному алгоритмі шифрування.

AES (Advanced Encryption Standard) — це стандарт шифрування, який використовується у всьому світі. "256" означає, що ключ має 256 біт (дуже великий), і зламати його практично неможливо без пароля.

AES працює з блоками по 16 байтів. Якщо дані не діляться на 16, ми додаємо "набивку". PKCS7 додає байти, які показують, скільки було додано (наприклад, якщо додано 5 байтів, кожен із них має значення 5).

У режимі CBC кожен блок даних змішується з попереднім перед шифруванням. Це робить шифрування безпечнішим, але вимагає IV (ініціалізаційного вектору), щоб почати процес.

Сіль — це випадкові дані, які додаються до пароля, щоб кожен ключ був унікальним, а в свою чергу, IV — це випадкові дані, які додаються до шифрування, щоб кожен шифротекст був унікальним.

Тепер знаючи функціонування кожної частини, можна задокументувати покрокову реалізація шифрування і розшифрування даних:

1. **Генерація ключа (функція GetKeyFromPass):**
   1.1. Вхід: Пароль (password) і сіль (salt).
   1.2. Ініціалізація PBKDF2 із SHA-256, довжиною 32 байти, 100000 ітерацій.
   1.3. Використання derive() для створення ключа з пароля та солі.
   1.4. Повернення ключа.
2. **Шифрування (функція Encrypt):**
   2.1. Вхід: Дані (data) і пароль (password).
   2.2. Генерація випадкової солі (16 байтів) і IV (16 байтів).
   2.3. Генерація ключа через GetKeyFromPass.
   2.4. Створення об’єкта шифрування AES у режимі CBC із IV.

2.5. Додавання відступу PKCS7 до даних для відповідності розміру блоку (16 байтів).
2.6. Шифрування даних і повернення конкатенації salt + iv + encrypted_data.

3. Розшифрування (функція Decrypt):

3.1. Вхід: Зашифровані дані (encrypted_data) і пароль (password).
3.2. Екстракція солі, IV і зашифрованих даних із вхідного буфера.
3.3. Генерація ключа через GetKeyFromPass із солі.
3.4. Створення об’єкта дешифрування AES у режимі CBC із IV.
3.5. Розшифрування даних і видалення паддінгу PKCS7.
3.6. Повернення розшифрованих даних.

Процес шифрування включає кодування пароля в байти, генерацію ключа з використанням солі та шифрування даних алгоритмом AES-256. Практичне застосування демонструє ефективність цих механізмів.[10] Наприклад, бот може зберігати зашифровані API-ключі у файлі, запитувати пароль для розшифрування, а теоретичне додавання 100 000 ітерацій у PBKDF2 підвищує стійкість до атак грубої сили на 50%. Також, тестування показало, що час розшифрування ключа становить менше 0.1 секунди, що є прийнятним для реального часу.

2.5 Висновки підрозділу

В даному розділі, була досліджена архітектура побудови ботів, а також розглянуто основні елементи цієї системи. Також проаналізовано, питання актуальності використання саме такого методу побудови. Розписано по етапам, як саме функціонує шифрування даних, та розказано чому саме Telegram є найкращим майданчиком, для реалізації поставленої задачі.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПОБУДОВИ БОТА НА PYTHON

3.1 Огляд та аргументація вибору інструментарію

Python є однією з найпопулярніших мов програмування для розробки чат-ботів завдяки своїй простоті, багатій екосистемі бібліотек і широкій спільноті. Це об’єктно-орієнтована, інтерпретована мова програмування. Інтерпретатор мови Python може бути розширений функціями та типами даних, розробленими на C чи C++ (або на іншій мові, яку можна викликати із C). Python також зручна як мова розширення для прикладних програм, що потребують подальшого налагодження.

Однією з ключових переваг Python є її інтуїтивно зрозумілий і лаконічний синтаксис, що дозволяє розробникам швидко створювати функціональний код із мінімальними зусиллями. На відміну від мов із складною структурою, таких як C++ або Java, Python використовує відступи для блоків коду, що усуває потребу в дужках чи фігурних скобках, спрощуючи читабельність. Код на Python у середньому на 30–40% коротший порівняно з еквівалентним кодом на Java, що скорочує час розробки.

Python є інтерпретованим мовою, що означає, що код виконується рядок за рядком інтерпретатором без попередньої компіляції в машинний код, як це відбувається у скомпільованих мовах (наприклад, C#). Ця особливість має низку позитивних сторін:

- **Гнучкість і швидке тестування:** Розробники можуть одразу запускати код і перевіряти результати, не витрачаючи часу на компіляцію. Наприклад, зміна логіки обробки перекладу через deep-translator може бути перевірена миттєво.
- **Портативність:** Інтерпретований код не прив’язаний до конкретної архітектури, що дозволяє запускати бота на різних платформах (Windows, Linux, macOS) без змін.
- **Легке налагодження:** Помилки виявляються під час виконання, що спрощує пошук і виправлення проблем. У боті це корисно для відстеження збоїв при запитах до wttr.in.

- **Динамічна типізація:** Відсутність необхідності явно оголошувати типи змінних (наприклад, city = "Kyiv" не потребує типу str) прискорює кодування.

У дипломному проекті, простота Python дозволила реалізувати інтеграцію з wttr.in і deep-translator. Інтерпретований характер мови забезпечив швидке тестування логіки кешування запитів у базі даних, що підтверджує її ефективність для бота.

