ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ЧАТ-БОТУ ДЛЯ МОНІТОРИНГУ СТАНУ НАВКОЛИШНЬОГО СЕРЕДОВИЩА МОВОЮ PYTHON**»

Виконав: студент 4 курсу, групи ПД-44
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності/спеціалізації)

Гаврилюк В.О.
(прізвище та ініціали)

Керівник Трінтіна Н.А
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

Київ –2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____Негоденко О.В.
"____" _____2022 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

ГАВРИЛЮКА ВАЛЕРІЯ ОЛЕКСАНДРОВИЧА
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка чат-боту для моніторингу стану навколишнього середовища мовою Python»

Керівник роботи: Трінтіна Н.А., к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «18» лютого 2022 року №?.

2. Строк подання студентом роботи «3» червня 2022 року

3. Вхідні дані до роботи:

3.1 Науково-технічна література, пов'язана з розробкою ботів
3.2 Алгоритм дії бота
3.3 Aiogram Bot API

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).

4.1. Аналіз обов'язків розроблюваного бота
4.2. Аналіз та дослідження існуючих аналогів
4.3. Розробка програмного забезпечення

4.4. Тестування розробленого програмного забезпечення

5. Перелік демонстраційного матеріалу
   5.1 Мета, об`єкт та предмет дослідження
   5.2 Аналоги
   5.3 Технічне завдання
   5.4 Програмні засоби реалізації
   5.5 Алгоритм подання нових заяв
   5.6 Алгоритм перегляду та опрацювання існуючих заяв
   5.7 Алгоритм модерації заяв
   5.8 Алгоритм роботи бонусного магазину
   5.9 Апробація результатів дослідження
   5.10 Висновки

6. Дата видачі завдання «11» квітня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	04.04.22 - 11.04.22	Виконано
2	Аналіз та дослідження існуючих аналогів	11.04.22 - 15.04.22	Виконано
3	Дослідження програмних засобів	15.04.22 - 21.04.22	Виконано
4	Моделювання об’єкту проектування	21.04.22 - 28.04.22	Виконано
5	Розробка функціоналу бота	28.04.22 - 03.05.22	Виконано
6	Вступ, висновки, реферат	03.05.22 - 07.05.22	Виконано
7	Розробка обов‘язкових демонстраційних матеріалів	07.05.22 - 10.05.22	Виконано
8	Попередній захист роботи	25.05.22	
9	Здача роботи	03.06.22	

Студент _ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 60с., 29 рис., 5 табл., 25 джерел.

PYTHON, TELEGRAM, PYCHARM, AIOGRAM, ЧАТ-БОТ, POSTGRESQL, МЕСЕНДЖЕР, GOOGLE CLOUD PLATFORM.

Об'єкт дослідження – способи та методи створення чат-боту на базі месенджера Telegram.

Предмет дослідження – телеграм чат-бот для забезпечення моніторингу стану навколишнього середовища.

Мета роботи – розробка комплексного чат-бота для забезпечення моніторингу стану навколишнього середовища за допомогою мови Python.

Методи дослідження – методи проектування та розробки програмного забезпечення, методи опрацювання та аналізу отриманих результатів, методи тестування програмного забезпечення.

Наукова новизна – розроблений алгоритм для подання та опрацювання заявок стану навколишнього середовища на певних ділянках.

В роботі розглянуті та проаналізовані відомості про існуючі аналоги програмного забезпечення для моніторингу стану навколишнього середовища. Досліджено сервіси та методи для створення чат-ботів. Розроблено принцип представлення отриманої інформації для кінцевого користувача.

Галузь використання – організації із захисту та контролю навколишнього середовища

ЗМІСТ

ВСТУП

Забруднення визначається як внесення в навколишнє середовище шкідливих забруднювачів, які негативно впливають на навколишнє середовище. Масштабне забруднення навколишнього середовища почалося з промисловою революцією і з тих пір не сповільнилося. Оскільки кількість населення та економіки продовжує зростати з роками, пропорційно зростає і забруднення. Це створило серйозну глобальну проблему, яка продовжує впливати на біорізноманіття, екосистему та здоров'я людей у всьому світі. Проте ми проживаємо у час інформаційних технологій і вирішення глобальної проблеми забруднення навколишнього середовища можна пришвидшити.

В наш час майже у кожної людини у власності присутній смартфон, в більшості з яких встановлений месенджер, який можна використовувати не лише для спілкування з іншими людьми, а й користуватись більш обширним функціоналом, що реалізується за допомогою чат-ботів.

Чат-бот – це програмне забезпечення, що використовується для ведення чату в месенджері за допомогою текстових повідомлень, замість прямого спілкування з живою людиною. Функціонал чат-боту може бути різноманітним, який починається від імітації спілкування з людиною до відправлення рекомендацій щодо інвестування або актуального прогнозу погоди.[1]

Популярність та актуальність чат-ботів збільшується з кожним днем, через свою зручність і практичність використання обраних рішень, адже їх використання не потребує від людей переходити на сторонні джерела або завантажувати будь - яке програмне забезпечення.

Підводячи підсумки проблеми, наведеної вище, щоб прискорити усунення проблеми забруднення навколишнього середовища та замотивувати суспільство займатись цим, було вирішено створити чат-бот моніторингу стану навколишнього середовища на платформі Telegram.

Була поставлена мета створити чат-бота, який дає змогу завантажувати знайдені місця забруднення навколишнього середовища за допомогою відправки

фото-файлу та свого місцезнаходження. Також переглядати та мати можливість усувати місця забруднення, що були додані іншими користувачами.

Опираючись на поставлену мету, були поставлені наступні задачі:

- аналіз обраної предметної області;
- порівняння існуючих аналогів програмного забезпечення;
- проектування та реалізація чат-боту для моніторингу стану навколишнього середовища;
- тестування отриманого чат-бота.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Месенджери

Месенджер – це програмне забезпечення, яке створене для смартфонів або персональних комп`ютерів, з метою миттєвого обміну між користувачами текстовими, аудіо або відео повідомленнями.

Створення та поширення програмного забезпечення для спілкування між користувачами почало свій рух з чатів, месенджерів та соціальних мереж. Проте, на теперішній час, все більшої популярності набувають месенджери через свій функціонал та зручність у використанні.

На графіку (рисунок 1.1.) представлено активну кількість користувачів в місяць у самих популярних месседжерах станом на січень 2022 року, за даними джерела Statista. [3]

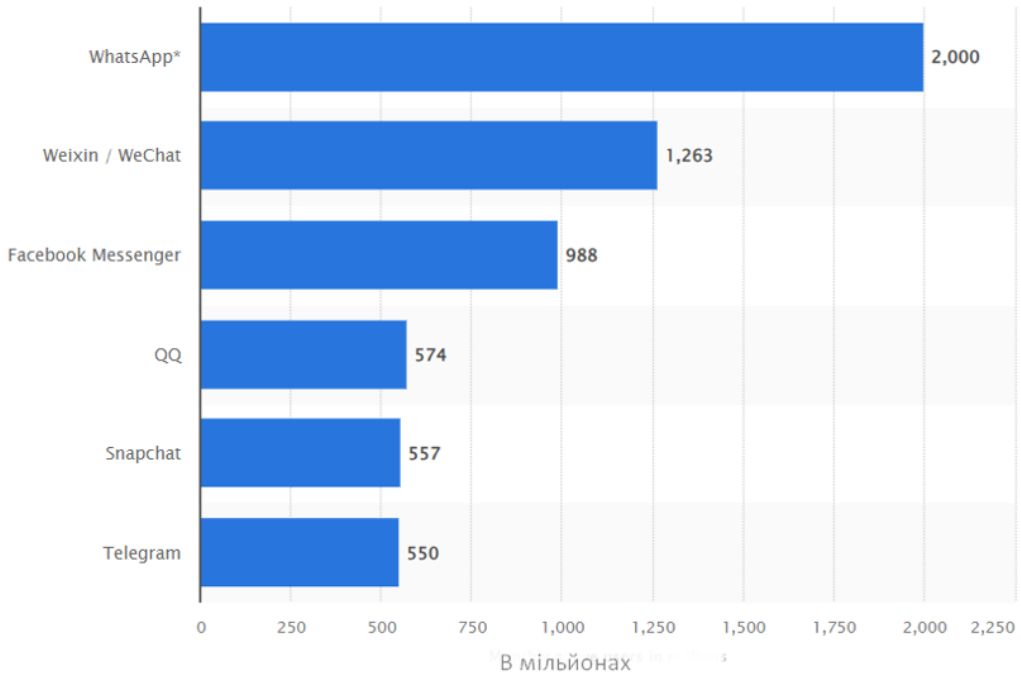


Рис. 1.1. – Активна кількість користувачів месенджерів в місяць.

Всі месенджери, на перший погляд, виконують однакові функції, проте всі вони мають певні особливості. Для аналізу було обрано самий популярний

месенджер в світі WhatsApp та один з найбільш зростаючих месенджерів Telegram.

WhatsApp - це безкоштовне американське програмне забезпечення, що доступне на міжнародному рівні, що виконує функції обміну миттєвими повідомленнями і служби голосової підтримки на основі IP, що належить Meta Platforms. Даний месенджер дозволяє користувачам надсилати текстові та голосові повідомлення, здійснювати голосові та відеодзвінки, а також ділитися фотографіями, документами, місцезнаходженням користувачів та іншим вмістом за допомогою інтернет мережі.[4]

Інтерфейс програмного забезпечення WhatsApp для мобільного пристрою можна побачити на рисунку 1.2.

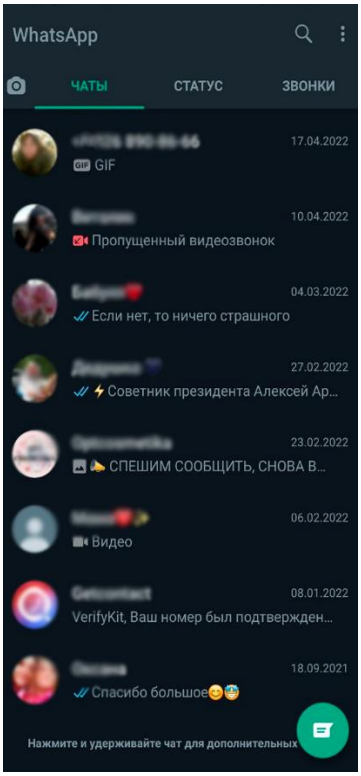


Рис. 1.2. – Інтерфейс месенджера WhatsApp для мобільного пристрою.

Клієнтський додаток WhatsApp працює на мобільних пристроях, але також можна отримати доступ із настільного комп’ютера, якщо пристрій користувача підключено до Інтернету під час використання настільної програми. [5]

З інтерфейсом месенджера WhatsApp для настільного комп`ютера можна ознайомитись на рисунку 1.3.

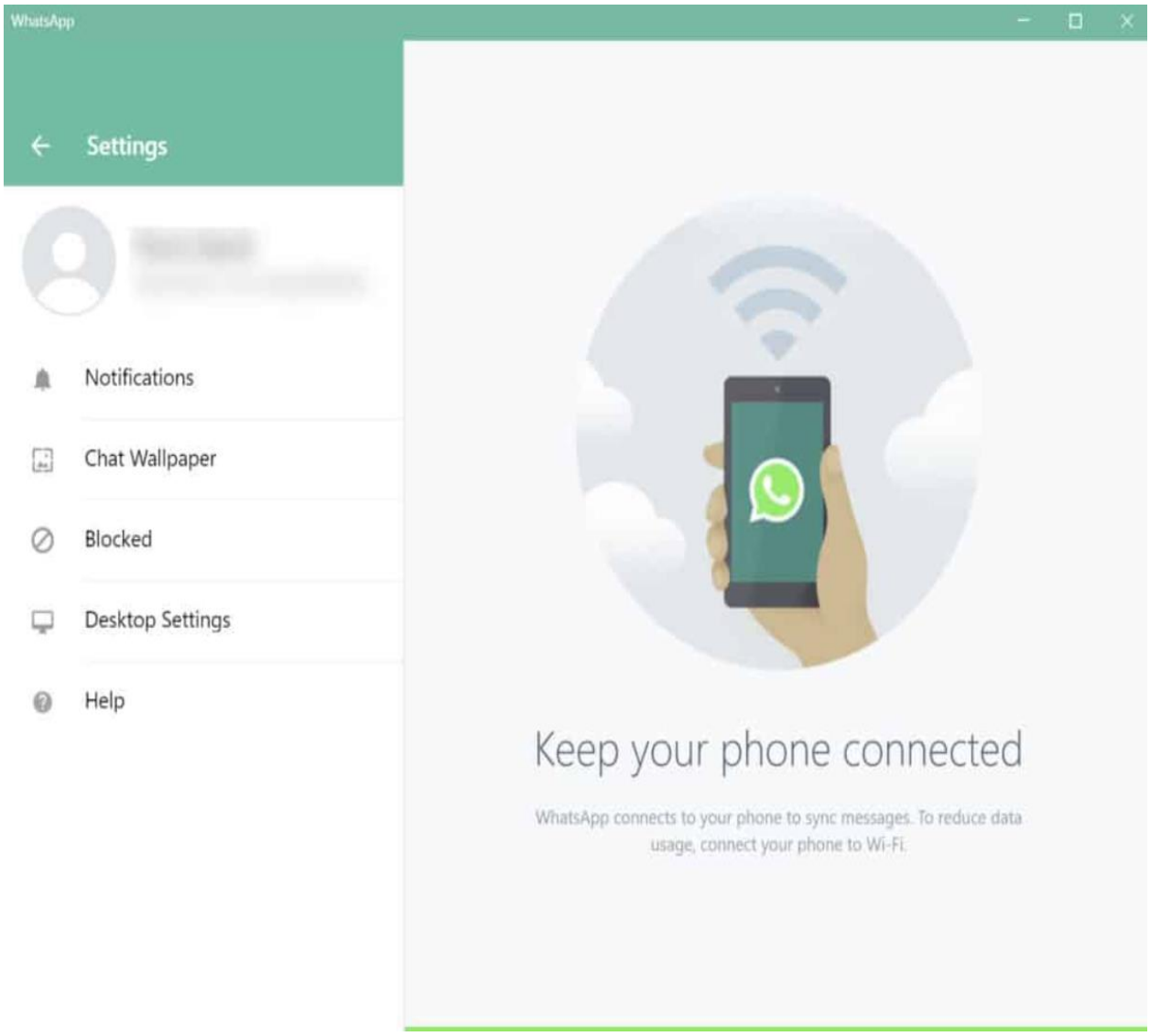


Рис 1.3. - Інтерфейс месенджера WhatsApp для настільного комп`ютера.

Даний додаток був створений WhatsApp Inc. Маунтін-В'ю в Каліфорнії, яка була придбана Facebook у лютому 2014 року за 19,3 мільярда долларів. [6] У 2015 році він зайняв перше місце по популярності в світі, маючи 900 мільйонів активних користувачів в місяць, [7] а у 2022 році кількість користувачів досягла відмітки 2 мільярди людей.

До переваг месенджера WhatsApp можна віднести:

- простота у використанні;

- можливість додавання ботів;
- голосові та відео дзвінки;
- відсутність рекламних оголошень;
- обмін файлами мультимедіа;
- наскрізне шифрування, що гарантує доступ до читання відправленого повідомлення лише за номером телефону;
- автоматичне додавання контактів до додатку з телефонної книги користувача;
- можливість змінити номер телефону власного облікового запису без втрачання даних.

До недоліків можна віднести:

- неможливість видалити відправлене повідомлення;
- дані не синхронізуються автоматично між пристроями;
- неможливість приховати свій номер телефону від користувачів, з якими ведеться спілкування.

Telegram – це програмне забезпечення, створене для обміну повідомленнями між користувачами з акцентом на швидкість, безпечність та простоту в використанні. Даний месенджер має змогу працювати на багатьох пристроях одночасно, так як повідомлення мають автоматичну синхронізацію між всіма підключеними приладами. За допомогою Telegram можна відправляти повідомлення, фотографії, фото або відео та файли будь-якого виду. Одною з переваг даного програмного забезпечення є можливість створювати групи вмістимістю до 200 000 людей. В доповнення до цього він підтримую скрізне шифрування голосових або ж відео дзвінків, що гарантує вашу безпеку спілкування. [2]

З інтерфейсом месенджера Telegram для мобільного пристрою можна ознайомитись на рисунку 1.4.

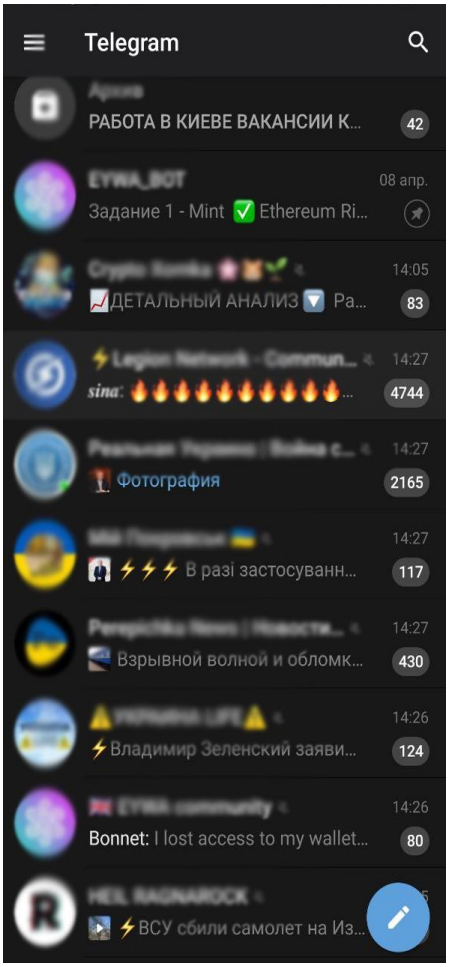


Рис. 1.4. – Інтерфейс месенджера Telegram для мобільного пристрою.

Авторами Telegram були два брати, Павло та Миколай Дурови, основними цілями яких було створити додаток для швидкого та безпечного спілкування. Їх проект був запущений 14 травня 2013 року і стрімко ввійшов у список одних з найперспективніших проектів, і у 2022 році досягає 550 мільйонів активних користувачів в місяць.[9]

Висока безпечність спілкування була реалізована розробниками за допомогою секретних чатів та унікальним протоколом шифрування MTProto (рисунок 1.5.).

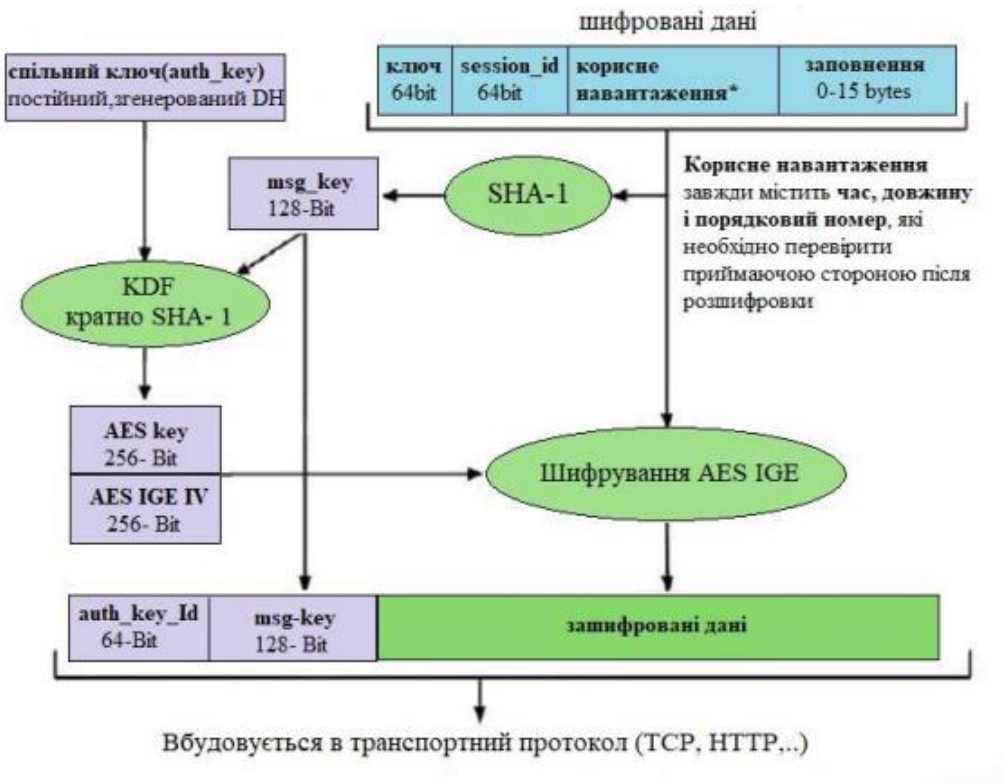


Рис. 1.5. – Протокол шифрування MTProto.

