**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ**
**ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**

# КВАЛІФІКАЦІЙНА РОБОТА

першого (бакалаврського) рівня вищої освіти

на тему: **«Розробка чат-боту Postman для месенджера Telegram»**

Виконав: студент 2 курсу групи Іт-22сп

Спеціальності 126 «Інформаційні системи та технології»
(шифр і назва)
Шевців Роман Степанович
(Прізвище та ініціали)

Керівник: к.т.н., в.о. доцента Падюка Р.І.
(Прізвище та ініціали)
Рецензент: к.т.н., доцент Шарибура А.О.
(Прізвище та ініціали)

**ДУБЛЯНИ-2023**

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА
ІНФОРМАЦІЙНИХ ТЕХНЕОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Перший (бакалаврський) рівень вищої освіти
Спеціальність 126 «Інформаційні системи та технології»

«ЗАТВЕРДЖУЮ»

Завідувач кафедри_____

д.т.н., проф. А. М. Тригуба
«____» _____2023 р.

# ЗАВДАННЯ

на кваліфікаційну роботу студенту

Шевціву Роману Степановичу

1. Тема роботи: «Розробка чат-боту Postman для месенджера Telegram»

Керівник роботи Падюка Роман Іванович, в.о. доцента
затверджені наказом по університету від 30.12.2022 року № 453/к-с.

2. Строк подання студентом роботи 10.06.2021 р.
3. Вихідні дані до роботи: *вимоги до проектування чат-ботів; методика проектування інформаційних систем; технічне завдання на проектування чат-бота .*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)
*Вступ.*
*1. Аналіз предметної області.*
*2. Постановка задачі.*
*3. Проектування чат-боту Postman для месенджера Telegram .*
*4. Охорона праці.*
*Висновки та пропозиції.*
*Список використаної літератури.*
5. Перелік ілюстраційного матеріалу (з точним зазначенням обов'язкових креслень): *Огляд платформ з підтримки чат-ботів, аналіз існуючих технологій зі створення чат-ботів, етапи створення чат-бота і пов'язані із ними технології, структурна схема проектованої інформаційної системи, сценарії використання бота та загальний вигляд робочих вікон*

6. Консультанти з розділів:

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1, 2, 3, 4 | *Падюка Р.І., в.о. доцента кафедри ІТ* | | |
| 5 | *Городецький І.М., доцент кафедри управління проектами та безпеки виробництва* | | |

7. Дата видачі завдання 2 січня 2023 р.

Календарний план

| № з/п | Назва етапів дипломного проекту | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | *Написання першого розділу* | *2.01-02.02.23* | |
| 2 | *Виконання другого розділу та аркушів ілюстраційного матеріалу до нього* | *03-28.02.23* | |
| 3. | *Виконання третього розділу та аркушів ілюстраційного матеріалу до нього* | *01.03-30.04.23* | |
| 4. | *Написання розділу «Охорона праці»* | *01-15.05.23* | |
| 5. | *Завершення оформлення розрахунково-пояснювальної записки та аркушів ілюстраційного матеріалу* | *15-30.05.23* | |
| 6. | *Завершення роботи вцілому* | *01 - 10.06.23* | |

Студент _____ Шевців Р.С.
(підпис)

Керівник роботи _____Падюка Р.І.
(підпис)

УДК 004.9 : 631.1

Розробка чат-боту Postman для месенджера Telegram.

Шевців Р.С. Кафедра ІТ – Дубляни, Львівський НУП, 2023.

Кваліфікаційна робота: 59 с. текст. част., 34 рис., 2 табл., 10 арк. ілюстраційного матеріалу, 22 джерел.

Проведено аналіз ринку месенджерів, та встановлено, що сучасні месенджери мають безліч схожих функцій, але на особливу увагу заслуговує месенджер Telegram, як кросплатформенний додаток, яким можна користуватись на телефонах, планшетах та ПК.

Здійснено огляд сервісів для отримання тимчасових адрес електронної пошти, який дав можливість, серед усіх досліджених вибрати декілька варіантів, які дозволяють за рахунок використання власного API створювати сторонні додатки та боти на популярних месенджерах.

Описано особливості розробки чат-ботів та вибрано засоби розробки, серед яких мова програмування Python та фреймворк Python Telegram Bot для роботи з API месенджера Telegram.

Реалізовано клієнтську частину чат-боту. Детально описано процес його розробки та функціонування, а саме продемонстровано процес спілкування бота і користувача і пояснено роботу кожної важливої функції бота.

Розроблено заходи щодо охорони праці.

Ключові слова: чат-бот, електронна пошта, телеграм, месенджер, фреймворк

Key words: chatbot, email, telegram, messenger, framework

# ЗМІСТ

# ВСТУП

У сучасному світі, де електронна комунікація відіграє важливу роль, електронна пошта є одним з основних засобів зв'язку. Однак, часто зручність використання поштових скриньок у великій кількості ситуацій перекривається проблемою надмірного отримання небажаних листів або спаму.

З метою забезпечення приватності та захисту основної поштової скриньки виникає потреба у використанні тимчасових електронних адрес. Тимчасові адреси надають змогу отримувати листи без розголошення основної поштової скриньки та спаму.

Основною метою дипломної роботи є створення зручного інструменту для користувачів, який дозволить генерувати тимчасові адреси та захищати основну поштову скриньку від небажаних листів.

Для досягнення поставленої мети, робота передбачає реалізацію функціоналу для генерації випадкових адрес електронної пошти та основного функціоналу для контролю за новоствореною поштовою скринькою.

Для реалізації проекту використовується мова програмування Python та фреймворк python-telegram-bot для взаємодії з Telegram API. Крім того, використовуються зовнішні сервіси, такі як API Temp Mail для генерації тимчасових адрес електронної пошти та отримання листів.

Результати цієї дипломної роботи будуть корисними для користувачів, які шукають зручний і безпечний спосіб використання тимчасових адрес електронної пошти. Розроблений Telegram-бот «Postman» дозволить користувачам легко генерувати тимчасові адреси, отримувати та контролювати листи, забезпечуючи приватність та безпеку їх основних поштових скриньок.

# АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Аналіз ринку месенджерів

Месенджер - це додаток для смартфона або персонального комп'ютера, що дозволяє миттєво обмінюватися з друзями текстовими повідомленнями, телефонними дзвінками та навіть розмовляти з використанням відеозв'язку. [5]

Розвиток інтернет-сервісів для спілкування почався з чатів, потім месенджери, а далі соціальні мережі. Але на сьогодні месенджери знову стали найбільш популярні та перспективніші сервіси для онлайн спілкування. Це підтверджують дослідження компанії «DigData» у 2021 році: кількість користувачів месенджерів в Україні збільшилася на 9%. Водночас з цим середня кількість месенджерів, якими користується один користувачам, збільшилася за 2021 рік з 1,64 до 3 штук. месенджери використовуються переважно дорослими, так на частку осіб до 18 років припадає 8,3% від загальної кількості аудиторії, користувачі у віці від 18 до 25 років складають 10,1%, від 25 до 35 років - 31,9%, від 35 до 45 років - 27,1%. При цьому 12,9% користувачів месенджерів знаходяться у віці від 45 до 55 років, а 7,5% - від 55 до 64 років. s лише 2,6% аудиторії месенджерів знаходиться в віці старше 64 років. Причиною повторної хвилі популярності месенджерів стали зміни в області мобільного Інтернету, швидкість набагато збільшилась, ціни стали нижчими ніж раніше та широке використання та поширення смартфонів [2].

Сучасні месенджери мають безліч схожих функцій, таких як надсилання повідомлень, аудіо- та відео спілкування, поширення медіа файлів. В наш час існує велика різноманітність месенджерів: Viber, WhatsApp, Facebook, Google Hangouts, але на особливу увагу заслуговує месенджер Telegram [1].

Telegram - це кросплатформенний додаток, яким можна користуватись на телефонах, планшетах та ПК. Він розроблений мовою програмування C ++, тому дозволяє поширювати повідомлення та файли більшості форматів. Telegram використовує спеціально розроблену серверну частину з закритим

кодом, які працюють на серверах Німеччини і США. [11]

Telegram месенджер має ряд переваг:

- приватність – всі чати шифруються, а повідомлення видаляються через зазначений час;
- швидкість – швидкість доставки повідомлень набагато вище, ніж в аналогів;
- розподіл – Telegram сервера розташовані по всьому світу, що підвищує працездатність та відмовостійкість;
- відкритість – використання відкритого протоколу MTProto і API, безкоштовних для користувачів;
- відсутність підписок і зайвої реклами;
- немає обмежень на розмір вкладених файлів та повідомлень.

З мінусів можна відзначити, що месенджер не має функції відеодзвінків.

Засновниками Telegram є брати Павло та Микола Дурови. Павло є фінансова та ідеологічна опора проекту, а Микола працює з технічними аспектами.