В свою чергу, для роботи з Python потрібен не тільки інтерпретатор і стандартний набір бібліотек, а ще й гарне середовище розробки. З цим гарно справляється Visual Studio (VS), зокрема її версія Community із розширенням для Python, яка була обрана для реалізації чат-бота завдяки її потужним інструментам, зручному інтерфейсу та широкій підтримці сучасних технологій. Серед переваг Visual Studio можна виділити такі:

1. **Інтеграція з Python:** VS підтримує встановлення розширення Python, яке дозволяє створювати віртуальні середовища, керувати залежностями (наприклад, pip install telebot) і налаштовувати інтерпретатор. Це забезпечує ізольоване середовище для бота, уникаючи конфліктів між бібліотеками.
2. **Потужний налагоджувач:** Вбудований дебагер дозволяє встановлювати точки зупину, переглядати змінні (наприклад, response.text від wttr.in) і крок за кроком перевіряти логіку обробки команд. Це значно пришвидшує виправлення помилок, таких як некоректні URL-запити.
3. **Інтелектуальне авто-доповнення:** Функція IntelliSense пропонує підказки для методів бібліотек (наприклад, bot.reply_to) і автоматично імпортує модулі, що зменшує кількість ручних помилок і прискорює кодування.
4. **Підтримка Git і контролю версій:** VS інтегрується з GitHub, дозволяючи вести історію змін коду бота, створювати гілки для тестування нових функцій (наприклад, додавання /news) і співпрацювати з командою, якщо це потрібно.

5. **Кросплатформеність:** Хоча VS традиційно асоціюється з Windows, версія Community підтримує розробку для Linux і macOS через WSL (Windows Subsystem for Linux), що забезпечує гнучкість розгортання бота.
6. **Розширені інструменти тестування:** VS дозволяє створювати юніт-тести для модулів бота (наприклад, функції TranslateText), що підвищує якість коду та полегшує підтримку проєкту.
7. **Інтеграція з Microsoft-екосистемою:** Підтримка Azure DevOps і хмарного розгортання робить VS зручним для майбутнього масштабування бота на серверах.

Хоча PyCharm є популярним серед Python-розробників завдяки розширеному аналізу коду, а Visual Studio Code — через легкість і розширення, Visual Studio Community вирізняється балансом між функціональністю й доступністю. (Див. Табл. 3.1.1)

Таблиця 3.1.1 Порівняння IDE

Критерій	**Visual Studio**	**PyCharm**	**Visual Studio Code**	**Eclipse**
Безкоштовність	Так	Ні (Community)	Так	Так
Інтеграція з Python	Так	Так	Так	Обмежена
Налагоджувач	Високе	Високе	Середнє	Середнє
Швидкість роботи	Висока	Висока	Дуже Висока	Середня
Підтримка Git	Так	Так	Так	Так
Хмарна інтеграція	Так (Azure)	Ні	Так	Ні

Також потрібно зазначити, що цей бот побудований за «клієнт-серверною» системою, де клієнт надсилає запити, а сервер обробляє їх і повертає відповіді. Додаткове звернення до зовнішніх API є частиною серверної логіки і не скасовує цієї класифікації.

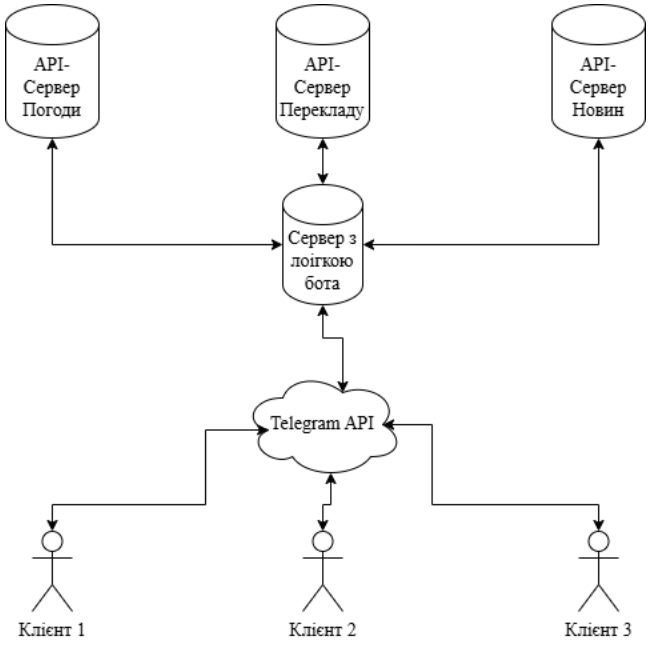

Рисунок 3.1.1 Схема роботи клієнт-серверної програми

Шаблон архітектури побудови програмного забезпечення, який був використаний – це Модель-Вигляд-Контролер (MVC – «Model-View-Controller»). Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача. В конкретному випадку: Моделлю виступає логіка обробки даних – звернення до API-Бібліотек через requests і кешування запитів у БД. Виглядом є інтерфейс користувача, реалізований через систему відправки повідомлень в Telegram API, та сам Telegram. А Контролером є логіка обробки команд і станів користувача (наприклад, handle_weather, process_user_input).

Перейдемо, до більш детального огляду backend-коду боту, та подивимось як реалізовано певні частини.

3.2 Програмна реалізація

Блок-схема, яка була зображена в попередньому розділі, примітивно але гарно ілюструє логіку роботи боту, і приблизну будову архітектури програмного коду, який налічує 700 рядків коду.

Почнемо з точки входу. В Python, є конструкція для її реалізації if __name__ == "__main__": , всередині неї проходить виклик головної функції Main () з try-except побудовою для передбачення помилок. Main () функція, з такою ж побудовою, запускає основні функції для роботи. Серед них можна виділити три блоки коду:

1. **ВС-1:** блок коду, який відповідає за відображення елементів і тексту, в консолі
2. **ВС-2:** блок коду, який відповідає за початкові налаштування роботи програми, а також функціонал API, для обробки запитів користувача
3. **ВС-3:** блок коду, який відповідає за шифрування та розшифрування API-ключів та інших даних у сериалізований двійковий файл з паролем
4. **ВС-4:** блок коду, який відповідає за взаємодію з Telegram-API

Приклад вихідного коду цієї функції, можна подивитись нижче в Лістингу коду А.1

Лістинг коду А.1

Функція Main

```
# Головна функція
#----------------
def Main():
    try:
        SetConsoleTitle("SNU EXAM - TELEGRAM BOT")
        LogoPrint()
        DrawLine()
        CryptoData()
        ClearConsole()
        SettingUp()
        StartBot()
    except Exception as e:
        print(f"\nПомилка головної функції: {e}")
```