Функція секретного чата була реалізована для тієї категорії користувачів, у яких є потреба максимально безпечного обміну повідомленнями. Ці повідомлення проходять шифрування ключами від одного пристрою до іншого, оминаючи зберігання на сервері додатку. Безпечність гарантується тим, що всі повідомлення, такі як: текст, відео файли, аудіо файли, заборонено пересилати в інші чати, а також робити знімок екрану. Дане листування прикріплюється до пристрою, з якого було почате листування, і при спробі змінити пристрій, все видаляється.[2]

До переваг месенджера Telegram відносяться:

- підтримка багатьма платформами (Windows, MacOs, Android, IOS, WindowsPhone);
- швидкість роботи;
- відкритість API та коду, що дає змогу розробникам створювати доповнення до Telegram;
- підтримка ботів;

- можливість додати на один пристрій одночасно 3 облікових засоби;
- одночасна синхронізація між всіма пристроями;
- секретні чати, які автоматично видаляються з часом;
- можливість створення груп з вмістимістю до 200 000 людей;
- голосові та відео повідомлення;
- відправлення будь - яких файлів розміром до 2гб.

До недоліків відносяться:

- підв`язування облікового запису до номеру телефона;
- неповний функціонал на версіях для Windows та MacOs, таких як відправлення власної геолокації або створення секретних чатів.

1.2. Чат-боти

Чат-бот – це вид програмного забезпечення, що імітує спілкування з користувачем-людиною за допомогою текстових або аудіо-повідомлень. У наш час чат-боти почали широко застосовуватись у різних сферах нашого життя, починаючи від простої імітації спілкування, завершуючи аналізом майбутнього курсу валюти.[10]

Першими чат-ботами були ELIZA та PARRY, які були створені психіатром Кеннетом Колбай та Джозефом Вейценбаумом, і були спробою створити програми, які могли хоча б тимчасово примусити людину повірити, що вона спілкується з такою ж самою живою людиною. Ефективність чат-боту PARRY була оцінена в 1970х роках за допомогою тесту Тьюрінга, суть якого полягає в визначені з ким спілкується людина, з людиною чи з комп`ютером. За результатами проведеного тестування було правильно ідентифіковано людину та чат-бота лише на рівні, що відповідає лише випадковим припущенням.[11]

З того часу чат-ботами був пройдений великий шлях. Розробники створюють сучасних чат-ботів на основі технологій штучного інтелекту, що включає алгоритми глибокого навчання та технології машинного навчання. Такі типи ботів вимагають дуже великої кількості даних. Чим більше користувач буде

взаємодіяти з ботом, тим краще буде працювати його розпізнавання та прогнозування наступних відповідей.

Через постійне вдосконалення чат-ботів зростає його популярність на споживчих та ділових ринках, оскільки споживачі почали все більше і більше переходити від голосового до більш пасивного текстового спілкування, чат-боти почали займати нішу, яку раніше займали телефонні дзвінки.

Розглянемо найпоширеніші типи чат-ботів:

- Скриптові або чат-боти швидкої відповіді – це найпростіший вид чат-ботів, які діють як ієрархічне дерево рішень. Ці боти взаємодіють з користувачем за допомогою попередньо визначених запитань, які продовжуються, поки не буде знайдена відповідь на запитання користувача;
- Чат-боти на основі меню – вимагають від користувача робити вибір із попередньо визначеного списку або меню для надання боту більшого розуміння потреб клієнта;
- Чат-боти на основі розпізнавання ключових слів – це чат боти, які намагаються зрозуміти, що вводить користувач і відповідно реагувати, орієнтуючись на ключові фрази, що вводить клієнт. Цей тип ботів поєднує в собі штучний інтелект та ключові слова, що регулюються для реагування належним чином;
- Гібридні чат-боти – це боти, що поєднують в собі елементи ботів на основі меню та розпізнавання ключових слів. В даному типі ботів користувач може, при бажанні, обрати відповіді на свої запитання за допомогою ключових слів, або ж в меню, якщо розпізнавання не принесло очікуваного результату;
- Контекстні чат-боти – це боти, що вимагають зосередженої уваги до даних. Вони використовують методи машинного навчання та штучного інтелекту для запам`ятовування розмови та взаємодії користувачів, для подальшого використання цих даних для власного покращення з часом. На відміну від пошуку по ключовим словам, ці

боти використовують пряме запитання користувача, опираючись на формулювання відправленого повідомлення, і на основі цього надають відповідь та самовдосконалюються;

- Голосові чат-боти – це тип ботів, що за допомогою штучного інтелекту можуть отримувати та інтерпретувати голосові дані для аналізу та подальшої відповіді користувачу людською мовою. Користувачі можуть взаємодіяти з голосовими ботами за допомогою голосових команд або текстових повідомлень та отримувати подальші відповіді щодо поставленого питання.

Чат-боти вже протягом багатьох років використовуються в програмах для обміну миттєвих повідомлень та інтерактивних онлайн іграх, але віднедавна все частіше вони почали застосовуватись в сферах послуг та продажів.

Організації мають змогу використовувати чат-ботів для таких цілей:

- обслуговування клієнтів – використовується для легшого надання агентами відповідей на запити, що часто повторюються. Наприклад, клієнт може за допомогою номера замовлення отримати інформацію щодо його статусу відправлення. Зазвичай, якщо бот не може дати відповідь на запит клієнта, чат переходить до живої людини співробітника, де клієнт отримує відповідь на питання, що його цікавить;
- Інтернет-магазини – використовується для відповідей на нескладні запитання про продукт магазину або для надання корисної для споживачів інформації, яку вони можуть шукати, наприклад, ціну товару або вартість доставки;
- віртуальні помічники – використовуються для виконання задач, в залежності від отриманої інформації від користувача. До прикладів даного типу ботів відносяться: Google Assistant, Kortana, Siri та інші.

Світ цифрових технологій розвивається дуже стрімко, тим самим зростають і потреби та очікування користувачів. Багато споживачів очікують, що технічна підтримка користувачів буде працювати цілодобово і вважають, що цей аспект

має таку ж саму важливість, як і якість продукту або послуги. Крім того, користувачі більш проінформовані про різноманітність товарів та послу і з меншою вірогідністю зберігають лояльність до бренду.

Чат-боти стали відповіддю на ці нові потреби та зростаючі очікування, так як вони можуть замінити живий чат з людиною та інші форми контакту, такі як телефонні дзвінки або електронна пошта.

До переваг чат-ботів в сфері товарів та послуг належать:

- можливість вести кілька розмов одночасно. Чат-боти мають змогу спілкуватися в один час з тисячами користувачів. Це дуже підвищує продуктивність бізнесу та зменшує час очікування клієнтів;
- економічна ефективність. Чат-бот це ввідносно дешева та одноразова інвестиція, на відміну від створеного персонального крос платформного додатку чи найму додаткового персоналу. Крім того, чат-боти значно зменшують кількість фатальних проблем, що зазвичай спричинені людськими факторами. Витрати на залучення нових користувачів також зменшуються за допомогою можливості ботів відповідати протягом декількох секунд;
- економія часу. Чат-боти мають можливість автоматизації завдань, які необхідно виконувати часто і в конкретний період часу. Це дає можливість співробітникам зосередитись на більш важливих завданнях і не дає клієнтам довго очікувати на отримання відповідей на їх питання;
- активна взаємодія з користувачами. Раніше організації давали перевагу пасивній взаємодії з клієнтами і очікували, поки покупці не звернуться самі. Чат-боти допомагають організаціям змінювати цей стиль на активну взаємодію, оскільки вони можуть ініціювати розмову самостійно та стежити за тим, як клієнти використовують певні веб сайти та цільові сторінки. Згодом дана інформація, зібрана в результаті моніторингу, допомагає пропонувати конкретні стимули

для покупців, допомогти користувачам з орієнтуванням на сайті або відповісти на питання, що їх зацікавили;

- відстеження та аналіз даних. Чат-боти збирають дані від кожної взаємодії користувачів для подальшої допомоги компаніям покращити якість своїх послуг та продуктів або для оптимізації власного веб сайту. Боти також мають змогу записувати дані користувачів для відстеження поведінки та моделі покупок. Згодом ця інформація дає організаціям певне уявлення про те, як краще продавати свої продукти та послуги, а також дізнаватись про проблеми, з якими стикаються користувачі під час процесу покупки;
- полегшення виходу на міжнародні ринки. Чат-боти мають змогу вирішувати проблеми та запити клієнтів, використовуючи декілька мов. Їх цілодобовий режим роботи дозволяє клієнтам користуватись ними, незалежно від часового поясу чи місця знаходження.
- розширення клієнтської бази. Чат-боти мають змогу покращити підбір потенційних клієнтів за допомогою поставок запитань протягом усього шляху покупця та надавати інформацію, яка може переконати користувача та зробити його постійним клієнтом. Потім чат-боти можуть надавати інформацію про потенційних клієнтів команді з продажу, яка може взаємодіяти з ними. Даними діями боти можуть підвищити коефіцієнт конверсії та забезпечити те, щоб потенційний покупець зробив замовлення.
- аналіз кваліфікації. Чат-боти можуть допомогти відділам продажів визначити кваліфікацію потенційного клієнта, використовуючи визначені ключові показники ефективності, такі як: бюджет, терміни та ресурси. Це може запобігти тому, щоб компанії витрачали час на некваліфікованих потенційних клієнтів і клієнтів, що забирали багато часу.

До проблем використання чат ботів можна віднести:

- перешкоди новітніх технологій. Так як технологія чат-ботів відносно нова, вона зіштовхується з перепонами, з якими організації можуть не впоратися. Хоча боти з підтримкою штучного інтелекту можуть вчитись на кожній взаємодії, покращуючи свою роботу, цей процес коштує дуже багато, якщо перші взаємодії змусять клієнта відсторонитись від використання сервісу;
- складність тексту користувачів. Через те, що всі люди друкують свої речення по-різному, це може призвести до неочікуваних намірів. Чат-боти повинні вміти обробляти як короткі, так і довгі повідомлення, а також спливаючі чати з великим вмістом, в порівнянні з декількома короткими представленнями;
- безпека даних. Користувачі мають довіряти чат-боту, для того, щоб поділитись особистими даними. Для цього організаціям необхідно переконатися, що вони запитують інформацію, яка необхідна лише для надання повного виконування функціоналу. Тому необхідно дбати про безпеку чат-ботів від хакерів;
- вплив поведінки, настрою та емоцій користувачів. Люди бувають непередбачуваними, а емоції та настрій часто контролюють поведінку користувачів, тому користувачі можуть швидко змінити свою думку та модель поведінки. Після початкового запиту про пропозицію вони можуть замість цього дати команду. Чат-боти повинні мати змогу адаптуватися і розуміти цю випадковість і спонтанність;
- оцінка якості користувачів. Користувачі завжди хочуть отримати найкращі враження, але рідко бувають задоволені. Вони завжди хочуть, щоб чат-бот був кращим, ніж зараз. Це означає, що організації, які використовують чат-ботів, повинні постійно оновлювати та вдосконалювати їх, щоб користувачі розуміли, що вони працюють з надійним та якісним джерелом.

Більшість експертів очікують, що популярність чат-ботів з часом буде все більше зростати. У майбутньому штучний інтелект та машинне навчання будуть продовжувати розвиватись, пропонуючи все більш інноваційні можливості для чат-ботів, та впроваджуючи нові види текстових і голосових користувацьких можливостей, які будуть трансформувати клієнтський досвід. Ці вдосконалення також зроблять вплив на збір даних і будуть надавати все більш глибоке уявлення про клієнтів, що призведе до прогнозування поведінки майбутніх користувачів.[12]

1.3. Огляд та аналіз існуючих аналогів

1.3.1. **UkrForest_bot – бот лісової галузі**

UkrForest_bot – це бот, що був створений державним лісовим агентством України, для опрацювання та моніторингу заявок від користувачів на незаконну рубку лісів, виявлені пожежі, оприлюднення місць працювання браконьєрів, а також для отримання відповідей на питання, що цікавлять у цій галузі.[13] Інтерфейс чат-боту продемонстрований на рисунку 1.6.

До плюсів даного чат-боту відносяться:

- легкий в освоєнні;
- багато варіантів використання боту для запобігання знищення лісової галузі;
- можливість залишити заявку за допомогою геопозиції користувача;
- можливість додавати фото матеріали до заяви.

До недоліків можна віднести:

- неможливість перегляду поданих заяв інших користувачів;
- неповний функціонал у версії для комп`ютера;
- відсутність бонусів за подані заявки користувачів.

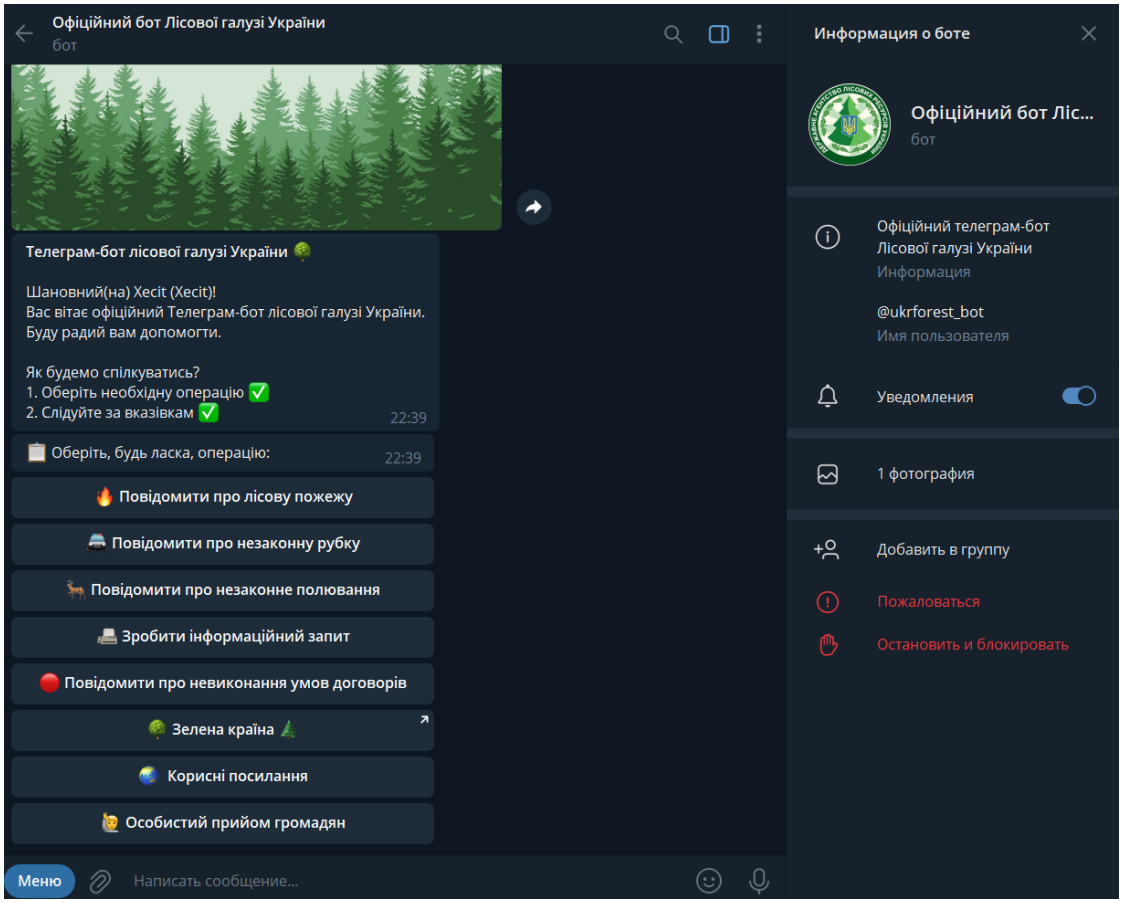


Рис. 1.6. – інтерфейс чат-боту UkrForest_bot

1.3.2. **SaveEcoBot – екологічний чат-бот**

SaveEcoBot – це чат-бот, що поєднує в собі дані про забруднення навколишнього середовища та інструменти його захисту [14] і надає користувачам такий функціонал як:

- перегляд оцінки впливу на навколишнє середовище в певних регіонах;
- перегляд даних про викид шкідливих речовин в атмосферу;
- перегляд даних про дозвіл користування природними ресурсами;
- перегляд даних про дозвіл користування водними об`єктами;
- перегляд даних про правила поведінки з небезпечними відходами.

З інтерфейсом чат-боту можна ознайомитись на рисунку 1.7.

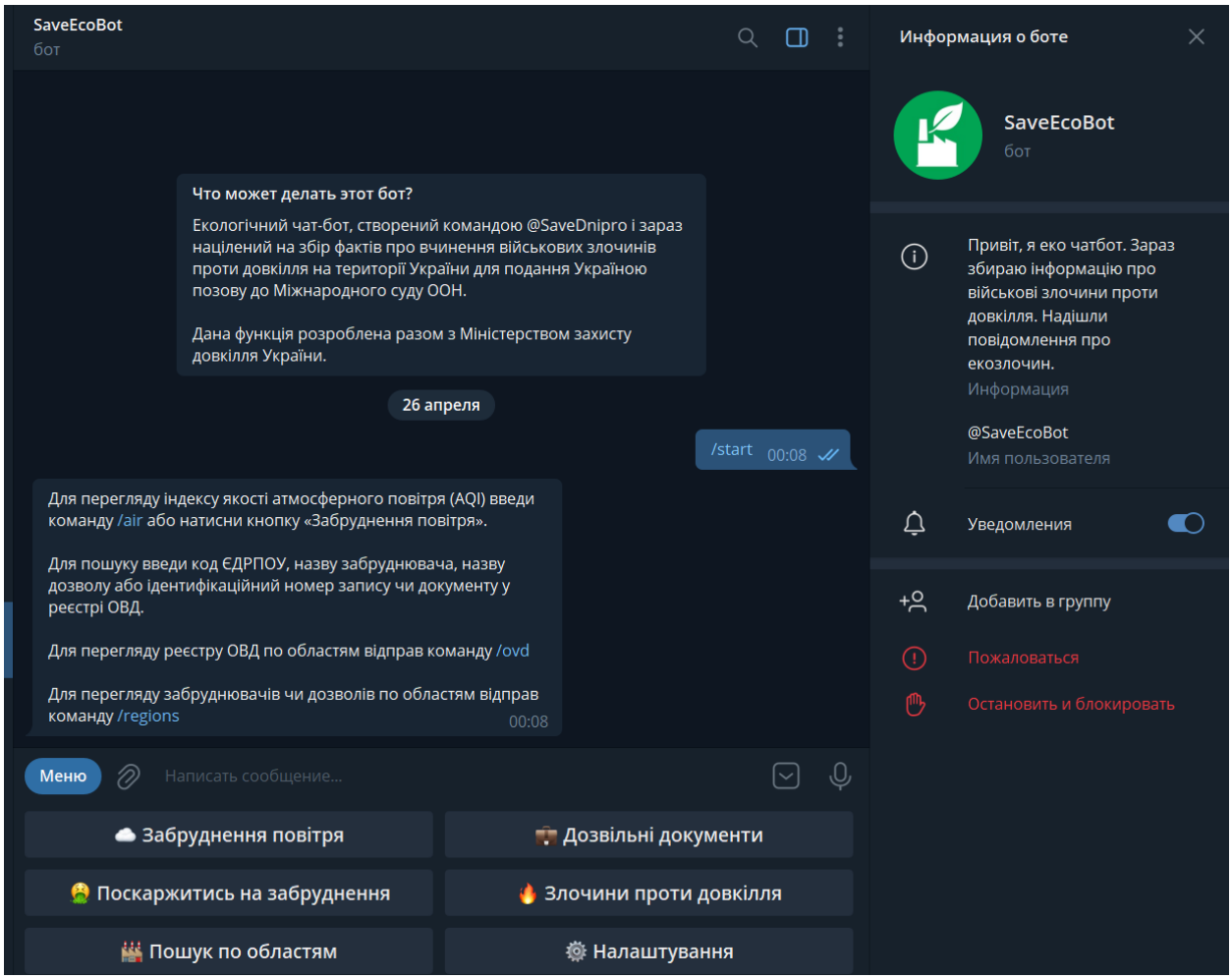


Рис 1.7. – Інтерфейс чат-боту SaveEcoBot

До переваг даного чат-боту відноситься:

- гнучке налаштування бота під потреби користувача;
- сортування даних за регіонами;
- моніторинг стану навколишнього середовища в реальному часі;
- можливість залишити власну заяву.