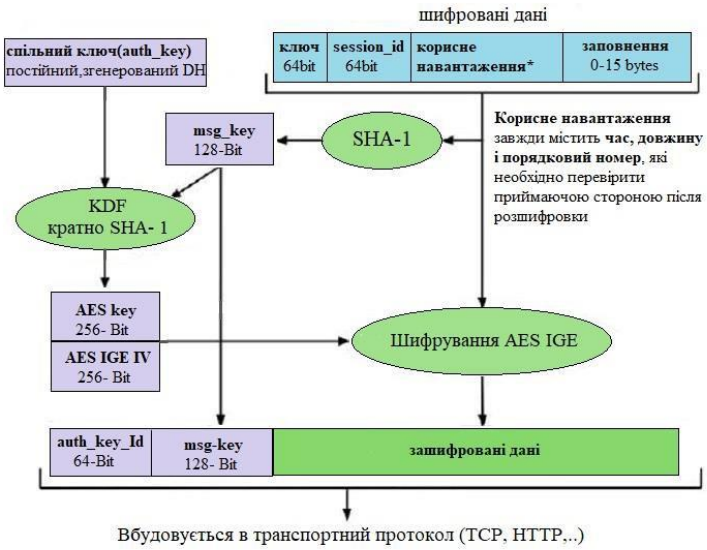


Рисунок 1.1. – Протокол шифрування MTProto [7].

Особливість популярного месенджера це секретні чати та спеціально

розроблений протокол шифрування MTProto (рисунок 1.1).

Секретні чати потрібні для тих користувачів, які бажають забезпечити себе максимально безпечним спілкуванням. Крім основного протоколу всі повідомлення проходять шифрування ключами за принципом від пристрою до пристрою. Це створено для того, щоб ніхто не зміг перехопити та не розшифрувати повідомлення, навіть співробітники Telegram. Повідомлення, фото та відео з секретних чатів пересилати та зберігати не можна і вони не зберігаються на серверах додатку. Але секретний чат має два мінуси:

- неможливо відкрити чат на пристрої, який не брав участі в його створенні;
- не захищеність від скриншоту інтерфейсу месенджера де відритий секретний чат.

Месенджер Telegram можна використовувати майже на всіх популярних операційних систем, наприклад, Android, iOS, Windows, Ubuntu та інші.

У 2018 році Telegram активно працював над розробкою своєї блокчейн-платформи TON(Telegram Open Network) і криптовалютою Gram на її основі. Але у 2020 році Павло Дуров заявив, про закриття проекту.

## 1.2 Загальні відомості про чат-боти та їх функції

Чат-бот - це програма, яка була розроблена на основі технологій машинного навчання та нейромереж. Вона створюється людиною для людей та навчається під певне коло цілей. Чат-бот імітує розмову з людиною в Інтернеті, саме тому даний сервіс найкраще зарекомендував себе в месенджерах. [4]

В останній час в додатках набирають популярність такі сервіси як чат-боти. Термін «чат-бот» був придуманий та створений Майклом Молдін в 1994 році для опису розмовних програм.

Пізнавши сучасний стан використання чат-ботів в додатках, можна прийти до висновку, що це є універсальний засіб, здатний до вирішення безліч різних завдань. За допомогою чат-ботів можна спілкуватися надавати медичну

консультацію і навіть робити замовлення товарів та послуг.

В незалежності від операційної системи, чат-бот – це прикладна програма, яка, отримує інформацію від користувача та формує логічно обґрунтовані відповіді. На сьогодні є дуже величезна різноманітність ботів помічників. Деякі з можливих варіантів представлені нижче:

- ігрові чат-боти (квести, загадки);
- рекламні чат-боти (SAT, Pepsi Cola);
- новинні чат-боти (Укрінформ, Pravda);
- чат-боти для замовлень, магазинів, сервісів (таксі «Uber», піцерія «Челентано», бронювання поїздів і літаків «Tickets.ua»);
- чат-боти для консультації та підтримки клієнтів (банк «Моно») та інші.

В останній час великі можливості мають інформаційні технології, пов'язані з ботами. Вони дуже міцно ввійшли в наше життя, що мають застосунок у всіх сферах нашої діяльності та з кожним днем їх роль стає все більшою. Тому, що основну частину свого часу люди проводять за комп’ютерами та смартфонами в email-клієнтах і месенджерах.

Боти - це програми, які використовуються для різних завдань користувача, що перебуває в месенджері. Бот в Telegram месенджеру виглядає як звичайний чат, але спілкування відбувається з програмою, а не з людиною. [5] За допомогою такого бота можна зробити замовлення машини, якщо це - бот таксі, або присилати свіжі новини, погоду, або замовити доставку їжі додому, якщо це - ресторанний бот.

Зовсім недавно чат-боти здобули не малу популярність, перетворившись з розваг в більш серйозну річ, такі боти в основному, стали використовуватися для вирішення серйозних завдань. В час інформаційних технологій - це нормальне явище, а тим більше для мережі Інтернет, адже сучасне суспільство перейшло на «нове» і ділове, і неділові спілкування. Зараз чат-боти це програми для вирішення бізнес-завдань. Чат-бот це додаток, який веде діалог з людьми, вибираючи логічні відповіді з бази даних, після питання де пообідати ви тут же отримуєте миттєву відповідь. Крім цього, чат-боти виконують багато корисних

операцій: виконання рутинної роботи, пошук інформації, об'єднання даних та роботою з клієнтами. Він використовується як віртуальний співрозмовник має базу знань та відповідей, за допомогою цього може дати відповідь на різні питань користувача. Для того щоб чат-бот міг дати логічну відповідь використовують ключові слова, збіг фрази та контексту. Постійно існують прості та легкі по виконання справи, на які не завжди є час, та бажання. На таку допомогу можна використати бота. Для пошуку та збору інформації можна використовувати чат-ботів. Наприклад, коли відбувається який-небудь захід чат-боти можуть повідомляти всім учасникам новини та довідкову інформацію.

## 1.3. Огляд веб-сервісів для отримання тимчасових адрес електронної пошти.

### 10MINUTEMAIL.NET

Цей безкоштовний сервіс дозволяє створити пошту швидко та легко. Все, що потрібно зробити, – натиснути на посилання «Отримати нову поштову скриньку». Після цього у тимчасовій поштовій скриньці з'явиться вітальний лист. За замовчуванням адреса надається на 10 хвилин (рис. 1.2)

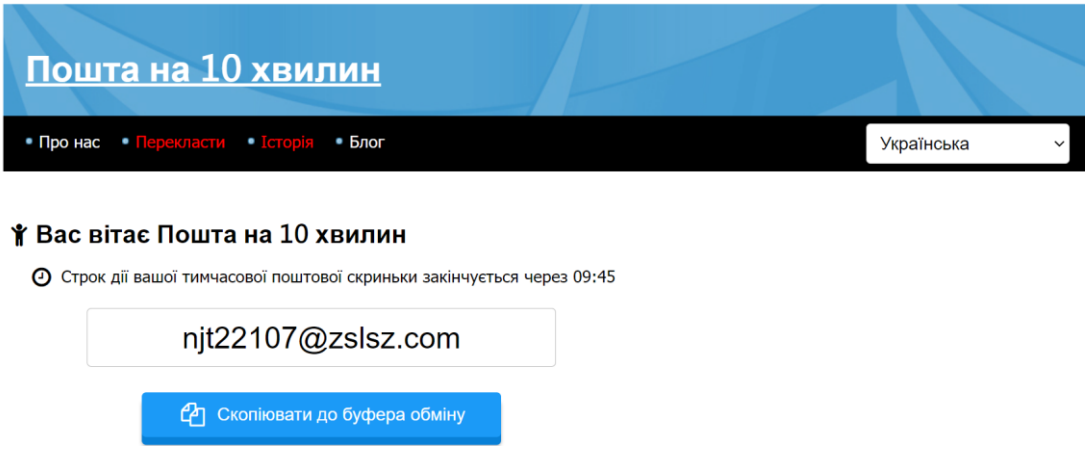


Рисунок 1.2 Сервіс тимчасової пошти 10minutemail.net

Якщо цього періоду вам недостатньо, можна продовжити час використання до 10 хвилин. Зробити це можна за допомогою посилання "Дайте мені ще 10 хвилин!". Скриньку можна відновити протягом години, перш ніж вона буде видалена остаточно.

## TEMPMAIL

При переході на сайт і проходженні капчі вам автоматично надається тимчасова email-адреса. Інтерфейс сервісу дуже простий: є кнопки для перегляду QR-коду, згенерованого для доступу до пошти, копіювання адреси в буфер обміну, оновлення списку листів та видалення поштової скриньки (рис 1..

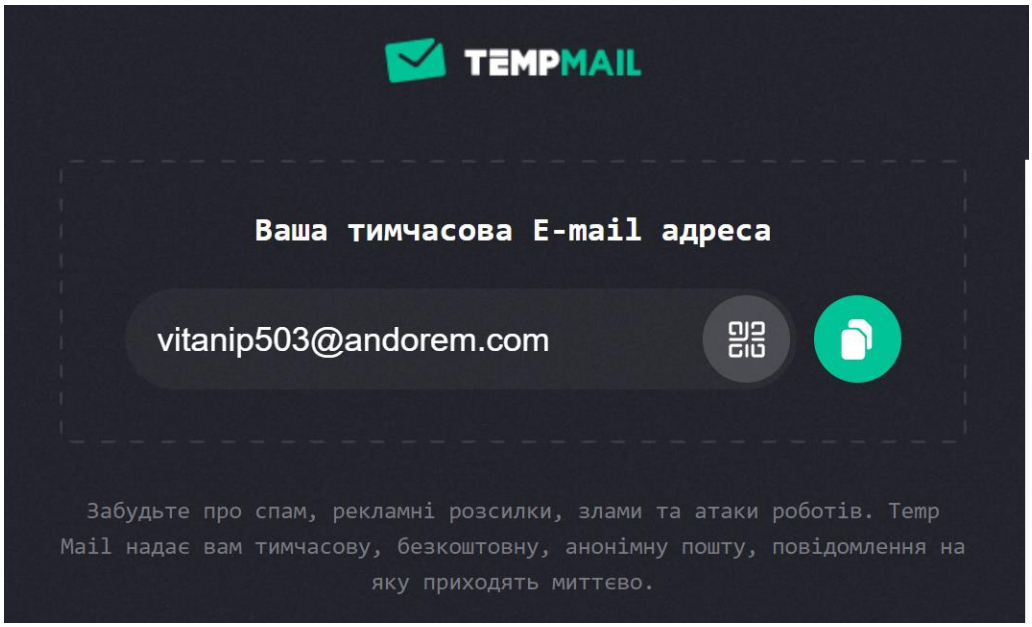


Рисунок 1.3 Сервіс тимчасової пошти Tempmail

На тимчасову пошту TempMail можна отримувати листи з вкладеннями. Для перегляду та завантаження вкладень потрібно відкрити лист і натиснути на посилання «Файли». Кнопка «Властивості» дозволяє зберігати листи. Надсилання листів з тимчасової адреси не передбачено.