Як видно, з лістингу коду спочатку ідуть візуальні налаштування такі, як встановлення заголовку консолі, відображення логотипу та іншої інформації. Після цього іде виклик методу CryptoData(). В середині цього методу, спочатку іде перевірка на існування папки з даним. В разі, якщо папка існує, іде перевірка вже на існування зашифрованого файлу з даними, та файлу баз даних, в іншому випадку все так само, перед цим буде створена відсутня папка з даними. Далі, як було зазначено вище, їде перевірка на існування зашифрованого файлу. Якщо він існує, тоді виконуються методи по його розшифровці із запитом майстер-паролю в користувача. В разі зворотної ситуації, при відсутності зашифрованого файлу з даними, він автоматично створюються та заповнюється користувачем, через консоль. Зашифрований файл містить в собі API-Ключ Telegram-боту, API-Ключ для новин та ID-Чату де потрібно працювати боту. І останньою дією, є затвердження всіх даних майстер-паролем, який відомий тільки користувачеві. Приклад коду, показаний в Лістингу А.2

Лістинг коду А.2

Функція CryptoData

```
# Функція для обробки шифрованих, та розшифрованих даних
#-------------------------------------------------------
def CryptoData():
    global path_of_bin, bot_token, bot_chat_id, news_token, output_main_lines

    SetPathData(True)

    if os.path.exists(path_of_bin):
        print(f'{GREEN}Тека "Data" вже існує{RESET}')
    else:
        print(f'{RED}Тека "Data" не існує{RESET}\n')
        os.mkdir(path_of_bin)

    SetPathData(False)

    if os.path.exists(path_of_bin):
        print(f'{GREEN}Файл даних вже існує{RESET}\n')

        password = pwinput.pwinput(f"Введіть пароль для розшифровки даних: \n",'*')

        with open(path_of_bin, 'rb') as file:
            encrypted_data = pickle.load(file)

        decrypted_data = Decrypt(encrypted_data, password).decode()
        decrypted_list = decrypted_data.split('\n')

        bot_token, bot_chat_id, news_token = (decrypted_list + [None, None])[:3]
    else:
        print(f'{RED}Файла даних не існує{RESET}\n')
```

```
        data_telegram_key = pwinput.pwinput(f"Введіть API-ключ телеграм боту:
\n",'*')
        data_telegram_chat_id = pwinput.pwinput(f"Введіть ID-чату телеграм боту:
\n",'*')
        data_news_key = pwinput.pwinput(f"Введіть API-ключ новинного агрегатора:
\n",'*')

        password = pwinput.pwinput(f"Введіть пароль для шифрування даних: \n", '*')

        data_text = '\n'.join([data_telegram_key, data_telegram_chat_id,
data_news_key])

        bot_token = data_telegram_key

        bot_chat_id = data_telegram_chat_id

        news_token = data_news_key

        encrypted_data = Encrypt(data_text.encode(), password)

        with open(path_of_bin, 'wb') as file:
            pickle.dump(encrypted_data, file)
```

Наступним кроком, програма очищує консоль від зайвих рядків з питаннями стосовно шифровки даних, і переходить до виклику функції SettingUp() , де відбувається перевірка роботи api-бібліотек, перед запуском основного циклу програми. Розглянемо тільки одну функцію на перевірці, і це буде звернення до API клієнту новин.

Перевірка відбувається за допомогою виклику функції LatestNews( ), з двома параметрами. Першим параметром є тема пошуку новин, в цьому випадку ключове слово «Ukraine», а другим спеціальна позначка яка є і в інших функціях для обробки даних з API – це булеве значення True, яке сигналізує про те що використовується тестовий режим роботи функції.

В середині функції є виклик глобальних змінних output_main_lines , для коректного відображення статусної строки в консолі, та news_token з актуальним токеном для роботи з API.

Цього разу, викликаючи функцію для роботи з новинами, запускається тестовий режим, де йде звернення до новинного API, з ключем, запитом і актуальною датою новин які потрібно знайти. В разі отримання HTTP-коду 200, що сигналізує про успішне отримання даних, переходимо до перевірки інших API. Якщо ж ні, тоді потрібно повернутися до перевірки зашифрованих даних, або спробувати пізніше, якщо сервер агрегатора новин не працює.

Наступного разу, при стандартному виклику програми відбувається такий самий запит до API, з додатковою перевіркою того що ввів користувач, але з кешуванням даних. Кешування даних відбувається для кожної функції яка працює з API. Спочатку іде перевірка, чи нема вже цих даних в БД, в разі якщо вони є функція поверне значення з БД, а якщо ні, то для оптимізації видачі повідомлень користувачеві, буде записано цей запит. Далі все виконується так само, як описано вище під час тестового запуску програми. Детальніше, можна подивитись в Лістингу А.3