До недоліків відносяться:

- неповний функціонал у версії для комп`ютера;
- відсутність бонусів за подані заяви користувача;
- неможливість перегляду поданих заяв інших користувачів;
- неможливість залишити заяву за допомогою фото матеріалів та геопозиції пристрою користувача.

Після дослідження та аналізу існуючих аналогів чат-боту, була поставлена мета створити чат-бота для моніторингу стану навколишнього середовища на базі месенджера Telegram. Для виконання поставленої мети буде використовуватись мова програмування Python v3.10 та окремі модулі, що відповідають за обробку запитів Aiogram для роботи з Telegram API. Для приєднання до віддаленого серверу буде застосовуватись сервіс Google Cloud Platform.

2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Мова програмування Python.

Python — це інтерпретована інтерактивна об'єктно-орієнтована мова програмування. Вона включає в себе модулі, винятки, динамічне введення, динамічні типи даних дуже високого рівня та класи. Він підтримує декілька парадигм програмування за межами об'єктно-орієнтованого програмування, наприклад, процедурне та функціональне програмування. Python поєднує в собі чудову потужність з дуже чітким синтаксисом. Він має інтерфейси до багатьох системних викликів і бібліотек, а також до різних віконних систем і розширюється на C або C++. Його також можна використовувати як мову розширення для програм, яким потрібен програмований інтерфейс. Нарешті, Python є портативним: він працює на багатьох варіантах Unix, включаючи Linux і macOS, а також на Windows. [15]

Дана мова програмування була розроблена на початку 1990-х років Гвідо ван Россумом. Це динамічна мова високого рівня з легко читаним синтаксисом. Програми на Python інтерпретуються, що означає, що немає потреби в компіляції в двійкову форму перед виконанням програм. Це робить програми на Python трохи повільнішими, ніж програми, написані компільованою мовою, але при поточній швидкості комп’ютера та для більшості завдань це не є проблемою, а переносимість, яку Python отримує в результаті інтерпретації, є вигідним компромісом. Більш важливими та актуальними функціями Python для нашого використання є те, що його легко вивчати, легко читати, інтерпретувати та багатоплатформенність (програми Python працюють у більшості операційних систем); він пропонує безкоштовний доступ до вихідного коду; доступні внутрішні та зовнішні бібліотеки; і має підтримку Інтернет-спільноти. Python є відмінним вибором для вивчення мови. Простий синтаксис мови використовує обов’язковий відступ і виглядає схожим на псевдокод, який можна знайти в підручниках, які орієнтовані на студентів, які не програмують. Його простота – це

вибір дизайну, зроблений для того, щоб полегшити вивчення та використання мови. Ще однією перевагою, яка добре підходить для новачків, є додатковий інтерактивний режим, який дає миттєвий зворотній зв'язок з кожним твердженням, що, безумовно, заохочує експериментувати. Також слід звернути увагу на деякі недоліки Python. По-перше, час виконання повільніше, ніж для скомпільованих мов. По-друге, доступно менше числових і статистичних функцій, ніж у спеціалізованих інструментах, таких як R або MATLAB. (Однак модуль Numpy надає кілька числових і матричних функцій для Python.) І, по-третє, Python не так широко використовується, як JAVA, C або Perl.

Python використовується для розробки програмного забезпечення різних сфер застосування, а саме:

- штучний інтелект. У галузі штучного інтелекту проводяться численні дослідження, оскільки його можна застосувати до багатьох складних додатків у секторах подорожей, здоров'я, розваг, і це лише деякі з них. Багато великих компаній інвестували в дослідження ШІ по всьому світу і працюють над пошуком рішень, які змінять спосіб функціонування багатьох галузей. В Python є багато бібліотек та інструментів, які підтримують роботу AI, порівняно з іншими мовами. Це робить Python улюбленою мовою багатьох програмістів у цій галузі;
- машинне навчання. Машинне навчання є підмножиною штучного інтелекту. Вона стосується здатності систем навчатися та розвиватися на основі попереднього досвіду без будь-якого втручання людини. Машинне навчання має багато застосувань у таких функціях, як рекомендації відео на веб-сайтах потокової передачі, соціальні мережі, віртуальні персональні помічники тощо. Існують підмножини машинного навчання , як-от навчання під керівництвом, навчання без нагляду та навчання з підкріпленням. Простота використання мови програмування Python разом з багатьма корисними бібліотеками,

такими як Numpy, Skikit-learn, Keras, TensorFlow, Matplotlib, змусила її широко використовуватися в програмах машинного навчання;

- data science. За останні кілька років спостерігається експоненційне зростання кількості даних, які генеруються в усьому світі. Data science — це міждисциплінарна галузь, яка використовує наукові методи, процеси, алгоритми та системи для вилучення знань та ідей із багатьох структурних і неструктурованих даних. Вона пов’язана з аналізом даних і великими даними. Python є дуже універсальною мовою, і тому знаходить багато застосувань у різних областях data science. Розширена підтримка бібліотеки Python допомагає аналізувати і працювати з великими обсягами даних;
- мережі. Python знайшов свій шлях у сфері мереж, оскільки він дуже безпечний і забезпечує підтримку багатьох бібліотек. Конфігурація маршрутизатора та інші мережеві завдання можна автоматизувати за допомогою Python;
- веб-розробка. Python також використовується у веб-розробці багатьма компаніями. Він надає багато функцій безпеки, має велику підтримку бібліотек і має такі фреймворки, як Django і Flask, які полегшують роботу програмістів. [17]

Мова програмування Python найбільш підходить в якості першої мови для програмістів початкового рівня, тому що він має потужні інструменти, які відображають те, як люди розмірковують і як вони реалізують код. Крім того, він мінімізує додаткові ключові слова, необхідні для написання синтаксично правильної програми. Такий підхід є більш продуктивним, ніж навчання мов C++ або Java, в яких багато термінів і елементів, що пов'язані зі специфікою мови, а не з реалізацією алгоритму. Крім того, викладачі великої кількості університетів, таких як Массачусетський технологічний інститут, Каліфорнійський університет в Берклі, Каліфорнійський університет в Девісі, Університет штату Сонома, Вашингтонський університет, Університет Ватерлоо, Лютер-коледж і Свартмор-

коледж, використовували його для викладання вступного курсу програмування студентам факультету комп'ютерних наук. [16]

Сьогодні для людини, що працює з комп`ютерами, важливо вивчити хоча б одну мову програмування, тому що всі інновації та технології засновані на глибокому розумінні комп'ютерів, операційної системи, програмного API або якоїсь апаратної периферії. Всі вони створені програмістами, використовують певний спосіб мислення. І щоб набути такий спосіб мислення, потрібно звикнути хоча б до однієї з мов програмування і отримати кваліфікацію в області розробки програмного забезпечення.

Для будь-якої людини, яка починає вивчати програмування, важливо зосередитися на поняттях програмування, а не на специфіці мови, тому що вони можуть бути різними для різних мов програмування, але Python забезпечує найвищий рівень програмування. Таким чином, людині не потрібно думати про управління пам'яттю, яке неминуче в C ++, або про ієрархію класів, яка неминуча в Java, або про типи змінних та оголошеннях, які існують майже в кожній мові програмування.

2.2 Система управління базами даних PostgreSQL

PostgreSQL — це об'єктно-реляційна система керування базами даних (ORDBMS), була розроблена в Каліфорнійському університеті на факультеті комп'ютерних наук Берклі. Postgres започаткував багато концепцій, що стали доступними в певних комерційних системах баз даних набагато пізніше.

PostgreSQL є нащадком цього оригінального коду Берклі з відкритим кодом. Він підтримує велику частину стандарту SQL і запропоновує багато сучасних функцій, таких як:

- складні запити;
- зовнішні ключі;
- тригери;
- перегляди, що оновлюються;

- цілісність транзакцій;
- контроль багатоверсійного одночасного використання.

Крім того, PostgreSQL може бути розширений користувачем багатьма способами, наприклад, шляхом додавання нових функцій, операторів, типів даних, індексованих методів чи процедурних мов. А завдяки ліберальній ліцензії PostgreSQL може змінюватися, використовуватися та розповсюджуватися будь-ким безкоштовно для будь-яких цілей, хоч приватні, комерційні або академічні.

Система управління об’єктно-реляційною базою даних, тепер відома як PostgreSQL, походить від пакета POSTGRES, що був написаний в Каліфорнійському університеті в Берклі. Маючи понад два десятиліття розробки, PostgreSQL тепер є найдосконалішою базою даних з відкритим вихідним кодом, що доступна будь-де.

Проект POSTGRES, що очолювався професором Майклом Стоунбрейкером, був спонсорований Агентством перспективних дослідницьких проєктів оборони (DARPA), Дослідницьким офісом армії (ARO), Національним науковим фондом (NSF) і ESL, Inc. Розпочалася реалізація POSTGRES в 1986.

З того часу POSTGRES випустив декілька великих релізів. В 1987 році почала працювати перша «демо-програмна» система і була показана на конференції ACM-SIGMOD 1988 року. В червні 1989 року була випущена Версія 1 для кількох зовнішніх користувачів. Версія 3 з'явилася в 1991 році і додала підтримку декількох менеджерів сховища, покращений виконавець запитів і переписану систему правил. Наступні випуски до Postgres95 були зосереджені здебільшого на портативності та надійності.

POSTGRES використовувався для реалізації багатьох різних дослідницьких і виробничих застосувань. До них належать: система аналізу фінансових даних, база даних відстеження астероїдів, пакет моніторингу продуктивності реактивного двигуна, база даних медичної інформації та кілька географічних інформаційних систем. POSTGRES використовувався також як навчальний інструмент у кількох університетах. Нарешті, Illustra Information Technologies

підібрала код і комерціалізувала його. Наприкінці 1992 року POSTGRES став основним менеджером даних для проєкту наукових обчислень Sequoia 2000.

Протягом 1993 року розмір спільноти зовнішніх користувачів збільшився майже вдвічі. Стало все більш очевидним, що обслуговування коду прототипу та його підтримка забирали багато часу, який слід було б присвятити дослідженням баз даних. Намагаючись зменшити цей тягар підтримки, проект Berkeley POSTGRES офіційно завершився з версією 4.2.

Ендрю Ю і Джоллі Чен у 1994 році додали до POSTGRES інтерпретатор мови SQL. Під новою назвою Postgres95 пізніше був випущений в Інтернет, аби знайти свій власний шлях у світі як нащадок оригінального коду POSTGRES Berkeley з відкритим кодом. Код Postgres95 був зменшений на 25%. Багато внутрішніх змін покращили продуктивність і ремонтопридатність. Postgres95 випуск 1.0.x працював приблизно на 30–50% швидше, чим POSTGRES, версія 4.2.

До основних покращень, окрім виправлення помилок, відносились:

- мова запитів PostQUEL була замінена на SQL (реалізована на сервері). (Бібліотека інтерфейсу libpq було названо на честь PostQUEL.) Підзапити не підтримувалися до PostgreSQL, але їх можна імітувати в Postgres95 за допомогою визначених користувачем функцій SQL. Були реалізовані агрегатні функції. Також була додана підтримка речення запиту GROUP BY;
- нова програма (psql) була надана для інтерактивних запитів SQL, яка використовувала GNU Readline. Це значною мірою замінило стару моніторну програму;
- інтерфейс великих об'єктів був суттєво перероблений. Інверсія великих об'єктів була єдиним механізмом для зберігання великих предметів. Файлова система інверсії була видалена;
- вилучено систему правил на рівні екземпляра. Правила все ще були доступні як правила перепису;
- короткий підручник, який представляє звичайні функції SQL, а також функції Postgres95 був розповсюджений разом з вихідним кодом.

У 1996 році стало зрозуміло, що назва «Postgres95» втрачає свою актуальність, тому з часом і була обрана нова назва, PostgreSQL, щоб відобразити зв’язок між оригінальним POSTGRES і останніми версіями з підтримкою SQL. Одночасно була встановлена нумерація версій, починаючи з 6.0, повертаючи номери в послідовність, спочатку розпочату проєктом Berkeley POSTGRES.

Багато людей продовжують називати PostgreSQL «Postgres» за традицією або тому, що його так легше вимовляти. Це використання було швидко прийнято як псевдонім.

Під час розробки Postgres95 акцент був зроблений на виявленні та розумінні існуючих проблем у коді сервера. З PostgreSQL акцент перемістився на розширення функцій і можливостей, хоча робота в усіх сферах тривала . [18]

PostgreSQL – це потужна система об’єктно-реляційних баз даних з відкритим вихідним кодом, що використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають і масштабують найскладніші робочі навантаження даних.

PostgreSQL здобув міцну репутацію завдяки своїй перевіреній архітектурі, цілісності даних, надійності, надійному набору функцій, розширюваності та відданості спільноти відкритих вихідних кодів, які стоять за програмним забезпеченням, щоб постійно надавати продуктивні та інноваційні рішення. PostgreSQL працює на всіх основних операційних системах, має ACID-сумісність з 2001 року і має потужні доповнення, такі як популярний розширювач геопросторової бази даних PostGIS. В наш час PostgreSQL став реляційною базою даних з відкритим кодом, яку обирають багато людей та організацій.

PostgreSQL постачається з багатьма функціями, які допомагають розробникам створювати програми, адміністраторам захищати цілісність даних та допомагають керувати даними, незалежно від того, наскільки великий чи маленький набір даних. Окрім того, що PostgreSQL є безкоштовним і відкритим вихідним кодом, він дуже розширюваний. Наприклад, він дає змогу визначати власні типи даних, створювати власні функції та писати код з різних мов програмування без перекомпіляції бази даних.

PostgreSQL намагається відповідати стандарту SQL, якщо така відповідність не суперечить традиційним функціям або може призвести до неправильних архітектурних рішень. Багато функцій, необхідних стандартом SQL, підтримуються, хоча інколи вони мають дещо відмінний синтаксис або функції. Починаючи з версії 14 , яка вийшла у вересні 2021 року, PostgreSQL відповідає щонайменше 170 із 179 обов'язкових функцій для відповідності SQL:2016 Core.

Нижче наведено список різноманітних функцій, які можна знайти в PostgreSQL:

1) Типи даних:
    - примітиви: цілі, числові, рядкові, логічні;
    - структурований: дата/час, масив, діапазон/багатодіапазон, UUID;
    - документ: JSON/JSONB, XML, ключ-значення;
    - геометрія: точка, лінія, коло, многокутник;
    - налаштування: композитні, користувацькі типи.
2) Цілісність даних:
    - UNIQUE, NOT NULL;
    - первинні ключі;
    - зовнішні ключі;
    - обмеження виключення.
3) Безпека:
    - аутентифікація: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, сертифікат тощо;
    - надійна система контролю доступу;
    - безпека стовпців і рядків;
    - багатофакторна аутентифікація за допомогою сертифікатів і додаткових методів.
4) Інтернаціоналізація, текстовий пошук:
    - підтримка міжнародних наборів символів;
    - зіставлення без урахування регістру;

- повнотекстовий пошук

PostgreSQL дуже розширюваний, адже багато функцій, таких як індекси, мають визначені API, щоб була змога розробляти PostgreSQL для вирішення проблем. Також він має високу масштабованість як за величезною кількістю даних, якими він може керувати, так і за кількістю одночасних користувачів, які він може вмістити. У виробничих середовищах є активні кластери PostgreSQL, які керують багатьма терабайтами даних, і спеціалізовані системи, які керують петабайтами. [20]

2.3 Google Cloud Platform

Google Cloud — це набір загальнодоступних хмарних обчислювальних послуг від компанії Google. Платформа включає в себе цілий ряд розміщених служб для обчислень, зберігання та розробки додатків, які працюють на обладнанні Google. Розробники програмного забезпечення, адміністратори хмарних технологій та інші IT-спеціалісти підприємства можуть отримати доступ до служб Google Cloud через загальнодоступний Інтернет або через виділене мережеве з’єднання.[21]

Google Cloud пропонує послуги для обчислень, зберігання, роботи в мережі, великих даних , машинного навчання та Інтернету речей , а також інструменти керування хмарою, безпеки та розробників.

Розглянемо продукти хмарних обчислень у Google Cloud:

- Google Compute Engine — це послуга, яка надає користувачам екземпляри віртуальної машини для розміщення робочого навантаження;
- Google App Engine – це платформа, яка надає розробникам програмного забезпечення доступ до масштабованого хостингу Google;
- Google Cloud Storage – платформа хмарного сховища, призначена для зберігання великих неструктурованих наборів даних. Google також

пропонує варіанти зберігання бази даних, зокрема Cloud Datastore для нереляційного сховища NoSQL , Cloud SQL для повністю реляційного сховища MySQL і рідну базу даних Cloud Bigtable від Google;

- Google Kubernetes Engine — це система керування контейнерів Docker та контейнерних кластерів, які працюють у загальнодоступних хмарних сервісах Google. Google Kubernetes Engine заснований на Kubernetes , відкритій системі керування контейнерами Google;
- операційний пакет Google Cloud – це набір інтегрованих інструментів для моніторингу, реєстрації та звітування про керовані служби, які запускають програми та системи в Google Cloud;
- безсерверні обчислення, які надають інструменти та послуги для виконання робочого навантаження на основі подій, наприклад Cloud Functions для створення функцій, які обробляють хмарні події, Cloud Run для керування та запуску контейнерних додатків і Workflows для організації безсерверних продуктів і API;
- бази даних - це набір продуктів баз даних, які надаються як повністю керовані служби, включаючи Cloud Bigtable для великомасштабних робочих навантажень з низькою затримкою, Firestore для документів, CloudSpanner як високомасштабована, високонадійна реляційна база даних та CloudSQL як повністю керовану базу даних для MySQL, PostgreSQL і SQL Server.

Google Cloud пропонує послуги розробки та інтеграції додатків. Наприклад, Google Cloud Pub — це керована служба обміну повідомленнями в режимі реального часу, яка дозволяє обмінюватися повідомленнями між додатками що використовуються в сервісі. Крім того, Google Cloud Endpoints дає змогу розробникам створювати служби на основі RESTful API, а потім робити ці служби доступними для клієнтів iOS, Android і JavaScript. Інші пропозиції включають сервери Anycast DNS , прямі мережеві з’єднання, балансування навантаження , послуги моніторингу та реєстрації.