Для роботи з сервісом з мобільних пристроїв можна завантажити програми App Store або Google Play. Також є платна версія Premium, яка дозволяє створювати до 10 адрес одночасно, підключати персональний домен і використовувати приватну адресу з повним володінням.

## MOHMAL

Ще один спосіб швидко створити пошту – скористатися сервісом MohMal, який надає електронну пошту на 45 хвилин. Як і під час роботи з попередніми сервісами, реєстрація не потрібна (рис. 1.4).

При створенні тимчасової адреси можна згенерувати випадкове ім'я або вказати його самостійно (у цьому випадку ви самі вигадуєте ім'я, що легко запам'ятовується, а домен вам запропонують один з шести на вибір).

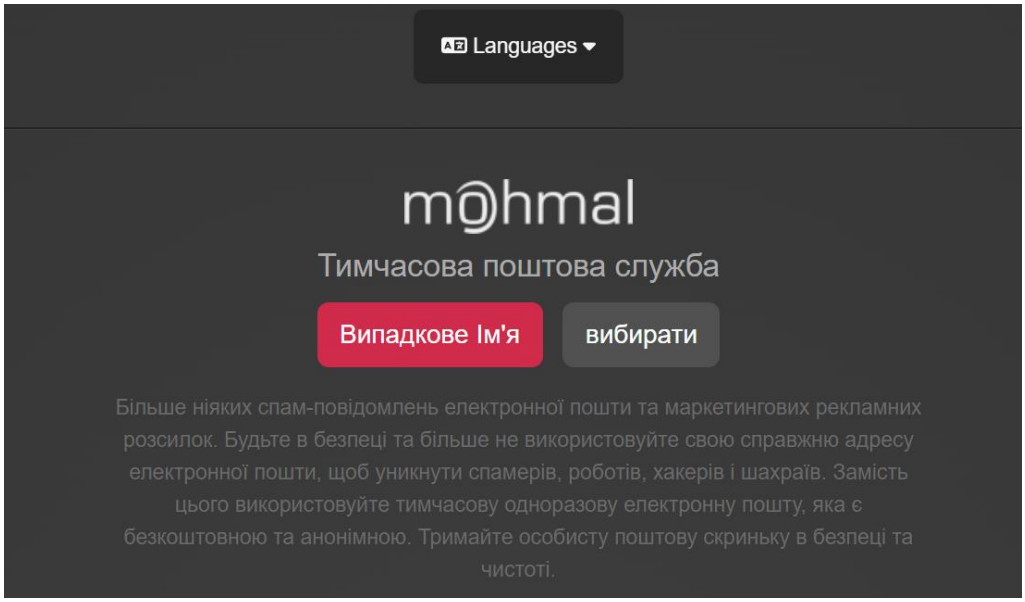


Рисунок 1.4 Сервіс тимчасової пошти MohMal

Після створення поштової скриньки буде запущено наочний таймер, який не дасть вам забути про час. Продовжити термін дії адреси можна за допомогою кнопки «Відновити»: час знову збільшиться до 45 хвилин. Продовжувати таким чином час можна необмежену кількість разів.

**TEMPAIL**

Даний сервіс надає анонімну безкоштовну адресу на строк до 7 днів. Кнопки на панелі праворуч дають можливість скопіювати згенерований email у буфер обміну, оновити папку «Вхідні», отримати QR-код та посилання для доступу до вашої поштової скриньки з будь-якого пристрою, а також видалити свою тимчасову адресу (рис.1.5).

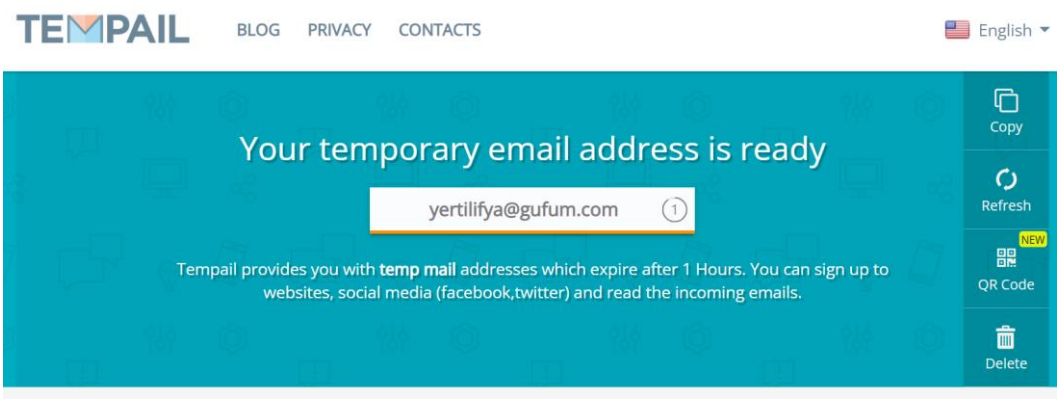


Рисунок 1.5 Сервіс тимчасової пошти Temppail

Особливості сервісу: оновлення пошти кожні 10 секунд, підтримка роботи декількох адрес одночасно. З можливостей роботи з листами є функції «Позначити як непрочитане» та «Видалити». Відповідати на отримані листи або зберігати їх на комп'ютер не можна, вкладення також не підтримуються.

Не всі з цих сервісів мають свої власні чат-боти на порулярних платформах. А надає декілька варіантів API для розробки свого власного бота лише сервіс TEMPMAIL, тому його ми і будемо використовувати для розробки бота в рамках цієї дипломної роботи.

## 1.4 Аналіз аналогів чат-ботів для створення тимчасових адрес електронної пошти на платформі Telegram.

Варто також проаналізувати функціонал аналогічних діючих чат-ботів для створення тимчасових адрес електронної пошти на вибраній нами платформі для розробки власного чат-бота. Бот Fake mail як і інші аналоги, надає можливість створювати тимчасові адреси електронної пошти для отримання листів без реєстрації (рис 1.6).

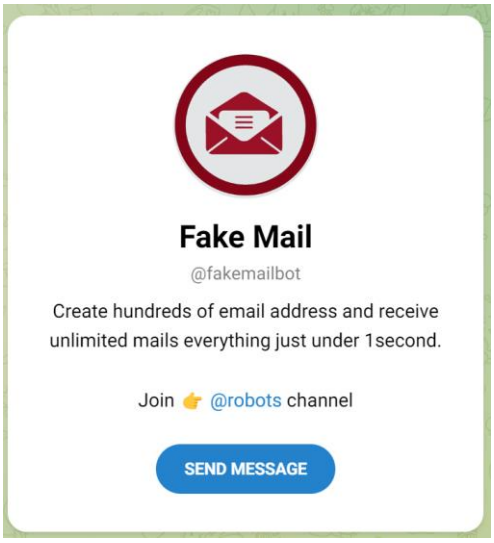


Рисунок 1.6 Бот Fake mail

Але серед особливостей можна виділити можливість створення власної назви тимчасої адреси електронної пошти і керування списком блокувань.

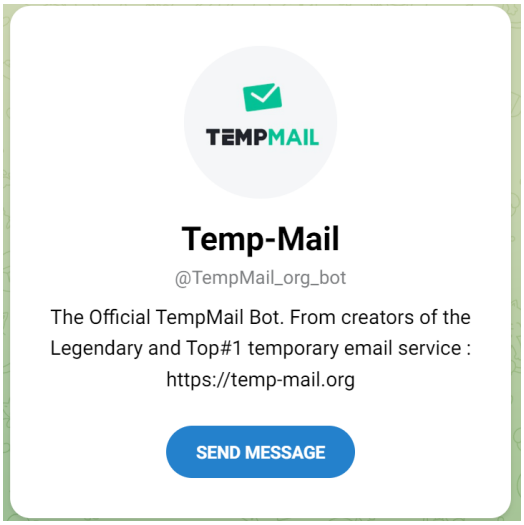


Рисунок 1.7 Бот TempMail

Офіційний бот TempMail (рис. 1.7) від творців легендарної та першої служби тимчасової електронної пошти: https://temp-mail.org дозволяє лише створювати адреси електронної пошти, а при отриманні нового листа генерує посилання для його перегляду на офіційні веб-сторінці у браузері.


Рисунок 1.8 Бот DropMail.me

Бот DropMail.me (рис. 1.8) крім створення тимчасових адрес електронної пошти дозволяє також переглядати список активних адрес.