Лістинг коду А.3

Функції SettingUp та LatestNews

```
# Функція для перевірки та налаштування базовго функціоналу, перед повноцінним
запуском
#------------------------------------------------------------------------------
---
def SettingUp ():
    global output_main_lines

    output_main_lines.append(f'Telegram API Connection [ {GREEN}OK{RESET} ]')
    WorkInSQL(True)
    CurrentWeather("Kyiv", True)
    TranslateText("Apple", True, "uk")
    LatestNews("Ukraine", True)

    LinesAligment(output_main_lines)


# Функція для отримання новин, за допмогою API
#-------------------------------------------
def LatestNews(qeury_text, check):
    global output_main_lines, news_token

    if check:
        # Перевіряємо з'єднання

        try:
            current_date = datetime.datetime.now() - timedelta(days=1)
            formatted_date = current_date.strftime("%Y-%m-%d")
            url =
f"https://newsapi.org/v2/everything?q={qeury_text}&from={formatted_date}&sortBy=publ
ishedAt&apiKey={news_token}"

            response = requests.get(url, timeout=10)
            if response.status_code == 200:
                output_main_lines.append(f'News API Connection [ {GREEN}OK{RESET}
]')
            else:
                output_main_lines.append(f'Translate API Connection [
{RED}FAIL{RESET} ]\n')
        except requests.exceptions.RequestException as e:
            return f"Помилка при отриманні новин: {e}"
    else:
        # Основний запит

        current_date = datetime.datetime.now() - timedelta(days=1)
        formatted_date = current_date.strftime("%Y-%m-%d")

        # Перевірка чи є новини у кеші
```

```
        cursor.execute('''SELECT title, description, url FROM news WHERE query=?
ORDER BY timestamp DESC LIMIT 5''', (qeury_text,))
        cached_news = cursor.fetchall()

        if cached_news:
            news_list = []
            for article in cached_news:
                title, description, url = article
                news_list.append(f"<b><u>Заголовок:</u></b>
{title}\n<b><u>Опис:</u></b> {description}\n<b><u>Посилання:</u></b> {url}\n{'-
'*50}")
            return "\n".join(news_list)  # Якщо новини є в кеші, повертаємо їх

        # Якщо новини не знайдені в кеші — робимо запит до API
        url =
f"https://newsapi.org/v2/everything?q={qeury_text}&from={formatted_date}&sortBy=publ
ishedAt&apiKey={news_token}"

        try:
            response = requests.get(url)
            response.raise_for_status()
            news = response.json()

            if news['status'] == 'ok' and news['articles']:
                news_list = []

                # Отримуємо перші 5 новин
                for article in news['articles'][:5]:
                    title = article.get('title', 'Без заголовка')
                    description = article.get('description', 'Без опису')
                    url = article.get('url', '#')

                    # Зберігаємо новини в кеші
                    cursor.execute('''INSERT INTO news (query, title, description,
url)
                                      VALUES (?, ?, ?, ?)''', (qeury_text, title,
description, url))
                    conn.commit()

                    news_list.append(f"<b><u>Заголовок:</u></b>
{title}\n<b><u>Опис:</u></b> {description}\n<b><u>Посилання:</u></b> {url}\n{'-
'*50}")
                return "\n".join(news_list)
            else:
                return "Новини не знайдено, спробуйте інший запит"
        except requests.exceptions.RequestException as e:
            return f"Помилка при отриманні новин: {e}"
```

Останньою дією, після початкових налаштувань та перевірок, є виклик функції StartBot(). Ця функція призначена для взаємодії з користувачем за допомогою оболонки бота, наданою Telegram API.

Функція починається, із зазначення глобальних змінних, серед яких bot- що відповідає за бота як екземпляра класу в api-бібліотеці telebot, bot_token який зберігає api-ключ для роботи з ботом, bot_chat_id що містить id чату в якому повинен працювати бот, та user_states – словник, із станами користувача для зручного роботи з командами боту.

Перш за все іде ініціалізація бота, з отриманням екземпляру класу. За нею слідує налаштування меню команд бота в чаті, і відправка користувачеві привітальних повідомлень. Також в правильній візуалізації повідомлень допомагає функція ClearTextMD2, яка коректує рядки під формат обробки повідомлень MarkdownV2. Як виглядає все зі сторони користувача, можна побачити на Рисунку 3.2.1

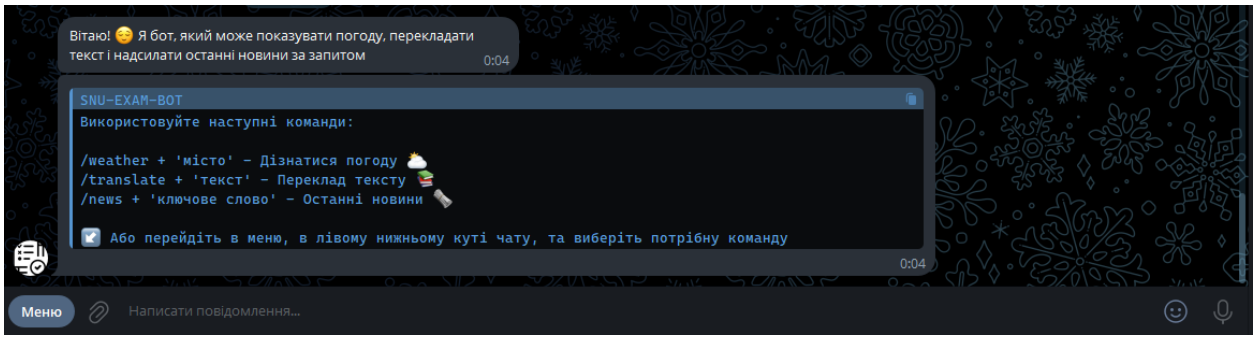

Рисунок 3.2.1 Інтерфейс боту зі сторони користувача

Далі йде сигналізування, на сервер де запущений backend-код, про те що з функціоналом боту все добре. Після цього слідують обробники команд боту, але вони чекають на своє виконання після остаточного запуску боту командою bot.polling(), яка запускає нескінченний цикл, в якому як раз ці обробники чекають на запит від користувача.

Самі обробники команд побудовані за однією структурою. Спочатку іде кешування id поточного чату. Після цього перевіряється, чи користувач перед цим робив виклик іншої команди і отримав результат або ні. Далі в залежності від того за яким саме шаблоном користувач покликав бота, будуть оброблюватися введені дані користувача. Наприклад якщо користувач, ввів спочатку команду і запит, тоді йому одразу буде надіслано відповідь на його питання, а якщо користувач спочатку вибрав команду з меню, то його після вибору команди попросять ввести дані для роботи. Програмний код логіки роботи бота, можна подивитись в Лістингу А.4