Google Cloud надає низку інструментів, призначених для допомоги при перенесенні даних і робочого навантаження. Приклади включають міграцію додатків у хмару, службу передачі даних BigQuery для планування та переміщення даних у BigQuery, службу міграції бази даних, щоб забезпечити легку міграцію до Cloud SQL, Migrate for Anthos, щоб допомогти перенести віртуальні машини в контейнери на GKE, Migrate for Compute Engine, щоб перенести віртуальні машини і фізичні сервери до Compute Engine, а також Storage Transfer Service для обробки передачі даних у Cloud Storage.

Набір послуг Google Cloud постійно розвивається, і Google періодично вводить, змінює або припиняє послуги на основі попиту користувачів або тиску конкуренції.

Основними конкурентами Google на ринку публічних хмарних обчислень є AWS і Microsoft Azure:

- AWS є найстарішою та найзрілішою загальнодоступною хмарою, яка з'явилася як публічна служба в 2006 році. Зазвичай вона пропонує найширший спектр загальних інструментів і послуг, і володіє найбільшою часткою ринку, звертаючись до широкої клієнтської бази, починаючи від окремих розробників і закінчуючи великими. підприємств державним установам;
- Microsoft Azure з'явився в 2010 році і виявився особливо привабливим для середовищ на базі Microsoft. Це полегшило перенесення робочих навантажень із центрів обробки даних до Azure і навіть створення гібридних середовищ. Azure є другою за величиною загальнодоступною хмарою, яка часто обслуговує великих корпоративних користувачів.

Але відмінності між даними сервісами все більш зменшуються, оскільки всі три загальнодоступні хмари розвиваються, щоб запропонувати подібні набори послуг і можливостей. Наприклад, конектор конфігурації Google Cloud, який використовується для модернізації додатків, поєднується з контролерами AWS та Microsoft Azure. Існує лише кілька сервісів Google Cloud, які не відповідають аналогові AWS або Microsoft Azure. Як приклад, служба бінарної авторизації

Google Cloud для безпеки контейнерів і інструмент звітів про помилки для розробників програмного забезпечення наразі не мають відповідних служб від AWS або Azure.

Користувачі хмарних систем повинні ретельно досліджувати та експериментувати з набором послуг, що надаються кожним постачальником хмар, перш ніж переходити на певну платформу, хоча багатохмарні середовища все частіше стають невід`ємною частиною серед корпоративних користувачів.[21]

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ЧАТ-БОТА ДЛЯ МОНІТОРИНГУ СТАНУ НАВКОЛИШНЬОГО СЕРЕДОВИЩА

3.1 Моделювання алгоритів використання чат-бота

Блок-схема — це графічне зображення системи, яка забезпечує функціональне уявлення про систему. Блок-схеми надають краще розуміння функцій системи та допомагають створювати взаємозв’язки всередині неї. Блок-схеми отримали свою назву від прямокутних елементів, що містяться в цьому типі діаграм. Вони використовуються для опису апаратних і програмних систем, а також для представлення процесів. Блок-схеми описуються і визначаються відповідно до їх функцій і структури, а також їх зв'язків з іншими блоками. [22]

Блок-схема — це фундаментальний спосіб, який розробники апаратного та програмного забезпечення використовують для опису цих систем, ілюструючи їхні робочі процеси та процеси. З іншого боку, електрики потребують, щоб вони представляли системи та їх зміни, наприклад, мехатронні системи в галузі вантажних перевезень.

Найчастіше блок-схеми дуже допомагають, коли необхідно чітке уявлення про потоки інформації або управління, а також коли проект містить багато процесів. Вони полегшують відображення складних алгоритмів, потоків даних або зв’язку між точними компонентами, наприклад, у масовому виробництві. Графічно представлені процеси проекту більш легкі для розуміння, ніж коли вони представлені у текстовій формі. [23]

З першою блок схемою можна ознайомитись на рисунку 3.1. в якій представлено один з основних функціоналів розроблюваної системи, а саме додавання до системи нових заявок щодо забруднення навколишнього середовища.

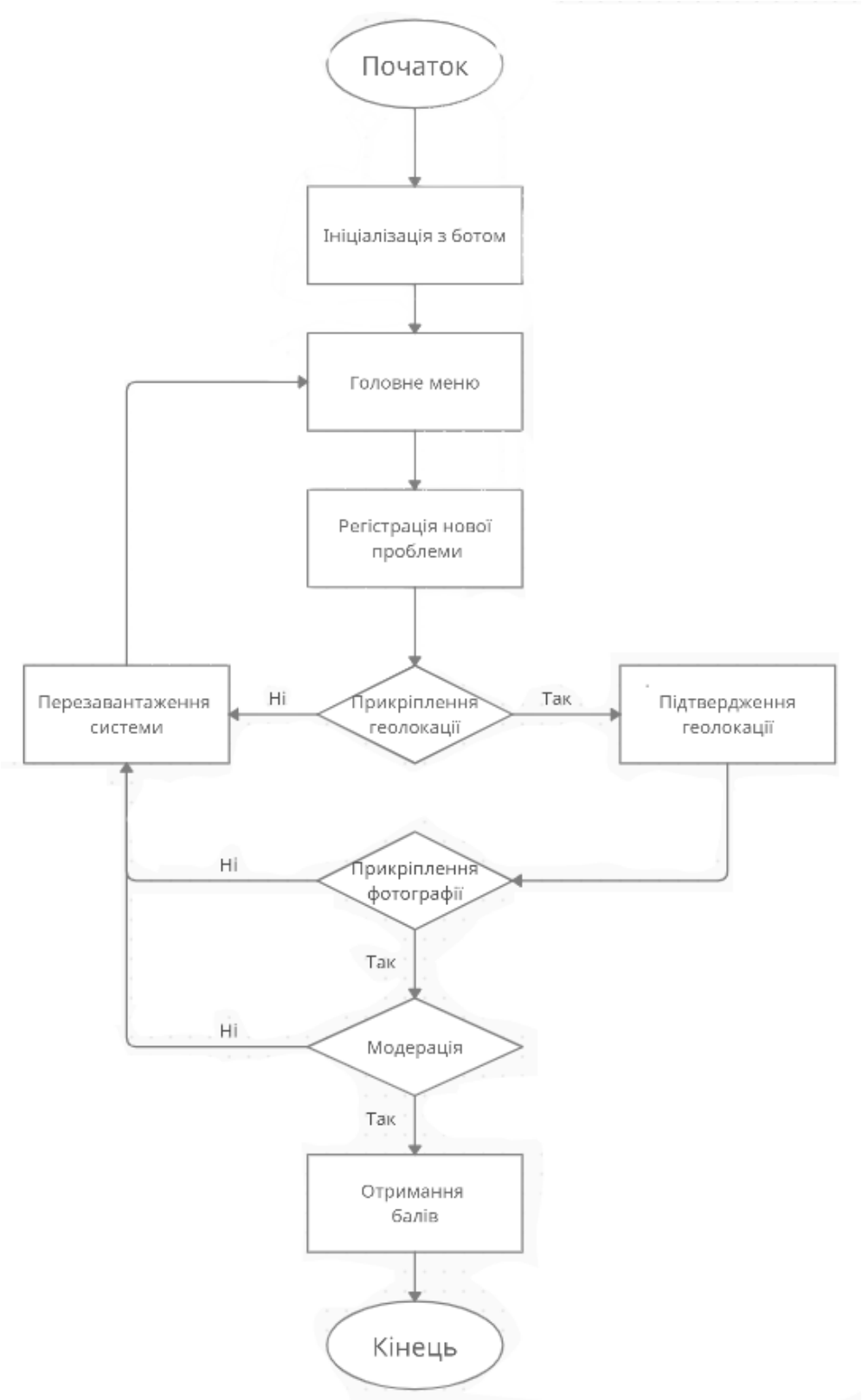


Рис. 3.1. – Алгоритм додавання нових заявок користувачів.

На блок-схемі (рисунок 3.2.) представлено алгоритм перегляду схвалених завантажених заявок іншими користувачами, та надана можливість їх опрацювання.

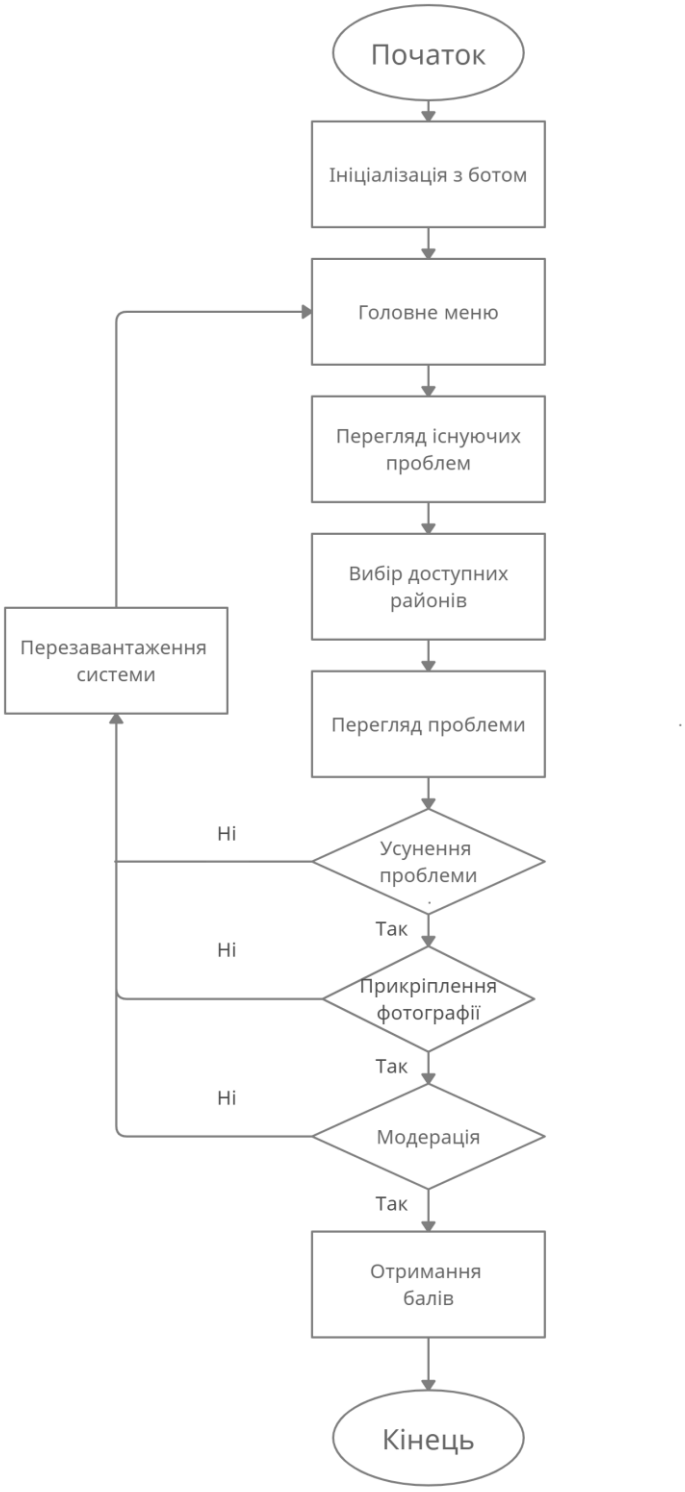


Рис. 3.2. – Алгоритм перегляду та опрацювання доступних заявок.

На блок-схемі (рисунок 3.3.) продемонстровано алгоритм модерації, а саме опрацювання та прийом поданих заявок користувачами.

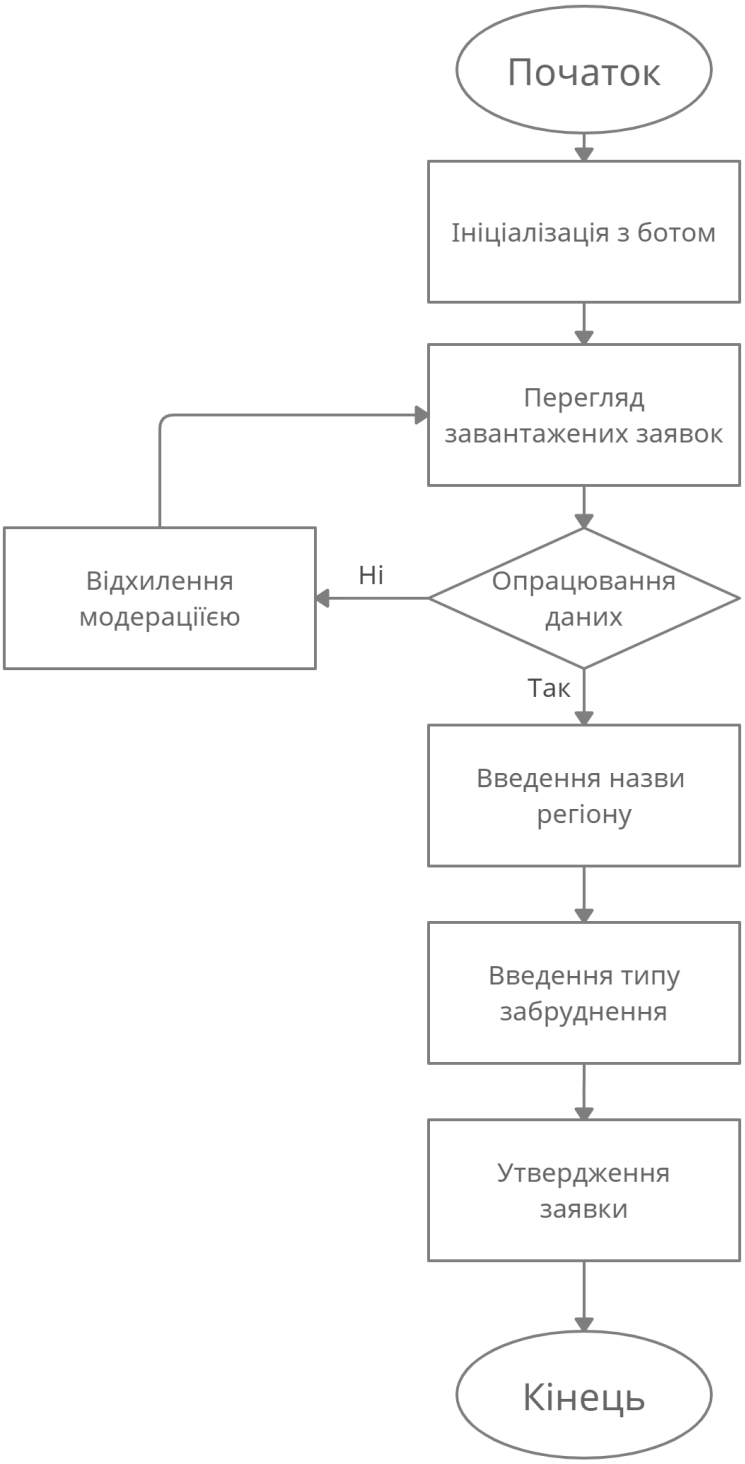


Рис. 3.3. – Алгоритм обробки та прийому заявок від користувачів.

На останній блок-схемі (рисунок 3.4.) продемонстровано алгоритм подання та обробки замовлень користувачів в бонусному магазині.

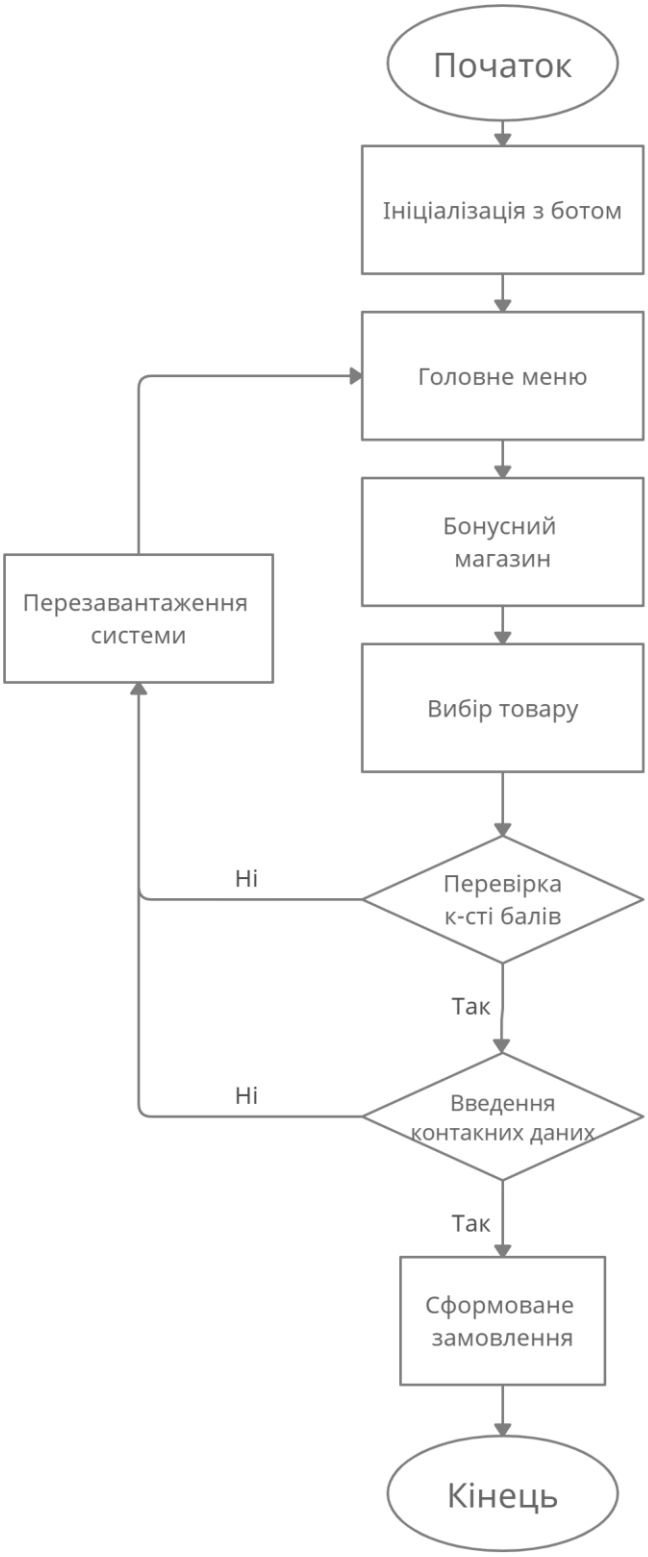


Рис. 3.4. – Алгоритм роботи бонусного магазину.

3.2 Реєстрація чат-бота в BotFather

Для створення нового бота необхідно отримати токен за допомогою бота BotFather.

Для знаходження цього бота, в пошуковій стрічці месенджера Telegram потрібно ввести запит @botfather. Цей етап продемонстровано на рисунку 3.5.

Рис. 3.5. – Пошук бота BotFather.

У вікні бота, нам пропонується почати його роботу за допомогою командт «/start», після чого нам відправляється список усіх доступних команд. Даний етап продемонстровано на рисунку 3.6.