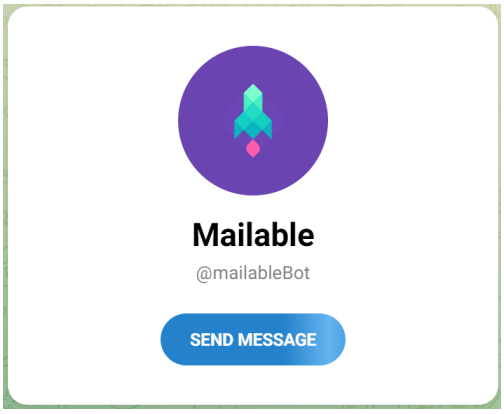


Рисунок 1.9 Бот Mailable

Найфункціональнішим серед предствлених ботів, на нашу думку, є бот Mailable (рис. 1.9). Він дозволяє, серед іншого, генерувати випадковий лист, установити пошту на ваш ідентифікатор електронної пошти, отримати список листів, заблокувати лист та отримати список поточних блокувань.

Не зважаючи на доволі різноманітний асортимент активних чат-ботів зі схожою функціональністю, на нашу думку є потреба у подальшому удосконаленні таких ботів, додаванням нових команд, які дозволять більш продуктивно використовувати тимчасові електронні скриньки. Тому створення нашого бота з розширеним функціоналом є доволі актуальною задачею.

# 2. ПОСТАНОВКА ЗАДАЧІ ТА ПЛАНУВАННЯ ПРОЕКТУ

## 2.1. Мета та задачі проекту

Метою розробки цього бота є надання користувачам можливості генерувати випадкові тимчасові адреси електронної пошти і отримувати доступ до листів, надісланих на ці адреси. Такий бот може бути корисним у випадках, коли користувачам потрібно створити тимчасову поштову скриньку для реєстрації на різних сайтах, отримання підтверджень або забезпечення приватності.

Цей бот надає зручний спосіб створення та використання тимчасових адрес електронної пошти через Telegram, спрощуючи процес отримання та керування листами. Користувачі можуть швидко генерувати адреси, перевіряти статус листів та отримувати необхідну інформацію, зберігаючи при цьому свою основну електронну пошту приватною та безпечною.

Для виконання цього проекту потрібно виконати наступні завдання:

- Проаналізувати предметну область, а саме ринок месенджерів, а також здійснити огляд веб сервісів для отримання тимчасових адрес електронної пошти;
- Здійснити загальний огляд чат-ботів та їх функцій. Проаналізувати аналоги чат-ботів для створення тимчасових адрес електронної пошти на платформі Telegram;
- Здійснити вибір засобів реалізації проекту та середовища розробки;
- Зареєструвати бот на платформі Telegram та вибрати платформу керування API для роботи з сервісом тимчасових поштових скриньок;
- Реалізувати клієнтську частину чат-бота та відлагодити її роботу.

## 2.2 Вибір засобів реалізації проекту

**Мова програмування.** Python – скриптова та високорівнева мова програмування, що фокусується на читабельності коду. У наш час Python є широко поширеною мовою, яка використовується в багатьох областях, наприклад, для розробки прикладного ПЗ, web-додатків та в якості вбудованого скрипта в багатьох іграх. Мова націлена на покращення продуктивності праці, вона допомагає розробникам робити кодування за кілька кроків у порівнянні з Java або C++ [18].

Python - це динамічна програма програмування високого рівня, інтерпретована та загальноприйнята, що фокусується на читанні коду. Синтаксис в Python допомагає програмістам робити кодування за кілька кроків порівняно з Java або C ++. Вона була заснована в 1990 році розробником Гвідо Ван Россумом, при її використані програмувати легко та цікаво. [22]

Мова Python є широко використовувана у великих компаніях тому, що вона підтримує різні парадигми програмування: об’єктно орієнтовану, функціональну, структурну, імперативну та аспектно орієнтовану. Python має велику стандартну бібліотеку яка є комплексна та має автоматизоване керування пам'яттю та динамічні функції.

Більшість великих компаній які спеціалізуються на розробці програмного забезпечення, віддають перевагу мові Python тому, що вона є універсальною та через меншу кількість програмних кодів. Майже 15% розробників користуються нею в таких операційних системах, як UNIX, Linux, Windows та Mac OS. Розробники у великих компаніях користуються Python, оскільки вона має в розробці програмного забезпечення з характерні функції, такі як [18]:

- інтерактивність;
- інтерпретовність;
- модульність;
- динамічність;

- об’єктно-орієнтованість;
- портативність;
- високорівневість;
- можливість розширення на C ++ та C.

До переваг мови програмування Python можна віднести відкриту розробку коду, досить легка у навчанні, особливо на початковому рівні, особливості синтаксису допомагають програмісту писати код швидше, легко читається та безліч корисних бібліотек та розширень мови [18].

Наступна значна перевага мови Python це велика чисельність модулів, які можна імпортувати, щоб забезпечити багато різних додаткових можливостей. Ці модулі пишуться на мові C, на самому Python і можуть бути використані більш досвідченими програмістами. До приклада можна навести такі модулі:

- NumPy (Numerical Python) – дає розробнику набагато ширші можливості в математиці, такі як робота та маніпуляція з векторами та матрицями;
- Tkinter – він надає можливість розробляти додатки з додаванням графічного інтерфейсу користувача (GUI);
- OpenGL (Open Graphics Library) – є дуже великою бібліотекою графічного моделювання двовимірних та тривимірних об'єктів OpenGL.

Єдиним мінусом, про який говорив сам розробник Python, - це низька швидкість виконання Python програми порівняно з іншими. Але це не грає велику роль якщо порівнювати з перевагами мови при кодуванні програм не дуже критичних до швидкості виконання [22].

**Модуль Telegram Bot API.** Bot API це створений HTTP-інтерфейс для роботи з ботами в Telegram месенджері. Кожний бот - це спеціальний акаунт, розроблений для автоматизованого оброблення та відправлення повідомлень.

Існує два різних за принципом способи отримати оновлень від бота:[10]

- long pulling - додаток який автоматично зв’язується з сервером Telegram на наявність нових оновлень для бота. Встановлена швидкість за замовчуванням 100мс;

- webhook - сервера месенджера які автоматично сповіщають додаток на сервері на наявність нових оновлень. Такі оновлення будуть збережені на сервері, до того часу поки вони не опрацюються, але не довше однієї доби. В будь якому разі, незалежно від способу отримання оновлень, буде відправлено об'єкт Update, серіалізовані в JSON.

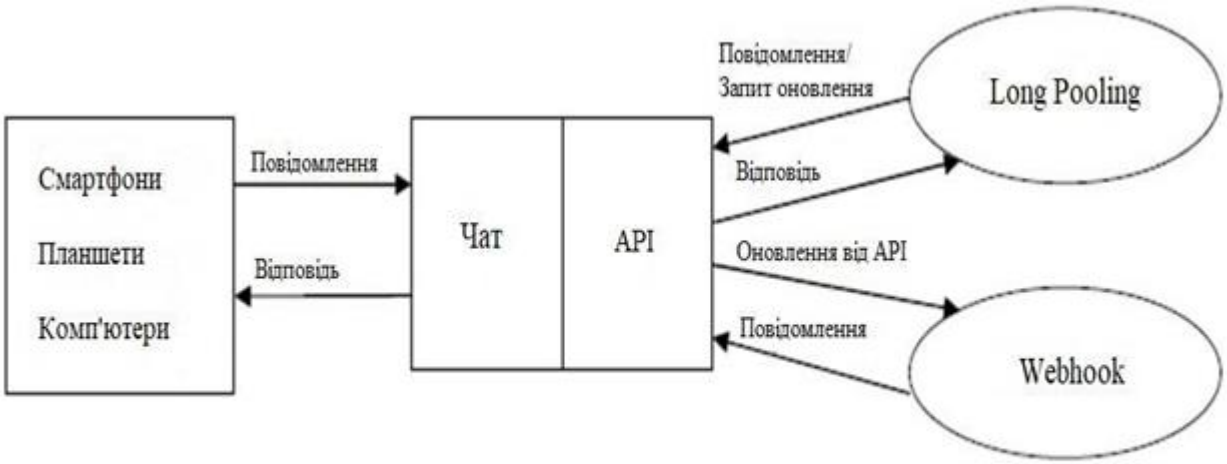


Рисунок 2.1. - Принцип роботи чат-бота на платформі Telegram [10].

Всі запити які будуть надходити до Telegram Bot API мають здійснюватися через HTTPS в наступному вигляді: https://api.telegram.org/bot<token>/НАЗВА_МЕТОДУ.

Принцип роботи взаємодії чат-бота і користувача зображений на рисунку 2.1.

Для отримання унікального ключа потрібно написати спеціальному боту @BotFather. Зразки доступних для API методів можна побачити нижче:

- getUpdates - цей метод використовується для отримання оновлень за технологією long polling;
- setWebhook - метод який прив'язує до бота url домен, де знаходиться відкритий і запущений бот;
- sendMessage - метод який робить відправку текстового повідомлення в клієнт Telegram;

- sendLocation - метод відправляє повідомлення з координатами в клієнт Telegram;
- getFile - метод який робить повернення загруженого файлу по його імені. Допускаються також POST і GET запити. Існує 4 способи для відправки параметрів в Bot API [9]:
- Запит в URL
- application / x-www-form-urlencoded
- application / json (не прийнятний для завантаження файлів)
- multipart / form-data (для завантаження файлів)

**Модуль Requests.** Це бібліотека Python, яка виконує HTTP-запити (HyperText Transfer Protocol). Починаючи від передачі параметрів в URL-адресах до відправки користувацьких заголовків і перевірки SSL [10].