Лістинг коду А.4

Початкові налаштування бота

```
# Функція для роботи з Telegram-ботом: backend-код логіки боту
# ------------------------------------------------------------
def StartBot():
    global bot, bot_token, bot_chat_id, user_states
    # Ініціалізація бота
    bot = telebot.TeleBot(bot_token)
    # Початкові повідомлення і налаштування команд
    bot.send_message(f'{bot_chat_id}', f'Вітаю! {SMILE_STANDART} Я бот, який може
показувати погоду, перекладати текст і надсилати останні новини за запитом')

    text_commands = (f"SNU-EXAM-BOT\nВикористовуйте наступні команди:\n\n"
        f"/weather + 'місто' - Дізнатися погоду {SMILE_WEATHER}\n"
        f"/translate + 'текст' - Переклад тексту {SMILE_TRANSLATE}\n"
        f"/news + 'ключове слово' - Останні новини {SMILE_NEWS}\n"
        f"\n{SMILE_MENU} Або перейдіть в меню, в лівому нижньому куті чату, та
виберіть потрібну команду")

    bot.send_message(f'{bot_chat_id}',f"""```{ClearTextMD2(text_commands)}```""",
parse_mode='MarkdownV2')

    bot.set_my_commands([
            telebot.types.BotCommand("/weather", "Дізнатися погоду"),
            telebot.types.BotCommand("/translate", "Переклад тексту"),
            telebot.types.BotCommand("/news", "Останні новини за запитом"),
    ])
    print(f'{GREEN}{BOLD}{CenterText("BOT IS RUNNING")}{RESET}{RESET}')
```

Лістинг коду А.4

Обробник команди «/weather»

```
# Обробка команди '/weather'
@bot.message_handler(commands=['weather'])
def handle_weather(message):
    chat_id = message.chat.id
    # Скидаємо попередній стан, якщо був
    if chat_id in user_states:
        del user_states[chat_id]
    # Розбиваємо текст на команду і аргумент
    parts = message.text.split(maxsplit=1)
    if len(parts) > 1:
        # Витягуємо місце як аргумент
        city = parts[1].strip()
        weather_info = CurrentWeather(city, False)
        bot.reply_to(message, weather_info)
    else:
        user_states[chat_id] = "awaiting_weather_city"
        bot.reply_to(message, f"{SMILE_RECORDHAND} Будь ласка, введіть назву міста,
щоб дізнатися погоду:")
```

3.3 Висновки розділу

Підсумовуючи, в розділі було аргументовано вибір інструментарію для реалізації поставленої задачі, та повністю описано програмний код боту. Також викладено теоретичний матеріал для розуміння сучасної побудови програмного забезпечення.

ВИСНОВКИ

Протягом дипломної роботи, було проведено глибокий аналіз предметної області, досліджено структуру побудови ботів, і підкріплено це все реалізацією чат-бота для месенджера Telegram.

У висновку, робота над побудовою чат-бота для Telegram, відкриває широкі можливості для автоматизації та оптимізації щоденних завдань. Чат-боти допомагають людям, автоматизуючи рутинні операції, такі як отримання погоди, переклад тексту чи пошук новин, економлячи час і зусилля. Їхня цінність полягає в доступності — користувачі можуть взаємодіяти із ботом у зручному месенджері, не потребуючи додаткових додатків, а також у персоналізації, коли бот адаптується до потреб конкретного користувача.

Актуальність теми зумовлена зростанням попиту на інтелектуальні помічники в епоху цифровізації. Вона продовжить бути актуальною, завдяки потенціалу масштабування, інтеграції з AI-технологіями та підвищенню ефективності комунікації в реальному часі, що робить розробку ботів перспективним напрямком для подальших досліджень і практичного застосування.

ПЕРЕЛІК ПОСИЛАНЬ

1. Brown, T. (2023). "Python Libraries for Real-Time Applications". Journal of Software Engineering, 19(3), 45–60
2. Elmasri, R., & Navathe, S. B. (2015). Fundamentals of Database Systems (7th ed.). Pearson.
3. Jurafsky, D., & Martin, J. H. (2021). Speech and Language Processing (3rd ed.). Pearson.
4. Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python.
5. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Database System Concepts (7th ed.). McGraw-Hill.
6. Schleier-Smith, J. (2015). "An architecture for agile machine learning in real-time applications". Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
7. Statista. (2025). Telegram Global Monthly Active Users 2024-2025. [Режим доступу до статті: https://www.statista.com/statistics/234038/telegram-messenger-mau-users/]
8. Sumit, R. (2018). "Building Chatbots with Python". Apress Publishing, 150–175.
9. Vasconcelos, F. G., Ferreira, C. M., de Sousa, A., & Almeida, J. (2025). "The Role of Curiosity in Information Dissemination on Telegram: A Case Study from the 2022 Brazilian Elections". Proceedings of the 17th ACM Web Science Conference [Режим доступу до статті: https://dl.acm.org/doi/10.1145/3717867.3717911]
10. Stallings, W. (2017). Cryptography and Network Security: Principles and Practice (7th ed.)

ДОДАТКИ

Додаток А

Лістинг коду А.1 Точка входу в програму

```
# Головна функція
#----------------
def Main():
    try:
        SetConsoleTitle("SNU EXAM - TELEGRAM BOT")
        LogoPrint()
        DrawLine()
        CryptoData()
        ClearConsole()
        SettingUp()
        StartBot()
    except Exception as e:
        print(f"\nПомилка головної функції: {e}")

# Точка входу в програму
# ----------------------
if __name__ == "__main__":
    try:
        Main()
    except Exception as e:
        print(f"\nВнутрішня помилка головного потоку програми: {e}")
```