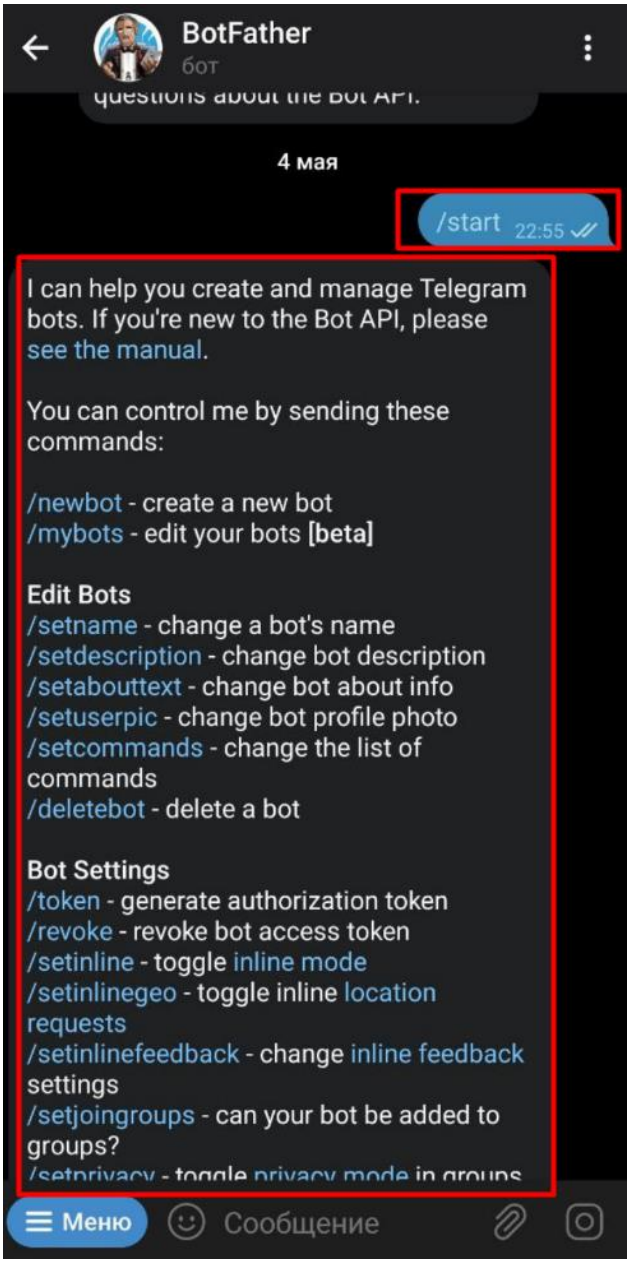


Рис. 3.6. – Початок роботи бота BotFarther.

В таблиці 3.1. представлено доступний функціонал для налаштування власного бота в боті BotFather.

Таблиця 3.1. – Основні команди та функціонал бота BotFather

Команда	Функціонал

/newbot	Створення нового боту
/mybots	Перегляд створених ботів
/setname	Редагування назви бота
/setdescription	Редагування вступного повідомлення повідомлення бота
/setabouttext	Редагування опису бота
/setuserpic	Налаштування зображення бота
/setcommands	Створення списку команд
/deletebot	Видалення бота
/token	Перегляд токена створеного бота
/revoke	Відхилення прав доступу до бота
/setinline	Надання функції виклику бота з інших чатів
/setinlinegeo	Надання функції перегляду розташування бота
/setjoingroups	Надання можливостії додавання бота до інших чатів
/setprivacy	Налаштування конфіденційності бота

На даному етапі розробки чат-бота, нас цікавить команда «/newbot», яка запускає процес створення нового бота в системі. Після введення команди нам висвічується повідомлення, для присвоєння назви нашого майбутнього бота (рисунок 3.7.).

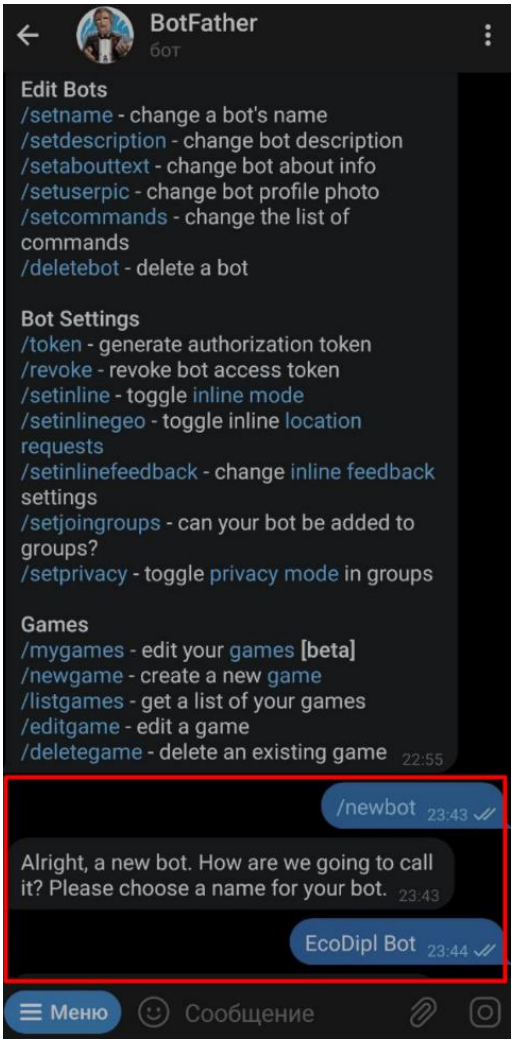


Рис. 3.7. – Присвоєння назви чат-боту.

Після присвоєння ім`я чат-боту, нам пропонується створити ім`я користувача для даного боту. Обов`язковою умовою цього етапу являється те, що нікнейм повинен містити в назві приставку bot. Ця умова створена для відрізнення користувачами справжніх людей від ботів. Даний етап продемонстровано на рисунку 3.8.

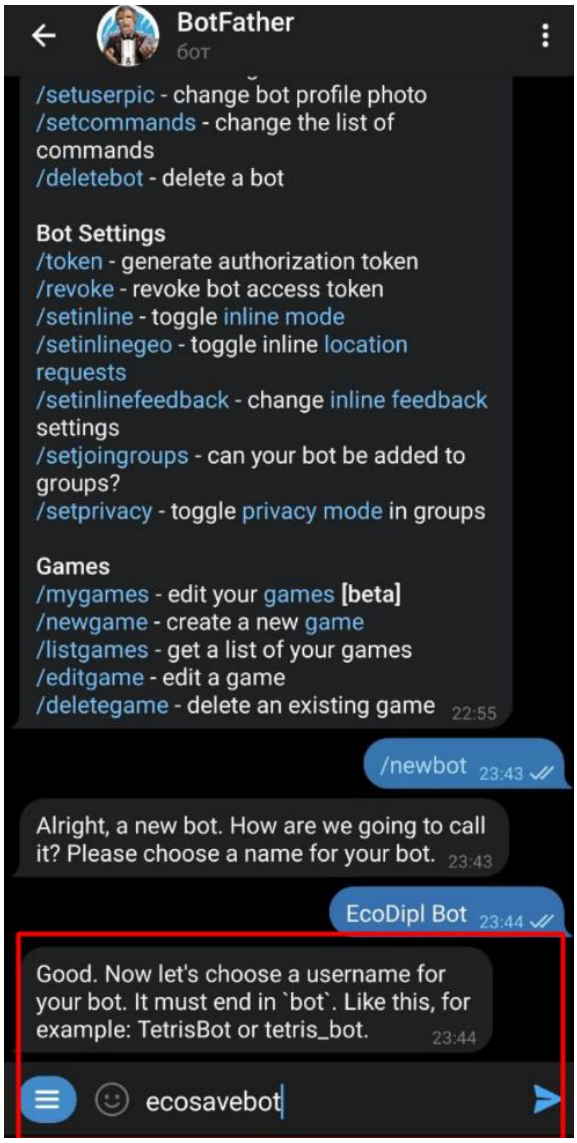


Рис. 3.8. – Присвоєння ім`я користувача боту.

Після закінчення цього етапи, бот відправляє нам фінальний результат, а саме секретний токен (рисунок 3.9.).

Токен — це рядок коду, який необхідний для авторизації бота та надсилання запитів до API бота . Токен необхідно тримати у безпеці та нікому про нього не повідомляти, адже його може використовувати будь-хто для керування вашим ботом. [24]

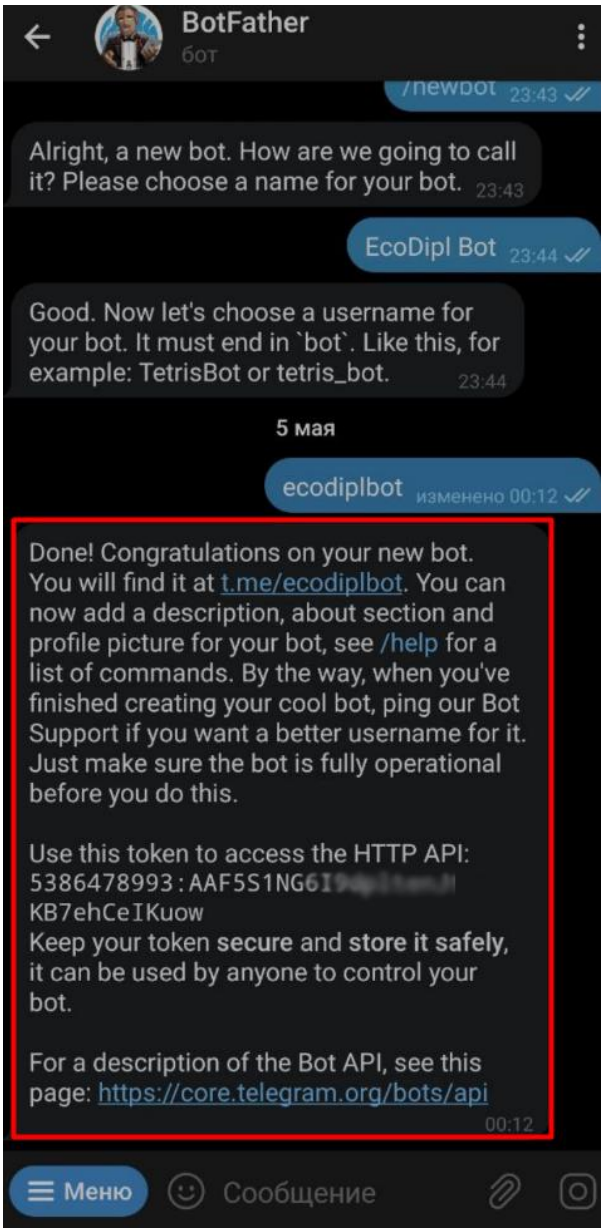


Рис. 3.9. – Отримання токену чат-бота.

3.3 Розробка основних механізмів роботи чат-бота.

Початком роботи чат-бота є файл start.py в якому відбувається процес ініціалізації користувача з чат-ботом, перевірка користувача на присутність в базі даних, та відображення привітального повідомлення. З даним процесом можна ознайомитись на рисунку 3.10.

```python
@dp.message_handler(commands=['start'], state="*")
async def on_process_start(message: types.Message, state: FSMContext):
    """ Привітальне повідомлення та перевірка наявності користувача в базі даних """

    await state.finish()
    user = await user_api.create(message.from_user.id)

    username = message.from_user.username
    if not username:
        username = message.from_user.first_name

    delete_keyboard_message = await message.answer(
        text=". . . ",
        reply_markup=types.ReplyKeyboardRemove(),
    )
    await delete_keyboard_message.delete()

    await message.answer_photo(
        photo="AgACAgIAAxkBAAKRgmJTazu02AWA386TGUlGmnR0QM8dAAJ4uDEbPpZJSrcB1qSIbWxvAQADAgADeQADIwQ",
        reply_markup=await inline.start_menu(),
        caption=f"<b>{username}</b> Вітаю вас в EcoDipl Bot.\n\n" +\
            "В даному боті ви можете допомогти врятувати "
            "наше навколишнє середовище від забруднення.\n\n"
            "У вас є можливість відправляти нам місця"
            "забруднення, або ж їх усувати.\n\n"
            "За кожну виконану дію ви будете отримувати бали, "
            "які ви зможете обміняти на товари в нашому магазині"
    )
```

Рис 3.10. – Реалізація ініціалізації користувача з чат-ботом.

На наступному етапі буде розглянуто реалізацію функціоналу завантаження забрудненого місця користувачем, а саме: прикріплення геолокації, прикріплення фотографії та відправлення заявки та модерацію. (Рис. 3.11. – 3.13.)

```
    await UploadPlace.get_place_geo.set()
    await call.message.delete()
    await call.message.answer(
        text="Відправте свою геолокацію біля місця забруднення",
        reply_markup=await reply.get_location(),
    )

    await call.answer()


@dp.message_handler(
    state=UploadPlace.get_place_geo,
    content_types=types.ContentType.LOCATION,
)
async def on_process_upload_place_step_2(
        message: types.Message,
        state: FSMContext
    ):
```

Рис. 3.11. – Реалізація прикріплення користувачем геопозиції.

```
    await UploadPlace.get_place_photo.set()
    async with state.proxy() as data:
        data['latitude'] = message.location.latitude
        data['longitude'] = message.location.longitude

    await message.answer(
        text="Відправте фото забруднення",
        reply_markup=types.ReplyKeyboardRemove(),
    )


@dp.message_handler(
    state=UploadPlace.get_place_photo,
    content_types=types.ContentType.ANY,
)
async def on_process_upload_place_step_3(
        message: types.Message,
        state: FSMContext,
    ):
```

Рис. 3.12. – Реалізація прикріплення користувачем фотографії.

```
async with state.proxy() as data:
    photo_id = message.photo[-1].file_id
    latitude = float(data['latitude'])
    longitude = float(data['longitude'])

record = await place_api.create(
    latitude=latitude,
    longitude=longitude,
    photo_id=photo_id,
)

location_url = LOCATION_TEMPLATE.substitute(
    latitude=latitude,
    longitude=longitude
)

await bot.send_photo(
    chat_id=MODER_TELEGRAM_GROUP_ID,
    photo=photo_id,
    caption=f"Запит на додавання забрудненого місця\n\n" +\
        f"<a href='{location_url}'>" +\
        f"Подивитись на мапі</a>",
    reply_markup=await inline.moder_menu(
        db_id=record.db_id,
        user_tg_id=message.from_user.id
    ),
)
```

Рис. 3.13. – Реалізація відправлення заявки на модерацію.

Всі заявки, що пройшли модерацію сортуються адміністратором по регіонам, а також за типами забруднення для зручності користувачів. Після вибору регіону та типу забруднення (Рис. 3.14.), користувачу відображається заявка з подальшою можливістю її опрацювати та вирішити. (Рис. 3.15.)

```
    async with state.proxy() as data:
        data['region_id'] = callback_data['region_id']

    pollution_types = await pollution_type_api.get_all_with_region(
        region_id=int(callback_data['region_id']),
    )

    await call.message.edit_text(
        text="Виберіть тип забруднення",
        reply_markup=await inline.pollution_menu(pollution_types),
    )


@dp.callback_query_handler(ctypes.pollution_menu.filter())
async def on_process_show_places(
        call: types.CallbackQuery,
        state: FSMContext,
        callback_data: dict,
    ):
    """ Перегляд місць в обраному регіоні """

    async with state.proxy() as data:
        region_id = int(data['region_id'])
        pollution_type_id = int(callback_data['type_id'])

    places = await place_api.get_all_with_region_and_type(
        region_id=region_id,
        pollution_type_id=pollution_type_id,
    )
```

Рис. 3.14. – Реалізація вибору регіону та типу забруднення.

```
await call.message.delete()
for place in places:
    location_url = LOCATION_TEMPLATE.substitute(
        latitude=place.latitude,
        longitude=place.longitude,
    )

    await call.message.answer_photo(
        photo=place.photo_id,
        caption=f"<a href='{location_url}'>Подивитись на мапі</a>\n\n" +\
            "Після завершення прибирання відправте фото підтвердження",
        reply_markup=await inline.place_menu(place.db_id),
    )
```

Рис. 3.15. – Реалізація опрацювання та вирішення заявки.

Етап модерації був реалізований таким чином, що чат-бот відправляє всі заявки в окрему групу модераторів, з можливістю подальшого її опрацювання. У модераторів є можливість відхилити заявку, або її ухвалити з подальшим вводом інформації щодо введення регіону та типу забруднення отриманої з поданої інформації (Рис. 3.16.).

```
    await ModerationPlace.get_place_name.set()
    async with state.proxy() as data:
        data['place_id'] = callback_data['place_id']
        data['user_tg_id'] = callback_data['user_tg_id']

    await call.message.delete()
    await call.message.answer(
        text="Надішліть назву регіону в якому знаходиться забруднене місце",
    )
    await call.answer()


@dp.message_handler(state=ModerationPlace.get_place_name)
async def on_process_get_place_name(
        message: types.Message,
        state: FSMContext,
    ):
    """ Получение названия региона """

    async with state.proxy() as data:
        data['place_name'] = message.text

    await ModerationPlace.get_type_pollution.set()
    await message.answer(
        text="Надішліть мені тип забруднення",
    )
```

Рис. 3.16. – Реалізація модерації заявок.

Після успішного проходження модерації, заявка заноситься в базу даних і відображається в меню користувачів , а людині яка подавала заявку відправляється повідомлення і нараховуються бонусні бали (Рис. 3.17.).

```
    async with state.proxy() as data:
        place_name = data['place_name']
        place_id = int(data['place_id'])
        user_tg_id = int(data['user_tg_id'])
        pollution_name = message.text

    type_pollution = await pollution_type_api.create(pollution_name)
    place_name = await place_name_api.create(place_name)
    await place_api.update_place_name(place_id, place_name.db_id)
    await place_api.update_type_pollution(place_id, type_pollution.db_id)

    user = await user_api.add_to_balance(
        tg_id=user_tg_id,
        amount=POINTS_PER_ACTION,
    )

    await bot.send_message(
        chat_id=user_tg_id,
        text="Запит на додавання забрудненого місця схвалено. " +\
            f"На ваш баланс нараховано {POINTS_PER_ACTION} балів. \n\n" +\
            f"Тепер на вашому балансі: {user.balance} балів.",
    )
    await state.finish()
    await message.answer(
        text="Нове місце успішно збережено",
    )
```

Рис. 3.17. – Реалізація внесення заявок в базу даних та нарахування бонусних балів користувачу.

За отримані бонусні бали користувачі мають змогу витратити їх у бонусному магазині купивши один із представлених товарів. Користувачу при достатній кількості балів потрібно заповнити певну форму, а саме відправити свої контактні дані (Рис. 3.18.), які згодом потрапляють у групу модерації для подальшого опрацювання.

```
    await Checkout.get_initials.set()
    async with state.proxy() as data:
        data['item'] = callback_data['item']

    user = await user_api.get(tg_id=call.message.chat.id)

    await call.message.delete()
    await call.message.answer(
        text="Введіть своє Прізвище Ім'я По батькові одним повідомленням, " +\
             "для того, щоб оформити замовлення",
    )

    await call.answer()


@dp.message_handler(state=Checkout.get_initials)
async def on_process_get_initials(message: types.Message, state: FSMContext):
    """ Отримуємо фіо користувача """

    await Checkout.get_contact.set()
    async with state.proxy() as data:
        data['initials'] = message.text

    await message.answer(
        text="Надішліть свій номер телефону, " +\
            "щоб закінчити оформлення замовлення",
        reply_markup=await reply.send_contact()
    )
```

Рис. 3.18. – Реалізація бонусного магазину чат-бота.

3.4 Тестування розробленого чат-бота.

Для повноцінного використання розробленого програмного забезпечення, його необхідно перш за все протестувати і створити текстову документацію, таку як тест кейси.