**Архітектура REST API.** REST – це стиль архітектури програмного забезпечення, який використовується для побудови розподілених масштабованих веб-сервісів, які використовують HTTP запити [12].

REST (Representational State Transfer) –стиль взаємодії компонентів розподіленого додатку в мережі. REST має узгоджений набір обмежень, що враховуються при проектуванні розподіленої гіпермедіа-системи. У визначених випадках (інтернет-магазини, пошукові системи) це спричинює підвищення продуктивності та змінення архітектури, а саме її спрощення. Компоненти в REST своєю взаємодією нагадують взаємодію клієнта і сервера в Інтернет.

Виклик віддаленої процедури в мережі Інтернет може представляти собою звичайний HTTP-запит (зазвичай «GET» або «POST»; такий запит називають «REST-запит»), а всі необхідні для передачі дані передаються в якості параметрів запиту.

Для веб-служб, які були побудовані з урахуванням REST, застосовують так званий термін «RESTful».

На відміну від веб-сервісів на основі SOAP, не існує єдиного офіційного стандарту для терміну RESTful веб-API. Оскільки REST є архітектурним стилем, в той час як SOAP є протоколом. REST не є стандартом сам по собі і не

зважаючи на це, більшість RESTful-реалізацій використовують відомі стандарти, такі як: HTTP, URL, JSON і XML.

**HTTP-запити.** Протокол Передачі Гіпертексту (HTTP) – це один із відомх протоколів стеку TCP / IP (Transmission Control Protocol / Internet Protocol), з самого початку він був розроблений для розміщення і отримання HTML (HyperText Markup Language) сторінок і на даний момент використовується для розподілених інформаційних систем. HTTP широко використовується у Інтернет, а саме для передачі інформації і являє собою найбільш використовуваний прикладний протокол [17].

HTTP це протокол, який визначається типом запит / відповідь. Коли клієнт або веб браузер, відправляє повідомлення з запитом на сервер, HTTP протокол самостійно визначає типи надісланих серверу повідомлень, які використовуються клієнтом для запиту до веб сторінки, а також типи повідомлень, які сервер використовує для відповіді. Три найбільш популярні типи повідомлень це: GET, POST і PUT. На рисунку 2.2 наведено приклад GET-запиту [17].

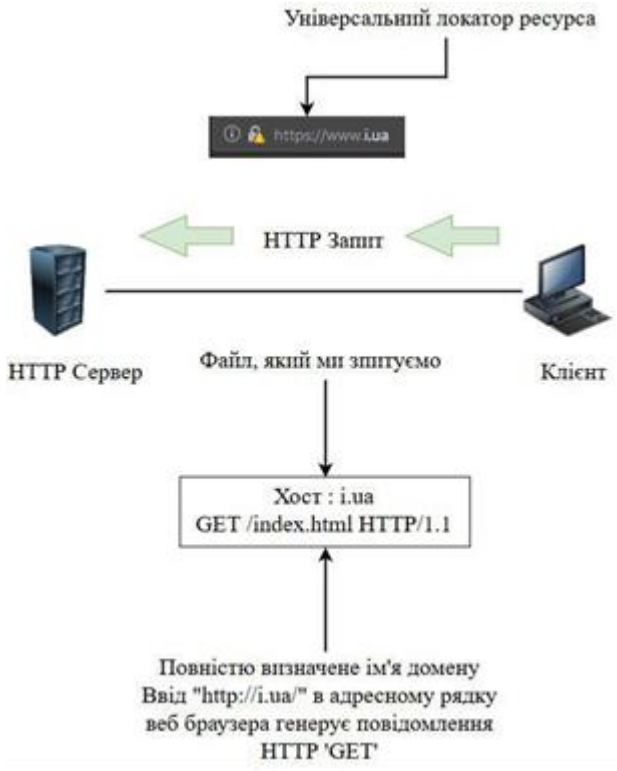


Рисунок 2.2 – Приклад HTTP протоколу з використанням GET

Для того, щоб відправити повідомлення, які завантажують дані на веб сервер використовуються POST і PUT. Наприклад, коли користувач вводить дані в форму, яка знаходиться на веб сторінці, POST додає ці дані в повідомлення і вони посилаються на сервер. PUT завантажує ці ресурси або інший контент на веб сервер.

HTTP протокол є надзвичайно гнучким, але це не робить його безпечним протоколом. Повідомлення POST відправляють дані на сервер у вигляді звичайного тексту, який може бути з легкістю перехоплений. Аналогічно, відповіді, які надходять від сервера, зазвичай також не зашифровані.

Для того, щоб створити безпечну комунікацію через Інтернет використовується безпечний HTTP протокол (HTTPS), він дозволяє отримувати доступ або публікувати інформацію на веб сервері. HTTPS дає змогу використовувати аутентифікацію і шифрування, щоб захистити дані від перехоплення, коли вони переміщуються між клієнтом і сервером. HTTPS визначає деякі свої правила для проходження даних між прикладним і транспортним рівнями.

**Формат JSON.** (JavaScript Object Notation) – це формат для обміну даними, комфортний для зберігання, читання та написання як людиною, так і комп'ютером. Він був заснований на підмножині відомої мови програмування JavaScript та визначений в стандарті ECMA-262 3rd Edition - December 1999. JSON – представляє собою майже звичайний текстовий формат, повністю незалежний від мови реалізації, але він використовує деякі правила, які знайомі програмістам C-подібних мов, таких як C, C ++, C # , Java, JavaScript, Perl, Python і багатьох інших. Саме ці властивості роблять JSON ідеальною мовою, яку можна використовувати для обміну даними [15].

JSON з самого початку був заснований на двох структурах даних:

а) колекція пар ключ / значення. Ця концепція, у різних мовах може бути реалізована як об'єкт, запис, структура, словник, хеш, іменований список або асоціативний масив;

б) упорядкований список значень. Більшість мов програмування реалізують це як масив, вектор, список або послідовність. Це універсальні структури даних. Майже всі сучасні мови програмування мають змогу підтримувати їх в будь-якій формі. Досить легко припустити, що формат даних, який зберігаються в JSON, незалежний від мови програмування та повинен бути заснований на структурах вказаних вище.

В нотації JSON це виглядає так:

а) об'єкт – це невпорядкований набір пар ключ/значення. Об'єкт починається з відкритої фігурної дужки і закінчується закритою фігурною дужкою. Кожне ім'я в об’єкті супроводжується двокрапкою, а пари ключ/значення розділяються комою [15] (рис. 2.3);

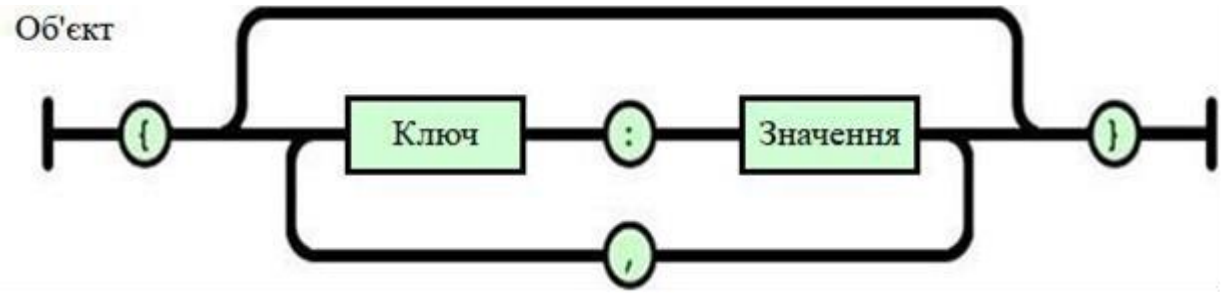


Рисунок 2.3 – Схематичне представлення об'єкта JSON

б) масив – це впорядкована колекція, яка може містити різні значення. Масив починається з квадратної дужки і закінчується цією дужкою. Значення в масиві розділяються комою (рисунок 2.4)[15];

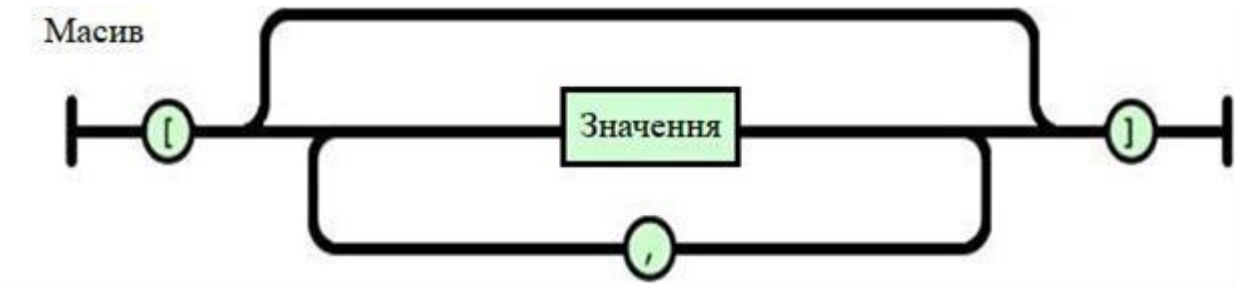


Рисунок 2.4 - Схематичне представлення масиву JSON

в) значення – може бути числом, рядком в подвійних лапках, true, false, null, об'єктом або масивом. Ці структури можуть бути вкладеними (рис. 2.5);

г) рядок – це так званий набір символів в кодуванні Unicode, укладений в подвійні лапки, або символ, який використовує зворотний слеш в якості символу екранування. Символ представляється як один символьний рядок. Схожий синтаксис використовується в мовах програмування C і Java (рис. 2.6);

Число в JSON передається так, як в C або Java, крім того, що використовується тільки десяткова система числення [15] (рис. 2.7).