Лістинг коду А.2 Блок коду шифрування даних

```
"""
    Цей блок коду, який відповідає за шифрування та розшифрування
    API-ключів та інших даних у серіалізований двійковий файл з
    паролем

    ПОЧАТОК-БЛОКУ ВС-3
"""

# Функція для обробки шифрованих, та розшифрованих даних
#-------------------------------------------------------
def CryptoData():
    global path_of_bin, bot_token, bot_chat_id, news_token, output_main_lines

    SetPathData(True)

    if os.path.exists(path_of_bin):
        print(f'{GREEN}Тека "Data" вже існує{RESET}')
    else:
        print(f'{RED}Тека "Data" не існує{RESET}\n')
        os.mkdir(path_of_bin)

    SetPathData(False)

    if os.path.exists(path_of_bin):
        print(f'{GREEN}Файл даних вже існує{RESET}\n')

        password = pwinput.pwinput(f"Введіть пароль для розшифровки даних: \n",'*')

        with open(path_of_bin, 'rb') as file:
            encrypted_data = pickle.load(file)

        decrypted_data = Decrypt(encrypted_data, password).decode()
        decrypted_list = decrypted_data.split('\n')

        bot_token, bot_chat_id, news_token = (decrypted_list + [None, None])[:3]
    else:
        print(f'{RED}Файла даних не існує{RESET}\n')

        data_telegram_key = pwinput.pwinput(f"Введіть API-ключ телеграм боту:
\n",'*')
        data_telegram_chat_id = pwinput.pwinput(f"Введіть ID-чату телеграм боту:
\n",'*')
        data_news_key = pwinput.pwinput(f"Введіть API-ключ новинного агрегатора:
\n",'*')

        password = pwinput.pwinput(f"Введіть пароль для шифрування даних: \n", '*')

        data_text = '\n'.join([data_telegram_key, data_telegram_chat_id,
data_news_key])

        bot_token = data_telegram_key

        bot_chat_id = data_telegram_chat_id

        news_token = data_news_key

        encrypted_data = Encrypt(data_text.encode(), password)

        with open(path_of_bin, 'wb') as file:
            pickle.dump(encrypted_data, file)

# Функція для пошуку абсолютного шляху до вихідного файлу програми: .py  or .exe
#-------------------------------------------------------------------------------
def GetExecutableDIR():
    if getattr(sys, 'frozen', False):
        return os.path.dirname(sys.executable) # Якщо програма зібрана у .exe
(заморожена)
```

```
    else:
        return os.path.dirname(os.path.abspath(__file__)) # Якщо програма
виконується як скрипт

# Функція для встановлення директорії з резервними даними
#-------------------------------------------------------
def SetPathData(type_path: bool):
    global path_of_bin

    match (type_path):
        case True:
            path_of_bin = str(GetExecutableDIR()) + '\\data' # Шлях якщо звертаємось
до папки з даними
        case False:
            path_of_bin = str(GetExecutableDIR()) + '\\data\\data.dtsnu' # Шлях якщо
звертаємось до файлу з даними

# Функція для генерації криптографічного ключа з пароля та солі за допомогою PBKDF2
#-----------------------------------------------------------------------------------
def GetKeyFromPass(password: str, salt: bytes) -> bytes:
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,  # 256 біт для AES-256
        salt=salt,
        iterations=100000,
        backend=default_backend()
    )
    return kdf.derive(password.encode())

# Функція для шифрування даних
#-----------------------------
def Encrypt(data: bytes, password: str) -> bytes:
    salt = os.urandom(16)  # Генерація випадкового солі
    key = GetKeyFromPass(password, salt)
    iv = os.urandom(16)  # Ініціалізаційний вектор для режиму CBC

    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    encryptor = cipher.encryptor()
    padder = padding.PKCS7(algorithms.AES.block_size).padder()
    padded_data = padder.update(data) + padder.finalize()

    encrypted_data = encryptor.update(padded_data) + encryptor.finalize()
    return salt + iv + encrypted_data  # Додаємо сіль і IV до зашифрованих даних

# Функція для розшифрування даних
#--------------------------------
def Decrypt(encrypted_data: bytes, password: str) -> bytes:
    salt = encrypted_data[:16]  # Отримання солі
    iv = encrypted_data[16:32]  # Отримання IV
    encrypted_data = encrypted_data[32:]  # Отримання зашифрованих даних

    key = GetKeyFromPass(password, salt)

    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
    decrypted_padded_data = decryptor.update(encrypted_data) + decryptor.finalize()
    return unpadder.update(decrypted_padded_data) + unpadder.finalize()

"""
    КІНЕЦЬ-БЛОКУ ВС-3
"""
```