Тест кейси - це особливий набір дій або інструкцій для виконання тестувальником, який перевіряє певний аспект функціональності продукту або програми. Якщо тест провалиться, результатом може стати дефект програмного забезпечення, який організація може усунути. [25]

Таблиця 3.2 – Тест кейс реєстрації нових заявок

Заголовок	Крок	Очікуваний результат
Реєстрація нових заявок (Рис. 3.19)	1)Ввести команду «/start»	1)З`являється головне меню
	2)Натиснути кнопку «завантажити місце забруднення»	2)Відкриється меню завантаження нової заяви
	3)Натиснути кнопку «Відправити геолокацію»	3) В чат відправлється поточне місцезнаходження користувача
	4)Відправити боту фото місця забруднення	4)Бот відповідає на запит і відправляє заявку на модерацію

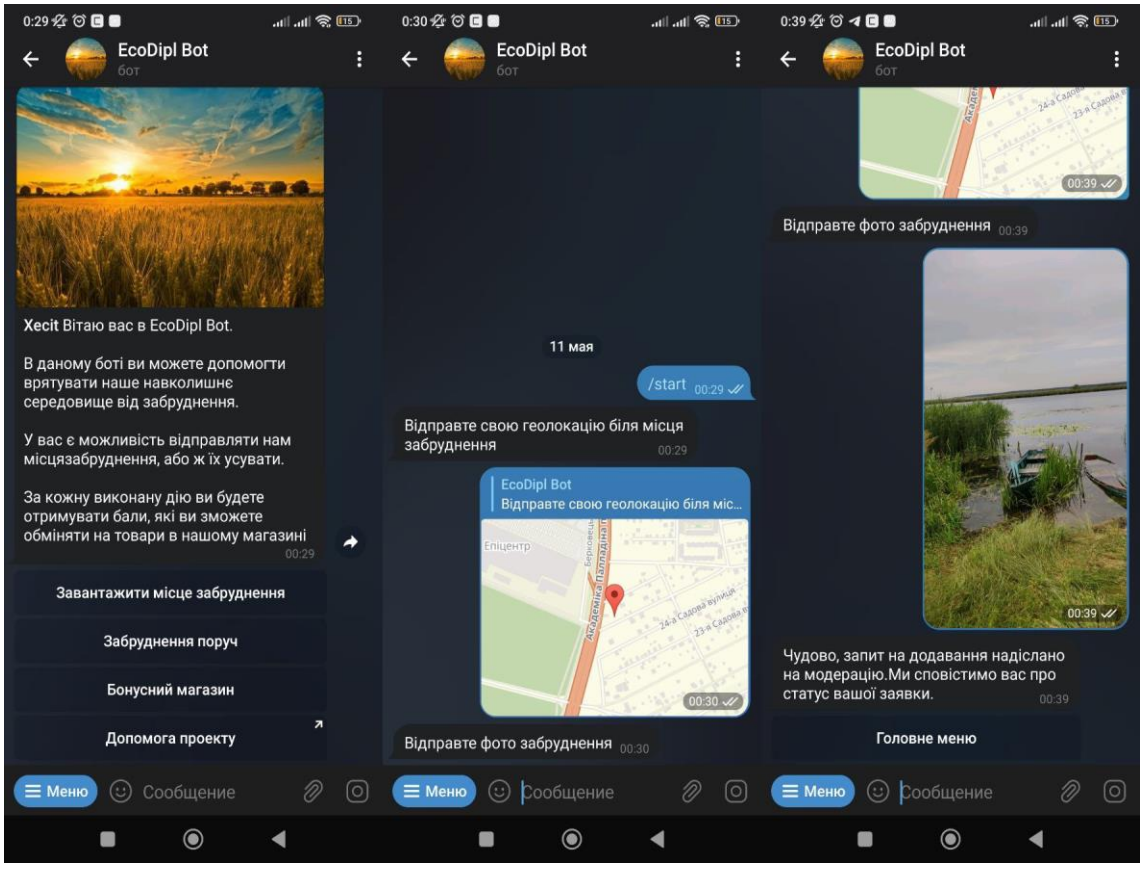


Рис. 3.19. – Реєстрація нових заявок.

Таблиця 3.3. – Тест кейс перегляду та вирішення завантажених заявок.

Заголовок	Крок	Очікуваний результат
Перегляд та вирішення завантажених заявок (Рис. 3.20)	1)Ввести команду «/start»	1)З`являється головне меню
	2)Натиснути кнопку «Забруднення поруч»	2)Відкриється меню вибору регіону
	3)Натиснути кнопку з цікавлячим регіоном	3) Відкривається меню вибору типу забруднення
	4)Натиснути кнопку з цікавляим типом забруднення	4)Відкривається заявка за обраними параметрами та пропонує відправити фото підтвердження усунення проблеми
	5)Відправити боту фото	5) Отриманий результат відправляється на модерацію

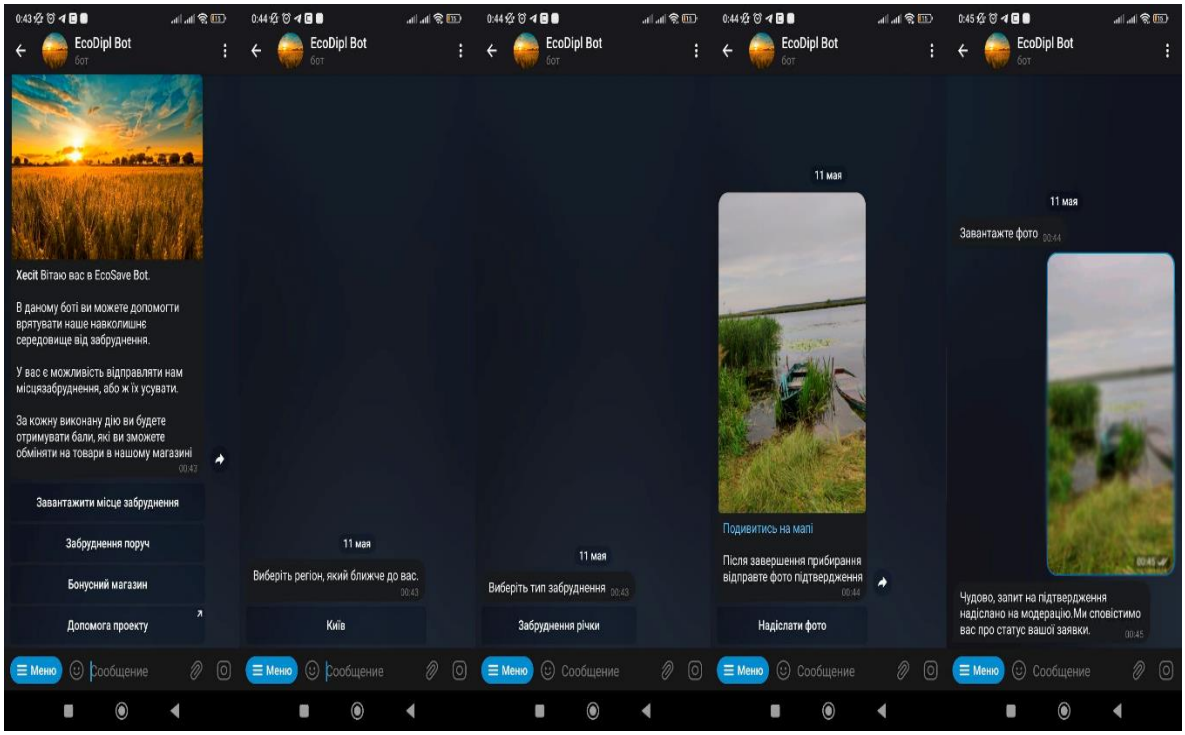


Рис. 3.20. – Перегляд та вирішення завантажених заявок.

Таблиця 3.4. – Тест кейс модерації заявок.

Заголовок	Крок	Очікуваний результат
Модерація заявок (Рис. 3.21)	1)Заявка відправляється у чат модерації	1) у чаті модерації відображається нова заявка
	2)Натиснути кнопку «Підтвердити»	2)Відображається меню для введення регіону.
	3)Відправити назву регіону	3) Відображається меню для введення назви типу забруднення
	4)Відправити назву типу забруднення	4)Заявка приймається і користувачу подавшому дану заявку нараховуються бонусні бали

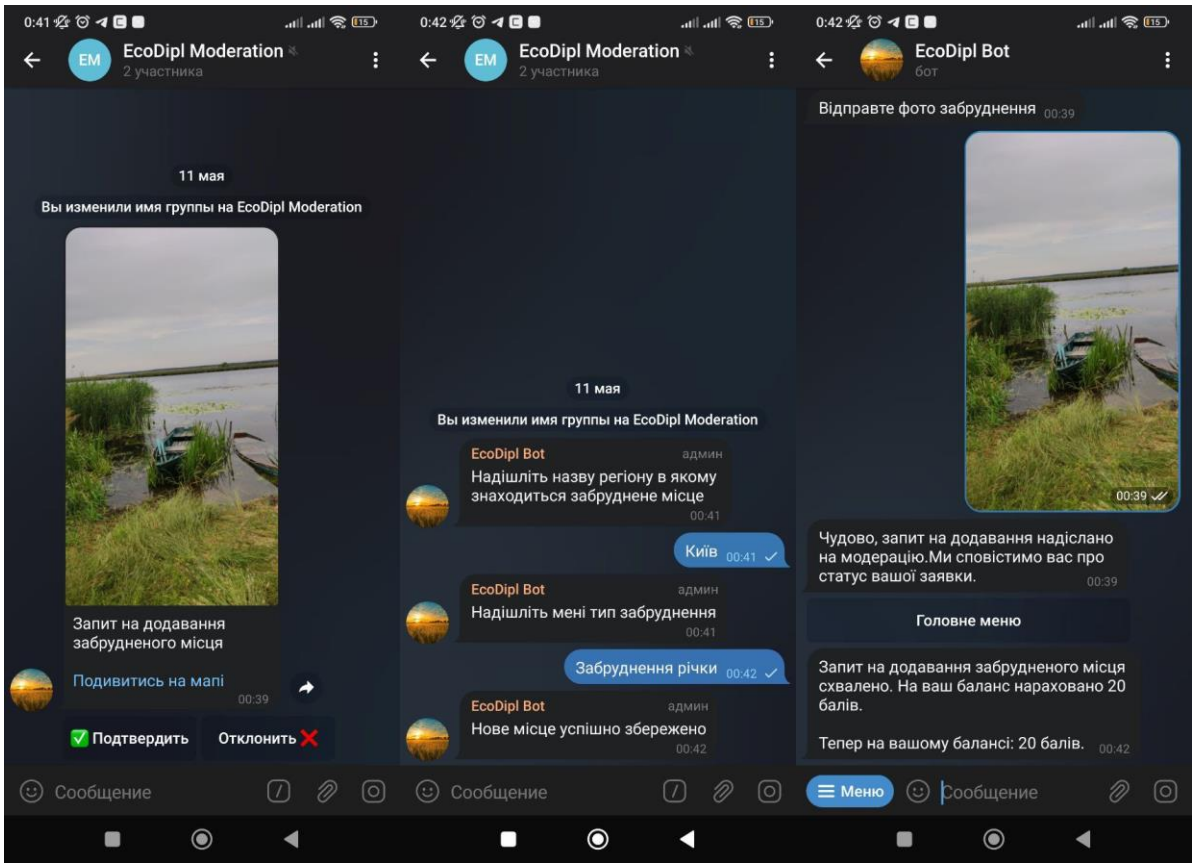


Рис. 3.21. – Модерація отриманих заявок.

Таблиця 3.5. – Тест кейс роботи бонусного магазину

Заголовок	Крок	Очікуваний результат
Робота бонусного магазину (Рис. 3.22)	1)Ввести команду «/start»	1)З`являється головне меню
	2)Натиснути кнопку «Бонусний магазин»	2)Відкриється меню бонусного магазину
	3)Натиснути кнопку з цікавлячим товаром	3) З'являється повідомлення з проханням ввести ім`я користувача
	4)Ввести ім`я користувача	4)Відкривається меню з проханням ввести номер телефону
	5)Натиснути кнопку «Відправити контакт»	5) Прикріпляється контакт і замовлення відправляється в групу модерації для подальшої обробки

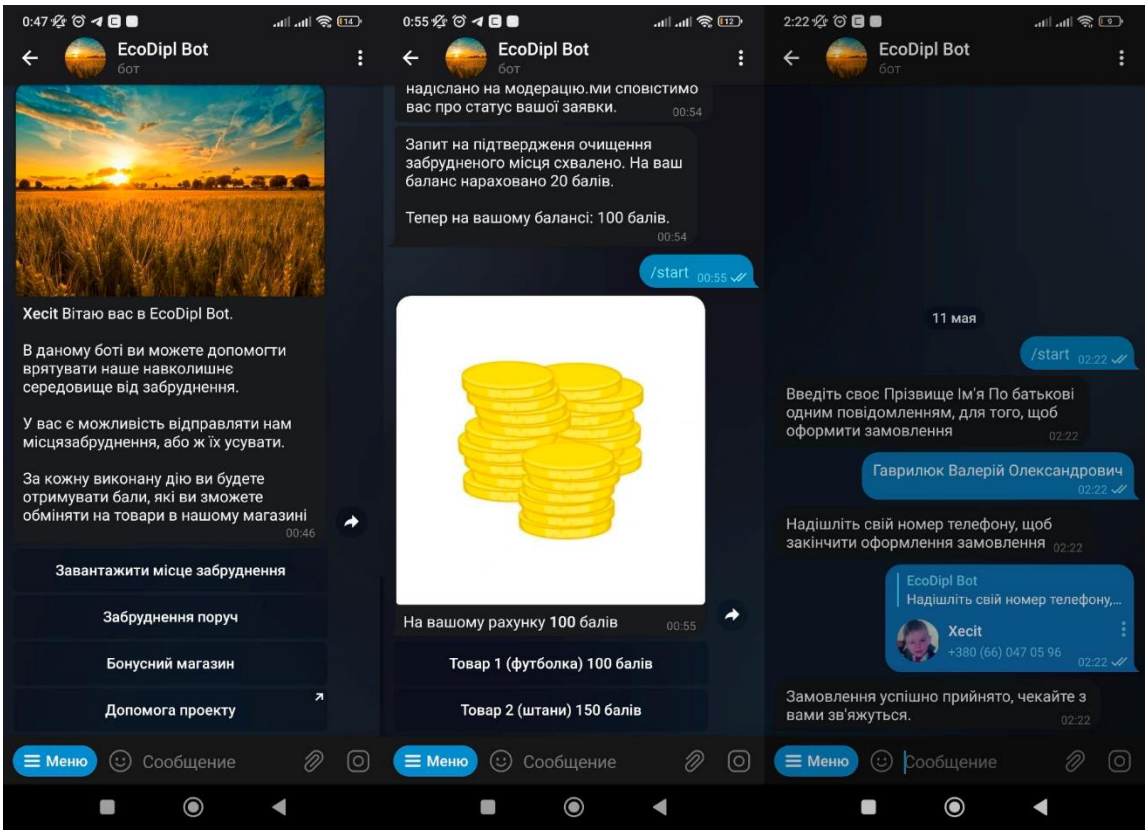


Рис. 3.22. – Робота бонусного магазину.

ВИСНОВКИ

В ході виконання випускної кваліфікаційної роботи був розроблений чат-бот для моніторингу стану навколишнього середовища. Було проведено аналіз предметної області, розглянуто існуючі аналоги розроблюваного продукту та сформовано вимоги до чат-боту.

Під час проведення аналізу предметної області були обрані та описані засоби для розробки чат-ботів, такі як: мова програмування Python, асинхронний фреймворк Aiogram, хмарна платформа Google Cloud Platform та система управління базами даних PostgreSQL.

Додаток є актуальним для волонтерських компаній з охорони природи та будь яких небайдужих людей до глобальної проблеми забруднення навколишнього середовища. Даний чат-бот має мету спростити та дати мотивацію людям усувати екологічні проблеми, за допомогою можливості особисто подавати та опрацьовувати заявки, отримуючи в результаті нагороду за виконані дії.

В результаті було спроектовано та розроблено чат-бота для моніторингу стану навколишнього середовища на базі месенджера Telegram, який відповідає всім поставленим вимогам і готовий для подальшого використання.

У подальшому розвитку проекта планується:

1) Перенести бота в інші месенджери.
2) Автоматизація етапу модерації, за допомогою впровадження штучного інтелекту.
3) Підтримка інших мов.

ПЕРЕЛІК ПОСИЛАНЬ

1. What is ChatBot and Why is it important? [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.techtarget.com/searchcustomerexperience/definition/chatbot

2. Telegram FAQ [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://telegram.org/faq#q-what-is-telegram-what-do-i-do-here

3. Most popular messanging apps|Statista [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps

4. As WhatsApp Tops 2 Billion Users [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.wsj.com/articles/as-whatsapp-tops-2-billion-users-ceo-vows-to-defend-encryption-11581516000

5. WhatsApp Desktop Client for Windows & Mac [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.makeuseof.com/tag/whatsapp-desktop-client-windows-mac-second-best/

6. Facebook to buy whatsapp for 16 billion [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.wsj.com/articles/facebook-to-buy-whatsapp-for-16-billion-1392847766

7. Facebook Inc.'s WhatsApp Hits 900 Million Users: What Now? | The Motley Fool [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: http://www.fool.com/investing/general/2015/09/11/facebook-incs-whatsapp-hits-900-million-users-what.aspx

8. Telegram FAQ [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://telegram.org/faq

9. Encrypted Messaging App Telegram Hits 100M Monthly Active Users, 350k New Users Each Day | TechCrunch [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://techcrunch.com/2016/02/23/encrypted-messaging-app-telegram-hits-100m-monthly-active-users-350k-new-users-each-day/

10. What are Chatbots? How do Chatbots work? Top 10 applications | Engati [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.engati.com/blog/what-are-chatbots

11. Internet History of 1970s | Internet History | Computer History Museum [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.computerhistory.org/internethistory/1970s/

12. What Is Machine Learning and Why Is It Important? [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML

13. ОФІЦІЙНИЙ ВЕБ-САЙТ ХАРКІВСЬКА ОБЛАСНА ДЕРЖАВНА АДМІНІСТРАЦІЯ [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://kharkivoda.gov.ua/news/111699

14. Про проєкт - SaveEcoBot [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.saveecobot.com/static/about

15. General Python FAQ — Python 3.10.4 documentation [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://docs.python.org/3/faq/general.html#what-is-python

16. PythonTutor [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://pythontutor.com/

17. The Future Scope of Python Programming Language - Coding Elements [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.codingelements.com/blog/top-reasons-behind-increasing-demand-for-python-programming

18. PostgreSQL: Documentation: 14: PostgreSQL 14.3 Documentation [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.postgresql.org/docs/14/index.html

19. PostgreSQL: Support [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.postgresql.org/support/

20. PostgreSQL: About [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.postgresql.org/about/

21. What is Google Cloud? [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.techtarget.com/searchcloudcomputing/definition/Google-Cloud-Platform

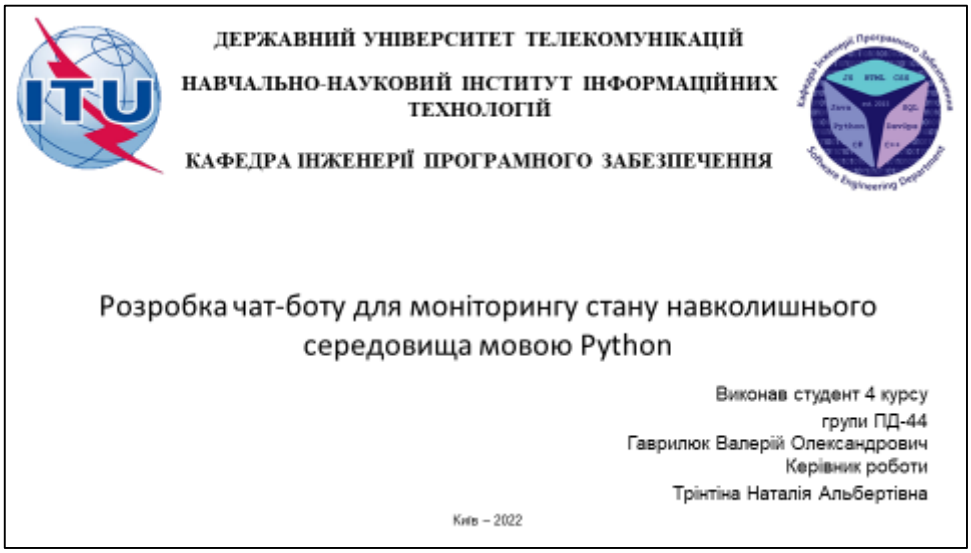
22. What is a block diagram? - Knolwledge Base [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.microtool.de/en/knowledge-base/what-is-a-block-diagram/