Рисунок 2.5 – Схематичне представлення значення в JSON

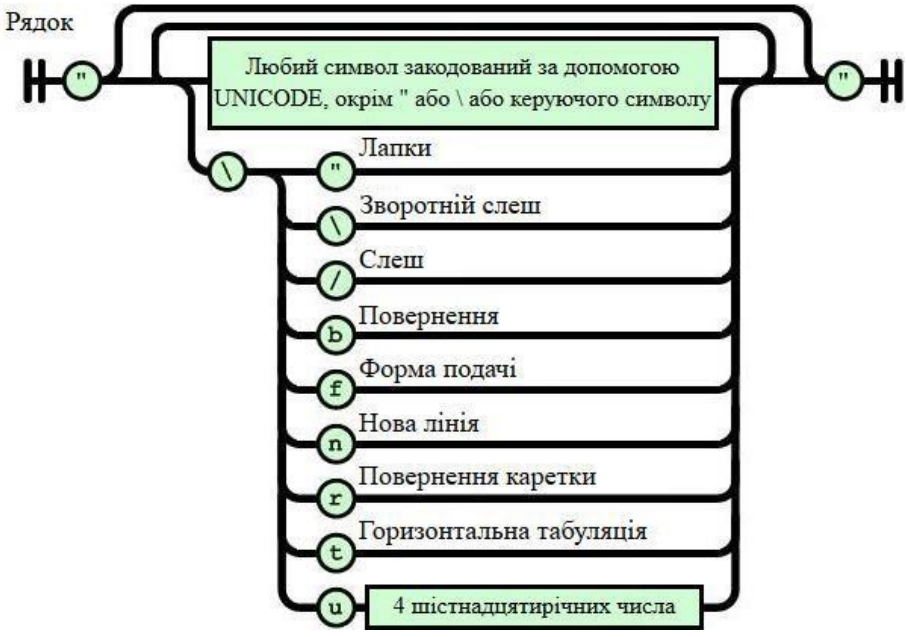


Рисунок 2.6 - Схематичне представлення рядки в JS

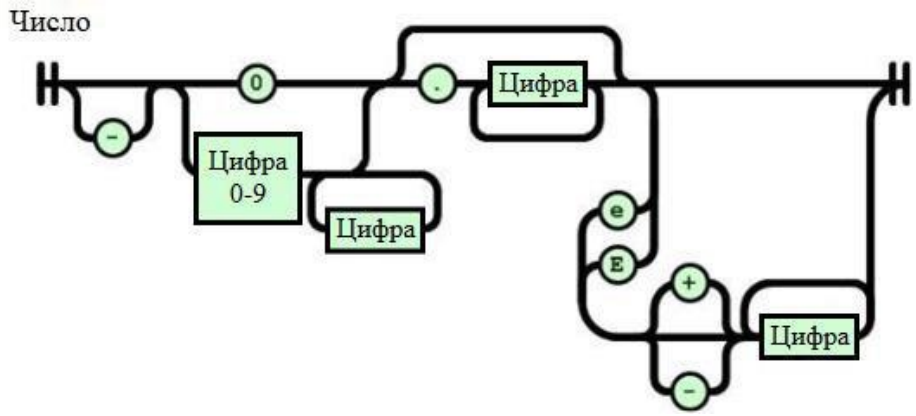


Рисунок 2.7 – Схематичне представлення числа в JSON

**Heroku.** Це хмарна PaaS-платформа, яка є сумісна с декількома мовами програмування. З 2010 року є дочірньою компанією Salesforce. Вона є однією з перших хмарних платформ яка з'явилася в липні 2008 року.[14] Спочатку Heroku мала підтримку тільки однієї мови програмування Ruby, але на сьогодні перелік мов які вона підтримує містить в собі Python, Java, Clojure, Node.js, PHP та інші. Сервера Heroku в основному використовуються на операційних системах Debian або Linux.

Платформа Heroku була заснована в 2008 році Джеймсом Лінденбаун, Адамом Віггінс та Оріон Генрі для роботи з проектами, створених на Rack. 10 січня 2010 року вона була куплена компанією Salesforc які зробили її своєю дочірньою компанією. У 2011 році Ю. Мацумото який був батьком мови програмування Ruby, перейшов до компанії в ролі інженера. Цього ж місяця було впроваджено підтримку Node.js та Clojure. Також платформа має підтримку таких баз даних, як Cloudant, Membase, Mongo та Redis [14].

Додатки, які працюють на цій платформі, використовують DNS - сервер (програми мають доменне ім'я типу «Ім’я_сервера.herokuapp.com»). Для таких програм виділяють малу частину віртуальних процесів, названі «dunos». Такі додатки розподіляються на спеціально розробленій віртуальній сітці, вона складається з декількох серверів. Платформа Heroku має власну систему контролю за оновленням версій [14].

### 2.3 Огляд можливостей середовища розробки

Visual Studio Code - це простий у використанні, але потужний редактор вихідного коду, який підтримує операційні системи такі як Windows, MacOS і Linux.[13]

IDE має підтримку великої кількості популярних мов програмування. Середовища виконання і розширення підтримувані IDE представлені нижче:

- JavaScript;
- TypeScript;
- Node.js;
- C ++;
- C #;
- Java;
- Python;
- PHP;
- Go;
- NET;
- Unity.

Для зручного використання програмних продуктів VS Code містить в собі вбудований відладчик, інструменти для роботи з Git-репозиторіями, підсвічування синтаксису, засоби для рефакторинга і IntelliSence (технологія автодоповнення, яка пропонує команду за першими літерами).

IDE VS Code була представлена 28 травня 2015 року компанією Microsoft в рамках конференції Blink. 19 жовтня 2015 року VS Code був відправлений у використання під ліцензію Масачусетського технологічного інституту.

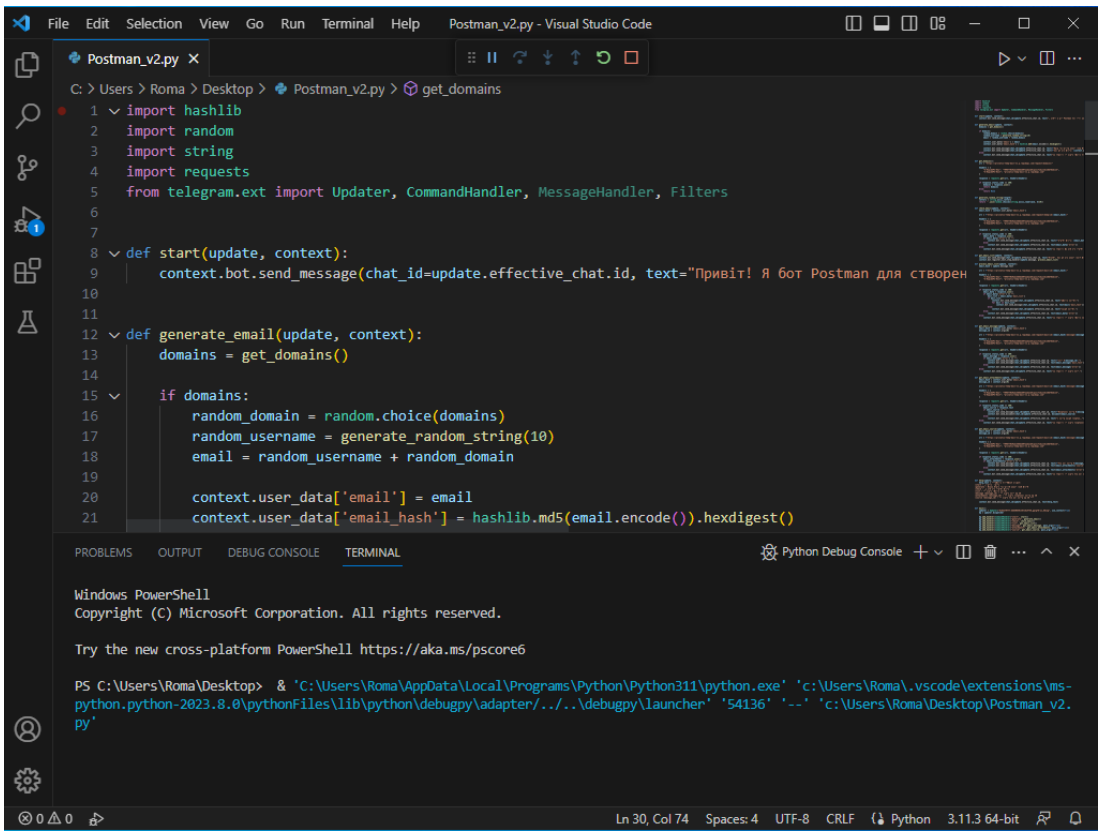


Рисунок 2.8. - Інтерфейс IDE Visual Studio Code.