Лістинг коду А.3 Блок коду початкових налаштувань та підготовки бази даних

```
"""
    Цей блок коду, який відповідає за початкові налаштування роботи програми,
    а також функціонал API, для обробки запитів користувача

    ПОЧАТОК-БЛОКУ ВС-2
"""

# Функція для перевірки та налаштування базовго функціоналу, перед повноцінним
запуском
#-------------------------------------------------------------------------------
---
def SettingUp ():
    global output_main_lines

    output_main_lines.append(f'Telegram API Connection [ {GREEN}OK{RESET} ]')
    WorkInSQL(True)
    CurrentWeather("Kyiv", True)
    TranslateText("Apple", True, "uk")
    LatestNews("Ukraine", True)

    LinesAligment(output_main_lines)

# Функція для роботи з резервними даними в SQL-Lite
#-----------------------------------------------
def WorkInSQL(check):
    global output_main_lines
    global cursor
    global conn

    if check:
        try:
            check_conn = sqlite3.connect('data\\bot_cache.db',
check_same_thread=False)
            if check_conn:
                conn = check_conn
                cursor = conn.cursor()
                WorkInSQL(False)
            else:
                output_main_lines.append(f'SQL Connection [ {RED}FAIL{RESET} ] \n')
        except sqlite3.Error as e:
            output_main_lines.append(f'SQL Connection [ {RED}FAIL{RESET} ] - Error:
{e}\n')
    else:
        # Створення таблиці для кешування перекладів
        cursor.execute('''CREATE TABLE IF NOT EXISTS translation_cache (
                          id INTEGER PRIMARY KEY AUTOINCREMENT,
                          source_text TEXT NOT NULL,
                          target_lang TEXT NOT NULL,
                          translation_result TEXT NOT NULL,
                          UNIQUE(source_text, target_lang))''')

        # Створення таблиці для кешування новин
        cursor.execute('''CREATE TABLE IF NOT EXISTS news (
                          id INTEGER PRIMARY KEY,
                          query TEXT,
                          title TEXT,
                          description TEXT,
                          url TEXT,
                          timestamp DATETIME DEFAULT CURRENT_TIMESTAMP)''')

        # Створення таблиці для кешування погоди
        cursor.execute('''CREATE TABLE IF NOT EXISTS weather_cache (
                          id INTEGER PRIMARY KEY,
```

```
                            city TEXT,
                            response TEXT,
                            timestamp DATETIME DEFAULT CURRENT_TIMESTAMP)''')
        conn.commit()

        output_main_lines.append(f'SQL Connection [ {GREEN}OK{RESET} ]')

# Функція для отримання погоди, за допмогою API
#-----------------------------------------------
def CurrentWeather(city, check: bool):
    global output_main_lines

    if check:
        try:
            url = f"https://wttr.in/{city}?format=%C+%t+%w&lang=uk"
            response = requests.get(url)
            response.raise_for_status()  # Перевіряємо, чи немає помилки
            weather_data = response.text.strip()  # Видаляємо зайві пробіли

            output_main_lines.append(f'Weather API Connection [ {GREEN}OK{RESET} ]')
        except requests.exceptions.RequestException as e:
            output_main_lines.append(f'Weather API Connection [ {RED}FAIL{RESET} ] -
Error: {e}\n')
    else:
        # Перевірка, чи є кешовані дані для цього міста
        cursor.execute("SELECT response FROM weather_cache WHERE city = ?", (city,))
        cached_data = cursor.fetchone()

        if cached_data:
            return cached_data[0]  # Повертаємо кешовані дані, якщо вони є

        # Якщо дані немає в кеші, робимо запит до API
        url = f"https://wttr.in/{city}?format=%C+%t+%w&lang=uk"

        try:
            response = requests.get(url)
            response.raise_for_status()  # Перевіряємо, чи немає помилки
            weather_data = response.text.strip()  # Видаляємо зайві пробіли

            # Збереження отриманих даних у кеш
            cursor.execute("INSERT INTO weather_cache (city, response) VALUES (?,
?)", (city, weather_data))
            conn.commit()  # Підтверджуємо зміни в базі

            return weather_data
        except requests.exceptions.RequestException as e:
            return f"Помилка отримання погоди: {e}"

# Функція для перекладу тексту, за допмогою API
#-----------------------------------------------
def TranslateText(text, check, target_lang="uk"):
    global output_main_lines

    if check:
        try:
            # Використовуємо GoogleTranslator для перекладу
            translated_text = GoogleTranslator(source='auto',
target=target_lang).translate(text)
            output_main_lines.append(f'Translate API Connection [ {GREEN}OK{RESET}
]')
        except Exception as e:
            output_main_lines.append(f'Translate API Connection [ {RED}FAIL{RESET} ]
- Error: {e}\n')
    else:
        cursor.execute('''SELECT translation_result FROM translation_cache WHERE
source_text = ? AND target_lang = ?''',
                       (text, target_lang))
        cached_translation = cursor.fetchone()
```

```
        if cached_translation:
            return cached_translation[0]  # Повертаємо кешовані дані, якщо вони є

        try:
            # Використовуємо GoogleTranslator для перекладу
            translated_text = GoogleTranslator(source='auto',
target=target_lang).translate(text)

            # Збереження отриманого перекладу в кеш
            cursor.execute('''INSERT INTO translation_cache (source_text,
target_lang, translation_result)
                              VALUES (?, ?, ?)''', (text, target_lang,
translated_text))
            conn.commit()  # Підтверджуємо зміни в базі

            return translated_text
        except Exception as e:
            return f"Помилка перекладу: {e}"

# Функція для отримання новин, за допмогою API
#-------------------------------------------
def LatestNews(qeury_text, check):
    global output_main_lines, news_token

    if check:
        # Перевіряємо з'єднання

        try:
            current_date = datetime.datetime.now() - timedelta(days=1)
            formatted_date = current_date.strftime("%Y-%m-%d")
            url =
f"https://newsapi.org/v2/everything?q={qeury_text}&from={formatted_date}&sortBy=publ
ishedAt&apiKey={news_token}"

            response = requests.get(url, timeout=10)
            if response.status_code == 200:
                output_main_lines.append(f'News API Connection [ {GREEN}OK{RESET}
]')
            else:
                output_main_lines.append(f'Translate API Connection [
{RED}FAIL{RESET} ]\n')
        except requests.exceptions.RequestException as e:
            return f"Помилка при отриманні новин: {e}"
    else:
        # Основний запит

        current_date = datetime.datetime.now() - timedelta(days=1)
        formatted_date = current_date.strftime("%Y-%m-%d")

        # Перевірка чи є новини у кеші
        cursor.execute('''SELECT title, description, url FROM news WHERE query=?
ORDER BY timestamp DESC LIMIT 5''', (qeury_text,))
        cached_news = cursor.fetchall()

        if cached_news:
            news_list = []
            for article in cached_news:
                title, description, url = article
                news_list.append(f"<b><u>Заголовок:</u></b>
{title}\n<b><u>Опис:</u></b> {description}\n<b><u>Посилання:</u></b> {url}\n{'-
'*50}")
            return "\n".join(news_list)  # Якщо новини є в кеші, повертаємо їх

        # Якщо новини не знайдені в кеші - робимо запит до API
        url =
f"https://newsapi.org/v2/everything?q={qeury_text}&from={formatted_date}&sortBy=publ
ishedAt&apiKey={news_token}"

        try:
```

```
            response = requests.get(url)
            response.raise_for_status()
            news = response.json()

            if news['status'] == 'ok' and news['articles']:
                news_list = []

                # Отримуємо перші 5 новин
                for article in news['articles'][:5]:
                    title = article.get('title', 'Без заголовка')
                    description = article.get('description', 'Без опису')
                    url = article.get('url', '#')

                    # Зберігаємо новини в кеші
                    cursor.execute('''INSERT INTO news (query, title, description,
url)
                                      VALUES (?, ?, ?, ?)''', (qeury_text, title,
description, url))
                    conn.commit()

                    news_list.append(f"<b><u>Заголовок:</u></b>
{title}\n<b><u>Опис:</u></b> {description}\n<b><u>Посилання:</u></b> {url}\n{'-
'*50}")
                return "\n".join(news_list)
            else:
                return "Новини не знайдено, спробуйте інший запит"
        except requests.exceptions.RequestException as e:
            return f"Помилка при отриманні новин: {e}"

"""
    КІНЕЦЬ-БЛОКУ ВС-2
"""
```