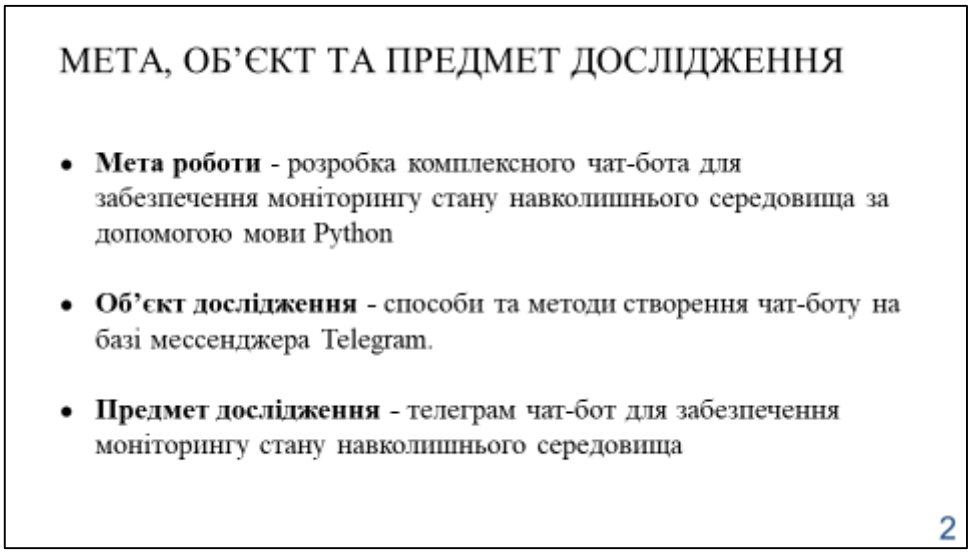
23. What is Block Diagram – Everything You Need to Know | EdrawMax Online [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.edrawmax.com/block-diagram/

24. Bots: An introduction for developers [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://core.telegram.org/bots#6-botfather

25. What Is a Test Case? Examples, Types and Format - Applause [Електронний ресурс]: [Веб–сайт]. – електронні дані. – Режим доступу: https://www.applause.com/blog/what-is-a-test-case-examples-types-format

ДОДАТОК А


ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
Розробка чат-боту для моніторингу стану навколишнього середовища мовою Python
Виконав студент 4 курсу
групи ПД-44
Гаврилюк Валерій Олександрович
Керівник роботи
Трінтіна Наталія Альбертівна
Київ – 2022


МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ
• Мета роботи - розробка комплексного чат-бота для забезпечення моніторингу стану навколишнього середовища за допомогою мови Python
• Об'єкт дослідження - способи та методи створення чат-боту на базі мессенджера Telegram.
• Предмет дослідження - телеграм чат-бот для забезпечення моніторингу стану навколишнього середовища
2

АНАЛОГИ

SaveEcoBot

Переваги

Недоліки

UkrForest_Bot

Переваги

Недоліки

ТЕХНІЧНІ ЗАВДАННЯ

- *Подання нових заяв користувачем за допомогою геолокації та фото матеріалів;*
- *Перегляд користувачами завантажених заяв та надання можливості їх опрацювання;*
- *Модерація отриманих заяв;*
- *Мотивація користувачам, шляхом впровадження бонусного магазину.*

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

PostgreSQL

Google Cloud Platform

Алгоритм подання нових заяв

6

Алгоритм перегляду та опрацювання існуючих заяв

7

Алгоритм модерації заяв

8

Алгоритм роботи бонусного магазину

9

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1) Гаврилюк В.О., Роль інформаційних технологій у захисті та відновленні навколишнього середовища / XIII Міжнародна науково-технічна конференція студентства та молоді «СВІТ ІНФОРМАЦІЇ ТА ТЕЛЕКОМУНІКАЦІЙ». Збірник тез 21.10.2021, ДУТ, м. Київ – К.: ДУТ, 2021. С. 162-164.

2) Гаврилюк В.О., Особливості та проблеми спілкування в мережі за домогою чат-ботів / XIII Міжнародна науково-технічна конференція студентства та молоді «СВІТ ІНФОРМАЦІЇ ТА ТЕЛЕКОМУНІКАЦІЙ». Збірник тез 21.10.2021, ДУТ, м. Київ – К.: ДУТ, 2021. С. 159-160.

10

ВИСНОВКИ

1. Було обгрунтовано актуальність та новизну розробленого чат-бота.
2. Було проведено аналіз предметної галузі.
3. Розроблено алгоритми роботи системи.
4. Оглянуто основні механізми роботи
5. Проведенено тестування

Перспективи подальшого розвитку:

1) Перенесення бота в інші мессенджери.

2)Автоматизація етапу модерації, за допомогою впровадження штучного інтелекту.

3)Підтримка інших мов.