Visual Studio Code був заснований на Electron – фреймворк який дозволяє використовуючи Python.py розробляти настільні додатки які працюють на движку Blink. Не дивлячись на це, що редактор створений на Electron, він не використовує редактор Atom. На заміну був реалізований вебредактор Monaco, розроблений для Visual Studio Online. Інтерфейс IDE Visual Studio Code представлений на рисунку 2.8.

Перед початком написання коду були імпортовані такі бібліотеки:

- Requests – ця бібліотека є однією з найпопулярніших бібліотек для здійснення HTTP-запитів у Python. Вона надає простий та зручний інтерфейс для взаємодії з веб-серверами і отримання відповідей у різних форматах, таких як JSON, XML, HTML та інші.

- Hashlib – призначення цієї бібліотеки у Python полягає в наданні функціоналу для обчислення хеш-сум та криптографічних хеш-функцій. Вона дозволяє використовувати різні алгоритми хешування, такі як MD5, SHA-1, SHA-256 та інші.

- Random – призначення бібліотеки полягає в наданні функціоналу для генерації випадкових значень. Вона містить різноманітні функції, які дозволяють генерувати випадкові числа, вибирати випадкові елементи зі списків або послідовностей, а також перемішувати дані. Ця бібліотека корисна для створення випадкових даних, генерації випадкових вибірок і виконання інших завдань, де потрібна випадковість.
- String – бібліотека у Python, яка надає набір корисних констант і функцій для роботи з рядками. Вона містить стандартні набори символів, такі як літери верхнього та нижнього регістру, цифри, пунктуаційні знаки тощо. Основне призначення string полягає у спрощенні маніпуляцій з рядками, таких як створення, форматування та обробка текстових даних.
- Updater з пакету python-telegram-bot є основною складовою для створення та керування ботами Telegram з використанням Python. Вона надає зручний інтерфейс для взаємодії з Telegram Bot API та обробки вхідних повідомлень, команд та подій. Updater також надає можливість налаштовувати обробники повідомлень за допомогою CommandHandler, MessageHandler та інших фільтрів, що дозволяє розподіляти обробку повідомлень на основі команд або умов.

## 2.4. Вибір платформи керування API для роботи з сервісом тимчасових поштових скриньок

Temp Mail API можна використовувати кількома способами, незалежно від того, чи це процес тестування вручну, виклик API або використання інфраструктури автоматизації, як-от Selenium, Cypress, Playwright або Puppeteer, для автоматизації тестування електронної пошти.

API Temp Mail також є хорошим вибором для створення програм або веб-сайтів, які регулярно створюють тимчасові одноразові електронні листи.

Щоб використовувати API Temp Mail необхідно вибрати одну з двох платформ керування API:

- ApiLayer (рис. 2.9).

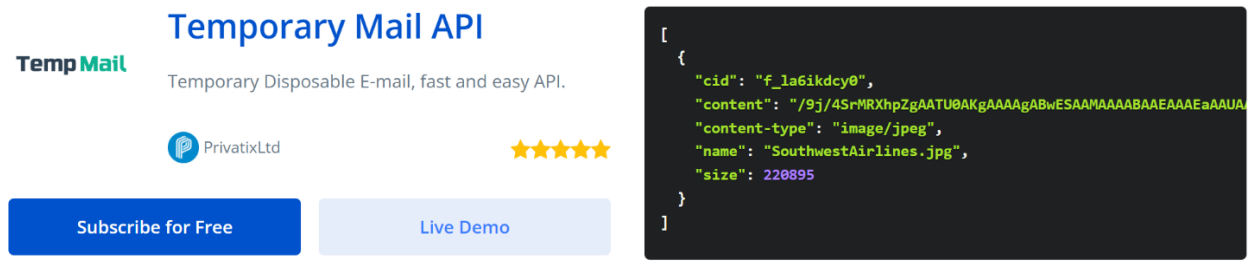


Рис. 2.9 API Temp Mail на платформі ApiLayer.

Безкоштовний тарифний план цього API передбачає до 500 запитів у місяць, чого на наш погляд, не достатньо для повноцінного відлагодження роботи нашого бота.

- RapidAPI (рис. 2.10). Натомість платформа має цілком безкоштовний тарифний план, який дозволяє виконати до 100 запитів даних з API TempMail за день, що вже, на нашу думку, цілком достатньо для відлагодження роботи бота.

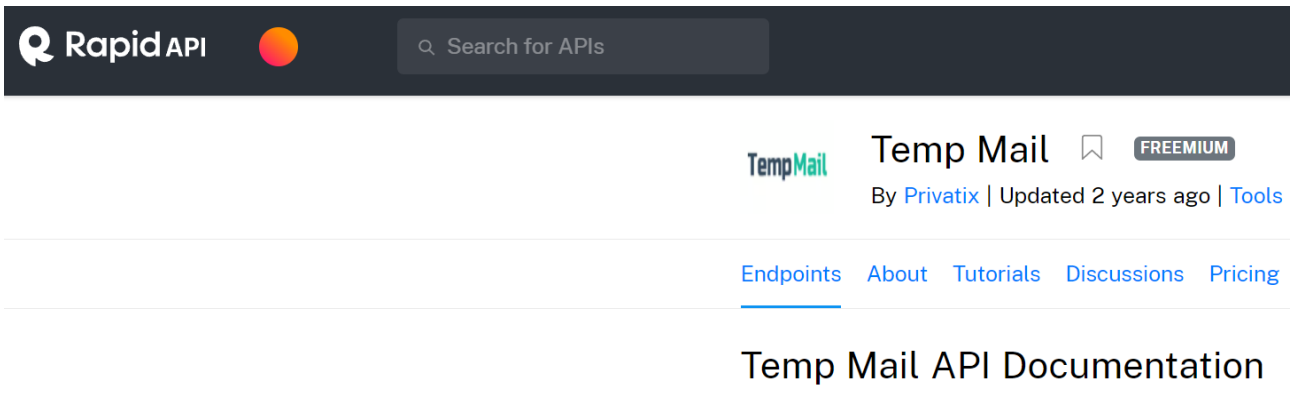


Рис. 2.6 API Temp Mail на платформі RapidAPI.

Тому очевидно, що наш вибір впав саме на цю платформу яка дозволяє отримати доступ до API сервісу TempMail з потрібним нам рівнем функцій для реалізації чат-боту.

Ще одним недоліком обох цих платформ, є некоректна робота з використанням безкоштовного тарифного плану з запитами на дії зі створеною за допомогою бота поштою.

# 3. ПРОЕКТУВАННЯ ЧАТ-БОТУ POSTMAN ДЛЯ МЕСЕНДЖЕРА TELEGRAM

## 3.1 Реєстрація чат боту для API Telegram

Для початку роботи розробки програми потрібно зареєструватись в спеціальний чат-бот «BotFather». Реєстрація починається з команди "/ Newbot", після чого потрібно ввести назву чату: "Bot" або "_bot" повинен бути вказаний в кінці, що є обов'язковою умовою. Після виконання всіх умов, BotFather видає ключ (спеціальний набір символів для доступу до HTTP API Telegram Bot) і URL- адресу доступу до чат-бота. Приклад реєстрації наведено на рис. 3.1.

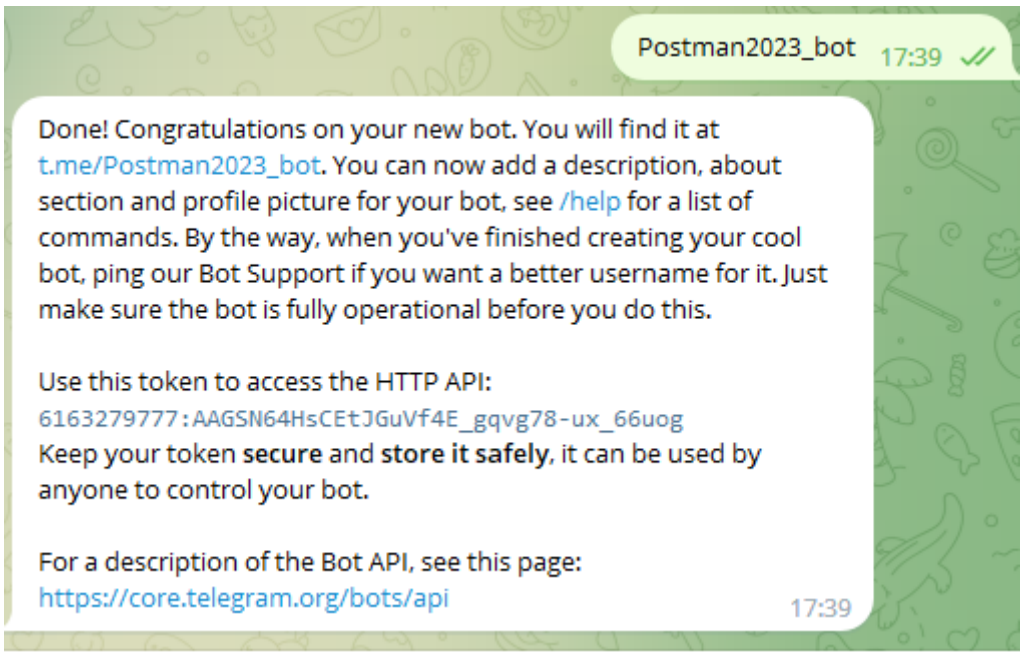


Рисунок 3.1. – Реєстрація чат боту в «BotFather»

Токен – це унікальний рядок із символів, який потрібний для того, щоб встановити справжність бота в системі. Токен генерується при створенні бота.[10]

Для встановлення додаткових параметрів: іконка чат-бота, повідомлення-вітання, опис чат-бота, а також можливість видалення існуючого бота існують наступні команди (Табл. 3.1).