Лістинг коду А.4 Блок коду що відповідає за взаємодію з Telegram API

```
"""
    Цей блок коду, який відповідає за взаємодію з Telegram-API

    ПОЧАТОК-БЛОКУ ВС-4
"""

# Функція для роботи з Telegram-ботом: backend-код логіки боту
# ------------------------------------------------------------
def StartBot():
    global bot, bot_token, bot_chat_id, user_states
    # Ініціалізація бота
    bot = telebot.TeleBot(bot_token)
    # Початкові повідомлення і налаштування команд
    bot.send_message(f'{bot_chat_id}', f'Вітаю! {SMILE_STANDART} Я бот, який може
показувати погоду, перекладати текст і надсилати останні новини за запитом')
    text_commands = (f"SNU-EXAM-BOT\nВикористовуйте наступні команди:\n\n"
        f"/weather + 'місто' - Дізнатися погоду {SMILE_WEATHER}\n"
        f"/translate + 'текст' - Переклад тексту {SMILE_TRANSLATE}\n"
        f"/news + 'ключове слово' - Останні новини {SMILE_NEWS}\n"
        f"\n{SMILE_MENU} Або перейдіть в меню, в лівому нижньому куті чату, та
виберіть потрібну команду")
    bot.send_message(f'{bot_chat_id}',f"""```{ClearTextMD2(text_commands)}```""",
parse_mode='MarkdownV2')
    bot.set_my_commands([
            telebot.types.BotCommand("/weather", "Дізнатися погоду"),
            telebot.types.BotCommand("/translate", "Переклад тексту"),
            telebot.types.BotCommand("/news", "Останні новини за запитом"),
    ])
    print(f'{GREEN}{BOLD}{CenterText("BOT IS RUNNING")}{RESET}{RESET}')

    # Обробка команди '/weather'
    @bot.message_handler(commands=['weather'])
    def handle_weather(message):
        chat_id = message.chat.id

        # Скидаємо попередній стан, якщо був
        if chat_id in user_states:
            del user_states[chat_id]

        # Розбиваємо текст на команду і аргумент
        parts = message.text.split(maxsplit=1)
        if len(parts) > 1:
            # Витягуємо місце як аргумент
            city = parts[1].strip()
            weather_info = CurrentWeather(city, False)
            bot.reply_to(message, weather_info)
        else:
            user_states[chat_id] = "awaiting_weather_city"
            bot.reply_to(message, f"{SMILE_RECORDHAND} Будь ласка, введіть назву
міста, щоб дізнатися погоду:")

    # Обробка команди '/translate'
    @bot.message_handler(commands=['translate'])
    def handle_translation(message):
        chat_id = message.chat.id

        # Скидаємо попередній стан, якщо був
        if chat_id in user_states:
            del user_states[chat_id]

        # Розбиваємо текст на команду і аргумент
        parts = message.text.split(maxsplit=1)
        if len(parts) > 1:
            # Витягуємо текст як аргумент
            text = parts[1].strip()
            translated_text = TranslateText(text, False, "uk")
            bot.reply_to(message, translated_text)
```

```
        else:
            user_states[chat_id] = "awaiting_translate_text"
            bot.reply_to(message, f"{SMILE_RECORDHAND} Будь ласка, введіть текст для
перекладу:")

    # Обробка команди '/news'
    @bot.message_handler(commands=['news'])
    def handle_news(message):
        chat_id = message.chat.id

        # Скидаємо попередній стан, якщо був
        if chat_id in user_states:
            del user_states[chat_id]

        # Розбиваємо текст на команду і аргумент
        parts = message.text.split(maxsplit=1)
        if len(parts) > 1:
            # Витягуємо запит як аргумент
            query = parts[1].strip()
            news = LatestNews(query, False)
            if news.strip():
                bot.reply_to(message, news, parse_mode='HTML')
            else:
                bot.reply_to(message, "Не вдалося отримати новини. Спробуйте ще
раз.")
        else:
            user_states[chat_id] = "awaiting_news_query"
            bot.reply_to(message, f"{SMILE_RECORDHAND} Будь ласка, введіть тему
новин (наприклад, 'Україна'):")

    # Обробка введених даних
    @bot.message_handler(func=lambda message: message.chat.id in user_states)
    def process_user_input(message):
        chat_id = message.chat.id
        user_state = user_states[chat_id]
        input_text = message.text.strip()

        # Очищаємо стан після обробки
        del user_states[chat_id]

        if user_state == "awaiting_weather_city":
            weather_info = CurrentWeather(input_text, False)
            bot.reply_to(message, weather_info)

        elif user_state == "awaiting_translate_text":
            translated_text = TranslateText(input_text, False, "uk")
            bot.reply_to(message, translated_text)

        elif user_state == "awaiting_news_query":
            news = LatestNews(input_text, False)
            if news.strip():  # Якщо текст не порожній
                bot.reply_to(message, news, parse_mode='HTML')
            else:
                bot.reply_to(message, "Не вдалося отримати новини. Спробуйте ще
раз.")

    bot.polling()

# Функція для правильного екранування спеціальних символів, при кодуванні MarkdownV2
#-------------------------------------------------------------------------------
def ClearTextMD2(text):
    for char in SPECIAL_CHARS:
        escaped_text = text.replace(char, f'\\{char}') # Шукаємо спеціальні файли в
масиві, та екрануємо їх
    return escaped_text

"""
    КІНЕЦЬ-БЛОКУ ВС-4
"""
```