11

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК Б

```
BOT_TOKEN=5169161949:AAGiUhEQQGGZu1Eo0purvz1TKdLZN_2YBb4

DB_PASS=root
DB_USER=postgres
DB_NAME=eco_places
DB_HOST=127.0.0.1:5432
# -*- coding: utf-8 -*-

from environs import Env
from string import Template

env = Env()
env.read_env()


BOT_TOKEN = env.str("BOT_TOKEN")

DB_NAME = env.str("DB_NAME")
DB_HOST = env.str("DB_HOST")
DB_USER = env.str("DB_USER")
DB_PASS = env.str("DB_PASS")


MODER_TELEGRAM_GROUP_ID = -1001690144817
PROJECT_DONAT_PAGE = "https://bank.gov.ua/ua/about/support-the-armed-forces"
LOCATION_TEMPLATE =
Template("https://www.google.com/maps/place/$latitude,$longitude")
POINTS_PER_ACTION = 20
# -*- coding: utf-8 -*-

from .models import db
from data.config import DB_NAME, DB_HOST, DB_USER, DB_PASS


async def create_db(drop_all):
    await db.set_bind(f"postgresql://{DB_USER}:{DB_PASS}@{DB_HOST}/{DB_NAME}")

    if drop_all:
        await db.gino.drop_all()
        await db.gino.create_all()
# -*- coding: utf-8 -*-

from gino import Gino
from sqlalchemy import Column, Sequence, BigInteger, Float, String


db = Gino()




class User(db.Model):
    __tablename__ = "user_eco"
    db_id = Column(BigInteger, Sequence(name="user_id_seq"), primary_key=True)
    tg_id = Column(BigInteger)
    balance = Column(BigInteger)
```

```
class Place(db.Model):
    __tablename__ = "place"
    db_id = Column(BigInteger, Sequence(name="place_id_seq"), primary_key=True)
    latitude = Column(Float)
    longitude = Column(Float)
    photo_id = Column(String)
    type_pollution = Column(BigInteger)
    place_name_id = Column(BigInteger)


class PlaceName(db.Model):
    __tablename__ = "placename"
    db_id = Column(BigInteger, Sequence(name="placename_id_seq"),
primary_key=True)
    name = Column(String)


class PollutionType(db.Model):
    __tablename__ = "pollutiontype"
    db_id = Column(BigInteger, Sequence(name="pollutiontype_id_seq"),
primary_key=True)
    name = Column(String)


class CompletedPlace(db.Model):
    __tablename__ = "completedplace"
    db_id = Column(BigInteger, Sequence(name="completedplace_id_seq"),
primary_key=True)
    place_id = Column(BigInteger)
    photo_id = Column(String)
```

```
# -*- coding: utf-8 -*-

from .models import CompletedPlace


async def get(db_id: int) -> CompletedPlace:
    """
    Получаем запись о очищенном от загрязнений месте

    Аргументы:
        db_id (int): id записи

    Возвращается:
        CompletedPlace: класс
    """

    record = await CompletedPlace.query.where(
        CompletedPlace.db_id == db_id,
    ).gino.first()

    return record


async def create(place_id: int, photo_id: str) -> CompletedPlace:
    """
    Создаем новую запись о очищенном месте в базе данных

    Если такая запись уже есть, просто возвращает ее

    Аргументы:
        place_id (int): id места в базе данных
```

```
        photo_id (str): id фото в Телеграмме

    Возвращает:
        CompletedPlace: класс
    """

    record = await CompletedPlace(
        place_id=place_id,
        photo_id=photo_id
    ).create()

    return record


# -*- coding: utf-8 -*-

from .models import Place
from sqlalchemy import and_


async def get(db_id: int) -> Place:
    """
    Получает запись о загрзненном месте из базы данных

    Аргументы:
        db id (int): id записи в базе данных

    Возвращается:
        Place: класс загрязненного места
    """

    return await Place.query.where(
        Place.db_id == db_id
    ).gino.first()


async def create(latitude: float, longitude: float, photo_id: str) -> Place:
    """
    Создает запись в базе данных о месте загрязнения

    Если место уже есть в базе данных, то создание пропускается, и
    возвращается ранее созданная запись

    Агрументы:
        latitude (float): широта нахождения места
        longitude (float): долгота нахождения места
        photo_id (str): id фото места

    Возвращается:
        Place: класс загрязненного места
    """

    record = await Place.query.where(
        and_(
            Place.latitude == latitude,
            Place.longitude == longitude,
        ),
    ).gino.first()

    if record:
        return record

    record = await Place(
```

```
        latitude=latitude,
        longitude=longitude,
        photo_id=photo_id,
    ).create()

    return record


async def update_place_name(db_id: int, place_name_id: str) -> Place:
    """
    Обновляет id названия региона загрязненного места

    Агрументы:
        db_id (int): id места из базы данных
        place_name_id (str): id названия региона из базы данных

    Возвращается:
        Place: класс загрязненного места
    """

    record = await get(db_id)
    record = await record.update(
        place_name_id=place_name_id,
    ).apply()

    return record


async def update_type_pollution(db_id: int, type_pollution: str) -> Place:
    """
    Обновляет тип загрязнения присутствующий на месте

    Аргументы:
        db_id (int): id из места из базы данных
        type_pollution (str): тип загрязнения

    Возвращается:
        Place: класс загрязненного места
    """

    record = await get(db_id)
    record = await record.update(
        type_pollution=type_pollution,
    ).apply()

    return record


async def delete(db_id: int):
    """
    Удаляет запись о загряненном месте из базы данных

    Аргументы:
        db_id (int): id записи загрязненного места
    """

    record = await get(db_id)
    await record.delete()


async def get_all_with_region_and_type(region_id: int, pollution_type_id: int) ->
list:
    """
```

```
    Получет список всех мест в указанном регионе и типе загрязнения

    Аргументы:
        region_id (int): id региона в базе данных
        pollution_type_id (int): id типа загрязнения в базе данных

    Возвращается:
        list: список записей
    """

    records = await Place.query.where(
        and_(
            Place.place_name_id == region_id,
            Place.type_pollution == pollution_type_id,
        )
    ).gino.all()

    return records

# -*- coding: utf-8 -*-

from .models import PlaceName


async def create(name: str) -> PlaceName:
    """
    Создает запись о регион в базе данных
    Если запись уже существует, тогда просто возвращает ее

    Аргументы:
        name (str): название региона

    Возвращается:
        PlaceName: класс названия региона
    """

    record = await get(name=name)

    if not record:
        record = await PlaceName(
            name=name,
        ).create()

    return record


async def get(name=None, db_id=None) -> PlaceName:
    """
    Получает запись о названии места из базы данныых

    Аргументы:
        name (_type_, optional): название места.
        db_id (_type_, optional): id места из базы данных.

    Возвращается:
        PlaceName: класс названия места.
    """

    if name:
        return await PlaceName.query.where(
            PlaceName.name == name,
        ).gino.first()
```

```
    if db_id:
        return await PlaceName.query.where(
            PlaceName.db_id == db_id,
        ).gino.first()


async def get_all() -> list:
    """
    Получает все записи название мест из базы данных

    Возвращается:
        list: список записей
    """

    records = await PlaceName.query.gino.all()
    return records
# -*- coding: utf-8 -*-

from .models import PollutionType, Place


async def create(name: str) -> PollutionType:
    """
    Создает запись о типе загрязнения в базе данных
    Если запись уже существует, тогда просто возвращает ее

    Аргументы:
        name (str): название загрязнения

    Возвращается:
        PollutionType: класс названия загрязнения
    """

    record = await get(name=name)

    if not record:
        record = await PollutionType(
            name=name,
        ).create()

    return record


async def get(name=None, db_id=None) -> PollutionType:
    """
    Получает запись о названии типа загрязнения из базы данныых

    Аргументы:
        name (_type_, optional): название типа загрязнения.
        db_id (_type_, optional): id места из базы данных.

    Возвращается:
        PollutionType: класс типа загрязнения.
    """

    if name:
        return await PollutionType.query.where(
            PollutionType.name == name,
        ).gino.first()

    if db_id:
        return await PollutionType.query.where(
            PollutionType.db_id == db_id,
```

```
        ).gino.first()


async def get_all() -> list:
    """
    Получает все записи тиво загрязнения из базы данных

    Возвращается:
        list: список записей
    """

    records = await PollutionType.query.gino.all()
    return records


async def get_all_with_region(region_id: int) -> list:
    """
    Получает доступные типы загрязнения в указанном регионе

    Аргументы:
        region_id (int): id региона из которого нужно получить доступные типы

    Возвращатеся:
        list: список записей
    """

    places = await Place.query.where(
        Place.place_name_id == region_id,
    ).gino.all()

    pollution_types = [
        place.type_pollution
        for place in places
    ]

    pollution_id = list(set(pollution_types))
    pollution_types = [
        await get(db_id=id_)
        for id_ in pollution_id
    ]

    return pollution_types
# -*- coding: utf-8 -*-

from .models import User


async def get(tg_id=None, db_id=None) -> User:
    """
    Получает пользователя из базы данные

    В функции можно использовать как id пользователя в Телеграмме
    так и id пользователя в базе данных.

    Args:
        tg_id (_type_, optional): id пользователя в Телеграмме. Defaults to None.
        db_id (_type_, optional): id пользователя в базе данных. Defaults to None.

    Возвращается:
        User: Класс пользователя из базы данных.
    """
    if tg_id:
        return await User.query.where(
```

```
            User.tg_id == tg_id
        ).gino.first()

    if db_id:
        return await User.query.where(
            User.db_id == db_id,
        ).gino.first()


async def create(tg_id: int) -> User:
    """
    Создает пользователя в базе данных

    Если пользователь есть в базе данных, то функция просто вернет его
    без повторного добавления в базу данных

    Аргументы:
        tg_id (int): id пользователя в Телеграмме

    Возвращается:
        User: _description_
    """

    record = await get(tg_id=tg_id)

    if record:
        return record
    else:
        record = await User(
            tg_id=tg_id,
            balance=0,
        ).create()
        return record


async def add_to_balance(amount: int, db_id=None, tg_id=None) -> User:
    """
    Добавляет к текущему балансу пользователя установленное значение

    Аргументы:
        amount (int): число которое добавляет к балансу пользователя
        db id ( type , optional): id пользователя в базе данных
        user_tg_id (_type_, optional): id пользователя в Телеграмме

    Возвращается:
        User: класс пользователя
    """

    if db_id:
        user = await get(db_id=db_id)

    if tg_id:
        user = await get(tg_id=tg_id)

    await user.update(
        balance=user.balance + amount,
    ).apply()

    record = await get(db_id=user.db_id)

    return record
```

```python
async def take_off_balance(amount: int, db_id=None, tg_id=None) -> User:
    """
    Отнимает от текущего баланса пользователя установленное значение

    Аргументы:
        amount (int): число которое отнимется от баланса пользователя
        db_id (_type_, optional): id пользователя в базе данных
        user_tg_id (_type_, optional): id пользователя в Телеграмме

    Возвращается:
        User: класс пользователя
    """

    if db_id:
        record = await get(db_id=db_id)

    if tg_id:
        record = await get(tg_id=tg_id)

    record = await record.update(
        balance=record.balance - amount,
    ).apply()

    return record
# -*- coding: utf-8 -*-

from aiogram import types
from loader import dp, bot
from aiogram.dispatcher import FSMContext
from data.config import LOCATION_TEMPLATE, MODER_TELEGRAM_GROUP_ID
from statesgroup import CompletePlace
from keyboards import inline, ctypes
from database import place_api


@dp.callback_query_handler(ctypes.place_menu.filter(button="send_photo"))
async def on_process_send_complete_photo(
        call: types.CallbackQuery,
        state: FSMContext,
        callback_data: dict,
    ):
    """ Получаем фото чистого места """

    await CompletePlace.get_place_photo.set()
    async with state.proxy() as data:
        data['place_id'] = callback_data['id']

    await call.message.delete()
    await call.message.answer(
        text="Завантажте фото",
    )


@dp.message_handler(
    state=CompletePlace.get_place_photo,
    content_types=types.ContentType.ANY,
)
async def on_process_get_photo(
        message: types.Message,
        state: FSMContext,
    ):
    """ Получили фото очищенного места, отправляем на модерацию """
```

```
    async with state.proxy() as data:
        place_id = int(data['place_id'])
        photo_id = message.photo[-1].file_id

    place = await place_api.get(place_id)
    location_url = LOCATION_TEMPLATE.substitute(
        latitude=place.latitude,
        longitude=place.longitude,
    )
    await bot.send_photo(
        chat_id=MODER_TELEGRAM_GROUP_ID,
        photo=photo_id,
        reply_markup=await inline.completed_menu(place_id, message.from_user.id),
        caption="Запит на підтвердження очищеного" +\
                "від забруднень місця.\n\n" +\
                f"<a href='{location_url}'>Подивитися на карті</a>",
    )

    await message.answer(
        text="Чудово, запит на підтвердження надіслано на модерацію." +\
            "Ми сповістимо вас про статус вашої заявки."
    )
# -*- coding: utf-8 -*-

from aiogram import types
from loader import dp, bot
from keyboards import ctypes
from database import place_api, user_api
from data.config import POINTS_PER_ACTION


@dp.callback_query_handler(ctypes.completed_menu.filter(action="reject"))
async def on_process_moder_reject(call: types.CallbackQuery, callback_data: dict):
    """ Отклонение запроса на подверждение очисти загрязненного места """

    user_tg_id = int(callback_data['user_tg_id'])
    await call.message.delete()
    await bot.send_message(
        chat_id=user_tg_id,
        text="Запит на підтвердженя очищення забрудненого місця відхилено",
    )
    await call.message.answer(
        text="Запит на підтвердженя очищення забрудненого місця успішно
відхилений."
    )
    await call.answer()


@dp.callback_query_handler(ctypes.completed_menu.filter(action="confirm"))
async def on_process_moder_confirm(call: types.CallbackQuery, callback_data:
dict):
    """ Принятие запроса на подверждение очисти загрязненного места """

    place_id = int(callback_data['place_id'])
    user_tg_id = int(callback_data['user_tg_id'])
    await place_api.delete(place_id)
    await call.message.delete()

    user = await user_api.add_to_balance(
        tg_id=user_tg_id,
        amount=POINTS_PER_ACTION,
    )
```

```
    await bot.send_message(
        chat_id=user_tg_id,
        text="Запит на підтвердженя очищення забрудненого місця схвалено. " +\
            f"На ваш баланс нараховано {POINTS_PER_ACTION} балів. \n\n" +\
            f"Тепер на вашому балансі: {user.balance} балів.",
    )

    await call.message.answer(
        text="Запит на підтвердженя очищення забрудненого місця схвалено.",
    )
# -*- coding: utf-8 -*-

from aiogram import types
from loader import dp, bot
from keyboards import ctypes
from statesgroup import ModerationPlace
from aiogram.dispatcher import FSMContext
from data.config import POINTS_PER_ACTION
from database import place_api, place_name_api, user_api, pollution_type_api


@dp.callback_query_handler(ctypes.moder_menu.filter(action="reject"))
async def on_process_moder_reject(call: types.CallbackQuery, callback_data: dict):
    """ Отклонение запроса на добавление загрязненного места """

    place_id = int(callback_data['place_id'])
    user_tg_id = int(callback_data['user_tg_id'])
    await place_api.delete(place_id)

    await call.message.delete()
    await bot.send_message(
        chat_id=user_tg_id,
        text="Запит на додавання забрудненого місця відхилено",
    )
    await call.message.answer(
        text="Запит на додавання забрудненого місця успішно відхилений."
    )
    await call.answer()


@dp.callback_query_handler(ctypes.moder_menu.filter(action="confirm"))
async def on_process_moder_confirm(
        call: types.CallbackQuery,
        state: FSMContext,
        callback_data: dict
    ):
    """ Підтвердження заявки"""

    await ModerationPlace.get_place_name.set()
    async with state.proxy() as data:
        data['place_id'] = callback_data['place_id']
        data['user_tg_id'] = callback_data['user_tg_id']

    await call.message.delete()
    await call.message.answer(
        text="Надішліть назву регіону в якому знаходиться забруднене місце",
    )
    await call.answer()


@dp.message_handler(state=ModerationPlace.get_place_name)
async def on_process_get_place_name(
        message: types.Message,
```

```
        state: FSMContext,
    ):
    """ Получение названия региона """

    async with state.proxy() as data:
        data['place_name'] = message.text

    await ModerationPlace.get_type_pollution.set()
    await message.answer(
        text="Надішліть мені тип забруднення",
    )


@dp.message_handler(state=ModerationPlace.get_type_pollution)
async def on_process_get_type_pollution(
        message: types.Message,
        state: FSMContext,
    ):
    """ Запис ухваленої заявки у базу даних"""

    async with state.proxy() as data:
        place_name = data['place_name']
        place_id = int(data['place_id'])
        user_tg_id = int(data['user_tg_id'])
        pollution_name = message.text

    type_pollution = await pollution_type_api.create(pollution_name)
    place_name = await place_name_api.create(place_name)
    await place_api.update_place_name(place_id, place_name.db_id)
    await place_api.update_type_pollution(place_id, type_pollution.db_id)

    user = await user_api.add_to_balance(
        tg_id=user_tg_id,
        amount=POINTS_PER_ACTION,
    )

    await bot.send_message(
        chat_id=user_tg_id,
        text="Запит на додавання забрудненого місця схвалено. " +\
            f"На ваш баланс нараховано {POINTS_PER_ACTION} балів. \n\n" +\
            f"Тепер на вашому балансі: {user.balance} балів.",
    )
    await state.finish()
    await message.answer(
        text="Нове місце успішно збережено",
    )

# -*- coding: utf-8 -*-

import asyncio
from aiogram import types
from loader import dp
from keyboards import ctypes, inline
from aiogram.dispatcher import FSMContext
from data.config import LOCATION_TEMPLATE
from database import place_api, place_name_api, pollution_type_api


@dp.callback_query_handler(ctypes.start_menu.filter(button="near_places"))
async def on_process_show_regions(call: types.CallbackQuery):
    """ Выбирем регион """

    regions = await place_name_api.get_all()
```

```
    await call.message.delete()
    await call.message.answer(
        text="Виберіть регіон, який ближче до вас.",
        reply_markup=await inline.regions_menu(regions),
    )
    await call.answer()


@dp.callback_query_handler(ctypes.regions_menu.filter())
async def on_process_show_types(
        call: types.CallbackQuery,
        state: FSMContext,
        callback_data: dict,
    ):
    """ Вибір регіону """

    async with state.proxy() as data:
        data['region_id'] = callback_data['region_id']

    pollution_types = await pollution_type_api.get_all_with_region(
        region_id=int(callback_data['region_id']),
    )

    await call.message.edit_text(
        text="Виберіть тип забруднення",
        reply_markup=await inline.pollution_menu(pollution_types),
    )


@dp.callback_query_handler(ctypes.pollution_menu.filter())
async def on_process_show_places(
        call: types.CallbackQuery,
        state: FSMContext,
        callback_data: dict,
    ):
    """ Перегляд місць в обраному регіоні """

    async with state.proxy() as data:
        region_id = int(data['region_id'])
        pollution_type_id = int(callback_data['type_id'])

    places = await place_api.get_all_with_region_and_type(
        region_id=region_id,
        pollution_type_id=pollution_type_id,
    )

    await call.message.delete()
    for place in places:
        location_url = LOCATION_TEMPLATE.substitute(
            latitude=place.latitude,
            longitude=place.longitude,
        )

        await call.message.answer_photo(
            photo=place.photo_id,
            caption=f"<a href='{location_url}'>Подивитись на мапі</a>\n\n" +\
                "Після завершення прибирання відправте фото підтвердження",
            reply_markup=await inline.place_menu(place.db_id),
        )

        await asyncio.sleep(1)
    await call.answer()
```

```python
# -*- coding: utf-8 -*-

from aiogram import types
from loader import dp, bot
from keyboards import ctypes, inline, reply
from database import user_api
from aiogram.dispatcher import FSMContext
from statesgroup import Checkout
from data.config import MODER_TELEGRAM_GROUP_ID


@dp.callback_query_handler(ctypes.start_menu.filter(button="shop"))
async def on_process_shop(call: types.CallbackQuery):
    """ Товари магазину """

    user = await user_api.get(tg_id=call.message.chat.id)

    await call.message.delete()
    await call.message.answer_photo(
        photo="AgACAgIAAxkBAAIDsGJ63kwrzfdIHIQrl38Ksk6uHbILAAKHvjEbFNHYS-
SAQVGs0qXMAQADAgADcwADJAQ",
        caption=f"На вашому рахунку <b>{user.balance}</b> балів",
        reply_markup=await inline.shop_menu(),
    )

    await call.answer()


@dp.callback query handler(ctypes.shop menu.filter())
async def on_process_item(
        call: types.CallbackQuery,
        state: FSMContext,
        callback_data: dict,
    ):
    """ Оформлення замовлення """

    await Checkout.get_initials.set()
    async with state.proxy() as data:
        data['item'] = callback_data['item']

    user = await user_api.get(tg_id=call.message.chat.id)
    if user.balance < 100:
        await state.finish()
        await call.message.delete()
        delete_message = await call.message.answer(
            text=". . .",
            reply_markup=types.ReplyKeyboardMarkup(),
        )

        await delete_message.delete()
        return await call.message.answer(
            text="У вас недостатньо балів на балансі",
            reply_markup=await inline.back_to_main_menu(),
        )

    await call.message.delete()
    await call.message.answer(
        text="Введіть своє Прізвище Ім'я По батькові одним повідомленням, " +\
             "для того, щоб оформити замовлення",
    )

    await call.answer()
```

```python
@dp.message_handler(state=Checkout.get_initials)
async def on_process_get_initials(message: types.Message, state: FSMContext):
    """ Отримуємо дані користувача """

    await Checkout.get_contact.set()
    async with state.proxy() as data:
        data['initials'] = message.text

    await message.answer(
        text="Надішліть свій номер телефону, " +\
            "щоб закінчити оформлення замовлення",
        reply_markup=await reply.send_contact()
    )


@dp.message_handler(
    state=Checkout.get_contact,
    content_types=types.ContentType.CONTACT,
)
async def on_process_get_contact(message: types.Message, state: FSMContext):
    """ Оформляем заказ """

    async with state.proxy() as data:
        initials = data['initials']
        item = data['item']
        phone_number = message.contact.phone_number

    amount = 100 \
        if item == "Футболка" \
        else 150

    await user_api.take_off_balance(
        tg_id=message.contact.user_id,
        amount=amount,
    )

    await bot.send_message(
        chat_id=MODER_TELEGRAM_GROUP_ID,
        text=f"{initials}, сделал заказ на '{item}'\n\n" +\
            f"Данные для связи: {phone_number}",
    )
    await message.answer(
        text="Замовлення успішно прийнято, чекайте з вами зв'яжуться.",
        reply_markup=types.ReplyKeyboardRemove(),
    )

    await state.finish()
# -*- coding: utf-8 -*-

from aiogram import types
from loader import dp
from database import user_api
from keyboards import inline
from aiogram.dispatcher import FSMContext
from aiogram import types


@dp.message_handler(commands=['start'], state="*")
async def on_process_start(message: types.Message, state: FSMContext):
    """ Привітальне повідомлення та перевірка наявності користувача в базі даних
"""
```

```
    await state.finish()
    user = await user_api.create(message.from_user.id)

    username = message.from_user.username
    if not username:
        username = message.from_user.first_name

    delete_keyboard_message = await message.answer(
        text=". . . ",
        reply_markup=types.ReplyKeyboardRemove(),
    )
    await delete_keyboard_message.delete()


    await message.answer_photo(

photo="AgACAgIAAxkBAAIDdWJ274AItMlt9bhHLzdN5Smxm814AAIcuDEbzdC5S9_cWIVWnKP1AQADAgA
DcwADJAQ",
        reply_markup=await inline.start_menu(),
        caption=f"<b>{username}</b> Вітаю вас в EcoDipl Bot.\n\n" +\
            "В даному боті ви можете допомогти врятувати "
            "наше навколишнє середовище від забруднення.\n\n"
            "У вас є можливість відправляти нам місця"
            "забруднення, або ж їх усувати.\n\n"
            "За кожну виконану дію ви будете отримувати бали, "
            "які ви зможете обміняти на товари в нашому магазині"
    )
# -*- coding: utf-8 -*-

from aiogram import types
from loader import dp, bot
from keyboards import ctypes, inline, reply
from statesgroup import UploadPlace
from aiogram.dispatcher import FSMContext
from database import place_api
from data.config import MODER_TELEGRAM_GROUP_ID, LOCATION_TEMPLATE


@dp.callback_query_handler(ctypes.start_menu.filter(button='upload_place'))
async def on_process_upload_place_step_1(call: types.CallbackQuery):
    """ Відправка повідомлення про прикріплення геолокації забредненого місця """

    await UploadPlace.get_place_geo.set()
    await call.message.delete()
    await call.message.answer(
        text="Відправте свою геолокацію біля місця забруднення",
        reply_markup=await reply.get_location(),
    )

    await call.answer()


@dp.message_handler(
    state=UploadPlace.get_place_geo,
    content_types=types.ContentType.LOCATION,
)
async def on_process_upload_place_step_2(
        message: types.Message,
        state: FSMContext
    ):
    """
    Відправка повідомлення про прікріплення фото забредненого місця
    """
```

```
    await UploadPlace.get_place_photo.set()
    async with state.proxy() as data:
        data['latitude'] = message.location.latitude
        data['longitude'] = message.location.longitude

    await message.answer(
        text="Відправте фото забруднення",
        reply_markup=types.ReplyKeyboardRemove(),
    )


@dp.message_handler(
    state=UploadPlace.get_place_photo,
    content_types=types.ContentType.ANY,
)
async def on_process_upload_place_step_3(
        message: types.Message,
        state: FSMContext,
    ):
    """
   Відправка заявки на модерацію
    """

    async with state.proxy() as data:
        photo_id = message.photo[-1].file_id
        latitude = float(data['latitude'])
        longitude = float(data['longitude'])

    record = await place_api.create(
        latitude=latitude,
        longitude=longitude,
        photo_id=photo_id,
    )

    location_url = LOCATION_TEMPLATE.substitute(
        latitude=latitude,
        longitude=longitude
    )

    await bot.send_photo(
        chat_id=MODER_TELEGRAM_GROUP_ID,
        photo=photo_id,
        caption=f"Запит на додавання забрудненого місця\n\n" +\
            f"<a href='{location_url}'>" +\
            f"Подивитись на мапі</a>",
        reply_markup=await inline.moder_menu(
            db_id=record.db_id,
            user_tg_id=message.from_user.id
        ),
    )

    await state.finish()
    await message.answer(
        text="Чудово, запит на додавання надіслано на модерацію." +\
            "Ми сповістимо вас про статус вашої заявки.",
        reply_markup=await inline.back_to_main_menu(),
    )


@dp.callback_query_handler(ctypes.start_menu.filter(button="back_to_menu"),
state="*", )
async def on_process_back_to_menu(call: types.CallbackQuery):
```

```
    await call.message.delete()

    username = call.from_user.username
    if not username:
        username = call.from_user.first_name

    await call.message.answer_photo(

photo="AgACAgIAAxkBAAIDdWJ274AItMlt9bhHLzdN5Smxm8l4AAIcuDEbzdC5S9_cWIVWnKP1AQADAgA
DcwADJAQ",
        reply_markup=await inline.start_menu(),
        caption=f"<b>{username}</b> Вітаю вас в EcoSave Bot.\n\n" +\
            "В даному боті ви можете допомогти врятувати "
            "наше навколишнє середовище від забруднення.\n\n"
            "У вас є можливість відправляти нам місця"
            "забруднення, або ж їх усувати.\n\n"
            "За кожну виконану дію ви будете отримувати бали, "
            "які ви зможете обміняти на товари в нашому магазині"
    )
# -*- coding: utf-8 -*-

from aiogram.utils.callback_data import CallbackData


start_menu = CallbackData("start_menu", "button")
moder_menu = CallbackData("moder_menu", "action", "user_tg_id", "place_id")
regions_menu = CallbackData("regions_menu", "region_id")
pollution_menu = CallbackData("pollution_menu", "type_id")
place_menu = CallbackData("place_menu", "button", "id")
completed_menu = CallbackData("completed_menu", "action", "user_tg_id",
"place_id")
shop_menu = CallbackData("shop_menu", "item")
# -*- coding: utf-8 -*-

from . import ctypes
from data.config import PROJECT_DONAT_PAGE
from aiogram import types


async def start_menu() -> types.InlineKeyboardMarkup:
    return types.InlineKeyboardMarkup().add(
        types.InlineKeyboardButton(
            text="Завантажити місце забруднення",
            callback_data=ctypes.start_menu.new(
                button="upload_place",
            ),
        )
    ).add(
        types.InlineKeyboardButton(
            text="Забруднення поруч",
            callback_data=ctypes.start_menu.new(
                button="near_places",
            ),
        ),
    ).add(
        types.InlineKeyboardButton(
            text="Бонусний магазин",
            callback_data=ctypes.start_menu.new(
                button="shop",
            ),
        ),
    ).add(
        types.InlineKeyboardButton(
```

```
            text="Допомога проекту",
            url=PROJECT_DONAT_PAGE,
        ),
    )


async def moder_menu(db_id: int, user_tg_id: int) -> types.InlineKeyboardMarkup:
    return types.InlineKeyboardMarkup().row(
        types.InlineKeyboardButton(
            text="✅ Подтвердить",
            callback_data=ctypes.moder_menu.new(
                action="confirm",
                user_tg_id=user_tg_id,
                place_id=db_id,
            ),
        ),
        types.InlineKeyboardButton(
            text="Отклонить ❌",
            callback_data=ctypes.moder_menu.new(
                action="reject",
                user_tg_id=user_tg_id,
                place_id=db_id,
            ),
        ),
    )


async def regions_menu(regions: list) -> types.InlineKeyboardMarkup:
    keyboard = types.InlineKeyboardMarkup()

    for region in regions:
        keyboard.add(
            types.InlineKeyboardButton(
                text=region.name,
                callback_data=ctypes.regions_menu.new(
                    region_id=region.db_id,
                ),
            ),
        )

    return keyboard


async def pollution_menu(pollution_types: list) -> types.InlineKeyboardMarkup:
    keyboard = types.InlineKeyboardMarkup()

    for type_ in pollution_types:
        keyboard.add(
            types.InlineKeyboardButton(
                text=type_.name,
                callback_data=ctypes.pollution_menu.new(
                    type_id=type_.db_id,
                ),
            ),
        )

    return keyboard


async def place_menu(place_id: int) -> types.InlineKeyboardMarkup:
    return types.InlineKeyboardMarkup().add(
        types.InlineKeyboardButton(
```

```
            text="Надіслати фото",
            callback_data=ctypes.place_menu.new(
                button="send_photo",
                id=place_id,
            ),
        ),
    )


async def completed_menu(place_id: int, user_tg_id: int) ->
types.InlineKeyboardMarkup:
    return types.InlineKeyboardMarkup().row(
        types.InlineKeyboardButton(
            text="✅ Подтвердить",
            callback_data=ctypes.completed_menu.new(
                action="confirm",
                user_tg_id=user_tg_id,
                place_id=place_id,
            ),
        ),
        types.InlineKeyboardButton(
            text="Отклонить ❌",
            callback_data=ctypes.completed_menu.new(
                action="reject",
                user_tg_id=user_tg_id,
                place_id=place_id,
            ),
        ),
    )


async def shop_menu():
    return types.InlineKeyboardMarkup().add(
        types.InlineKeyboardButton(
            text="Товар 1 (футболка) 100 балів",
            callback_data=ctypes.shop_menu.new(
                item="Футболка",
            ),
        ),
    ).add(
        types.InlineKeyboardButton(
            text="Товар 2 (штани) 150 балів",
            callback_data=ctypes.shop_menu.new(
                item="Штани",
            ),
        ),
    )


async def back_to_main_menu():
    return types.InlineKeyboardMarkup().add(
        types.InlineKeyboardButton(
            text="Головне меню",
            callback_data=ctypes.start_menu.new(
                button="back_to_menu",
            ),
        ),
    )
# -*- coding: utf-8 -*-

from aiogram import types
```

```
async def get_location():
    return types.ReplyKeyboardMarkup(
        resize_keyboard=True,
        row_width=1,
    ).add(
        types.KeyboardButton(
            text="Відправити локацію",
            request_location=True,
        )
    )


async def send_contact():
    return types.ReplyKeyboardMarkup(
        resize_keyboard=True,
        row_width=1,
    ).add(
        types.KeyboardButton(
            text="Надіслати контакт",
            request_contact=True,
        ),
    )
# -*- coding: utf-8 -*-

from loader import dp
from .logger_middleware import UpdateLoggerWiddleware


if __name__ == "middlewares":
    dp.middleware.setup(UpdateLoggerWiddleware())
# -*- coding: utf-8 -*-

from logzero import logger
from aiogram import types
from aiogram.dispatcher.middlewares import BaseMiddleware


class UpdateLoggerWiddleware(BaseMiddleware):
    async def on_process_update(self, update: types.Update, date: dict):
        logger.info(update)
# -*- coding: utf-8 -*-

from database import create_db
from aiogram import executor
from loader import dp, bot
from aiogram.types import BotCommand


async def set_all_default_commands(bot):
    await bot.set_my_commands(
        commands=[
            BotCommand("start", "Головне меню"),
        ]
    )


async def on_startup(dp):
    import middlewares, handlers
    await create_db(drop_all=True)
    await set_all_default_commands(bot)
```

```python
if __name__ == "__main__":
    executor.start_polling(dp, on_startup=on_startup)
# -*- coding: utf-8 -*-

from aiogram.dispatcher.filters.state import State, StatesGroup


class UploadPlace(StatesGroup):
    get_place_geo = State()
    get_place_photo = State()


class ModerationPlace(StatesGroup):
    get_place_name = State()
    get_type_pollution = State()


class CompletePlace(StatesGroup):
    get_place_photo = State()


class Checkout(StatesGroup):
    get_initials = State()
    get_contact = State()

# -*- coding: utf-8 -*-

from aiogram import types
from aiogram import Dispatcher, Bot
from aiogram.contrib.fsm_storage.memory import MemoryStorage

from data.config import BOT_TOKEN


bot = Bot(token=BOT_TOKEN, parse_mode=types.ParseMode.HTML)
storage = MemoryStorage()
dp = Dispatcher(bot=bot, storage=storage)
```