Таблиця 3.1 - Доступні команди для зміни чат-ботів

| Команда | Опис |
|---|---|
| /setname | Зміна імені |
| /setdescription | Створює текст, який буде відображатись при першому відкриті чат-бота |
| /setabouttext | Зберігає текст в полі «Про чат-бота» |
| /setuserpic | Зберігає вибраний рисунок |
| /setcomands | Дозволяє створити список доступних команд |
| /deletebot | Видаляє вибраного чат-бота |

Окрім команд на зміну основних параметрів чат-бота, існують команди, які дають змогу відображати фіксовані параметри токену, а також призначати значення, представлені в таблиці 2.

Таблиця 3.2. - Доступні команди для додаткової конфігурації чат-бота

| Команда | Опис |
|---|---|
| /token | Повертає отриманий раніше токен у вибраного бота |
| /revoke | Анулює токен доступу до бота |
| /setinline | Включає чи вимикає можливість викликати бота з інших чатів |
| /setinlinegeo | Включає або вимикає можливість передавати розташування бота з іншого чату |
| /setinlinefeedback | Дає можливість отримати інформацію про кількість вибраних користувачами команд |

Продовження таблиці 3.2

| | |
|---|---|
| /setjoingroup | Визначає чи є можливість додати в групові діалоги бота |
| /setprivcy | Включає режим конфіденційності. В цьому режимі бот отримує, обробляє та відсилає назад інформацію окремо для кожного користувача в чаті |

Отримавши налаштування та токен на стороні Telegram, можна розпочати розробку програмної частини чат-бота.

## 3.2 Особливості використання сервісу, який надає тимчасові поштові скриньки

API Temp Mail є сервісом, який надає тимчасові поштові скриньки з відкритим доступом до вхідних повідомлень. Використана нами платформа RapidAPI дозволяє розробникам використовувати API Temp Mail для отримання даних про поштові скриньки та їх вміст за допомогою HTTP-запитів (рис. 3.1).

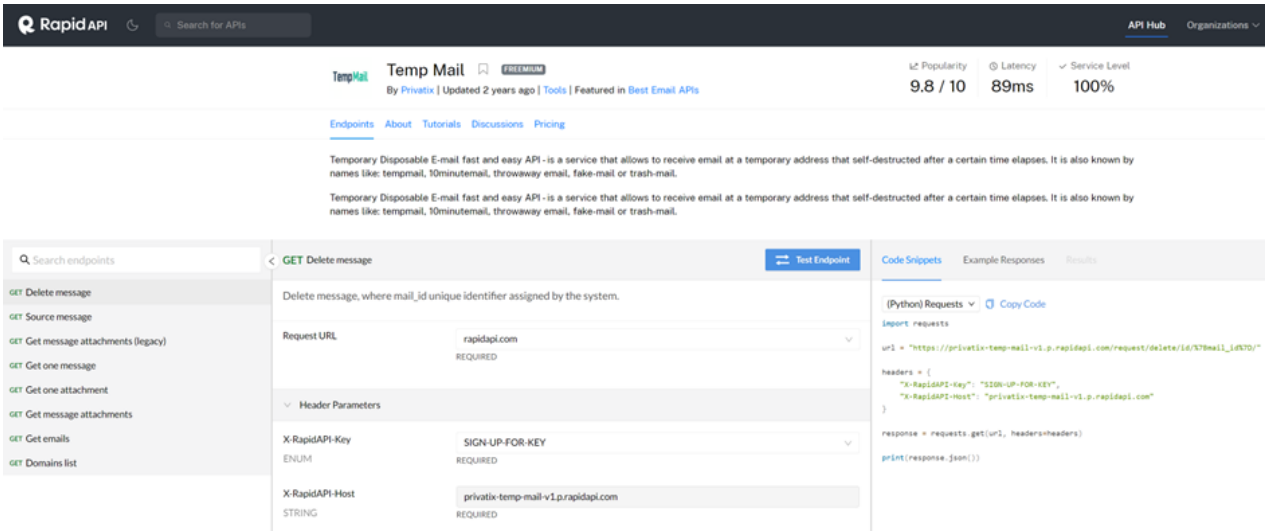


Рисунок. 3.2. – Загальний вигляд платформи RapidAPI

Принципи роботи з API Temp Mail на платформі RapidAPI включають наступні кроки:

- **Реєстрація та отримання ключа API.** Розробник повинен зареєструватися на платформі RapidAPI і отримати ключ API для доступу до сервісу Temp Mail. Цей ключ API використовується для аутентифікації запитів до API Temp Mail.
- **Встановлення заголовків запиту.** При виконанні запитів до API Temp Mail на платформі RapidAPI необхідно встановити певні заголовки, такі як "X-RapidAPI-Key" та "X-RapidAPI-Host", для ідентифікації та аутентифікації запитів (рис. 3.3).

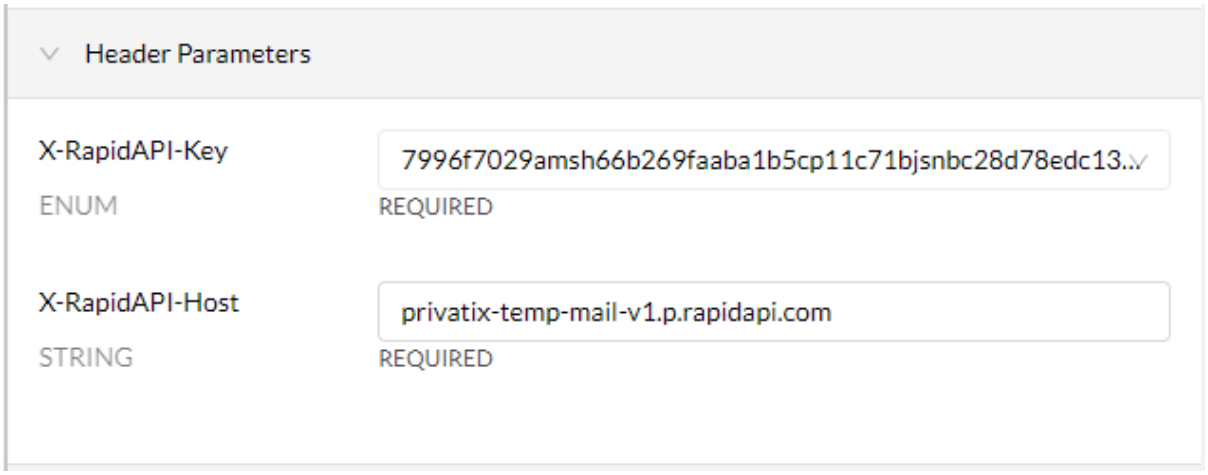


Рисунок 3.3. – Встановлені платформою RapidAPI заголовки HTTP-запитів

- **Виконання HTTP-запитів.** Розробник може використовувати різні HTTP-методи, такі як GET, POST, PUT або DELETE, для взаємодії з API Temp Mail на платформі RapidAPI. Запити виконуються за допомогою бібліотеки для роботи з HTTP, наприклад, requests в Python.
- **Обробка відповіді API.** Після виконання запиту до API Temp Mail, розробник отримує відповідь у форматі JSON або іншому підтримуваному форматі. Розробник може отримати доступ до різних полів відповіді, таких як статус, дані пошти, помилки тощо, і використовувати ці дані для подальшої обробки та відображення.

- **Обробка та використання даних.** Розробник може обробити отримані дані з API Temp Mail для відображення поштових повідомлень, перевірки статусу пошти, отримання вмісту листів тощо. Дані можуть бути використані для різних сценаріїв, таких як автоматичне отримання тимчасових листів, перевірка наявності нових листів або отримання прикріплених файлів.

### 3.3 Реалізація клієнтської частини чат-боту

Telegram користувачі можуть спілкуватись з чат-ботами двома способами: команди («/start», «/help» та інші) з відповідними параметрами, або вбудованою клавіатурою (inline keyboards). Нами обрано традиційний варіант спілкування з ботом через команди.

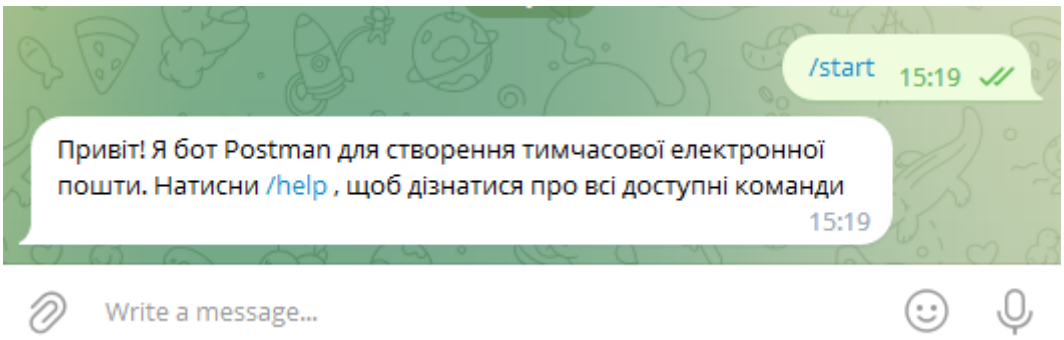


Рисунок 3.4. – Запуск команди «/start»

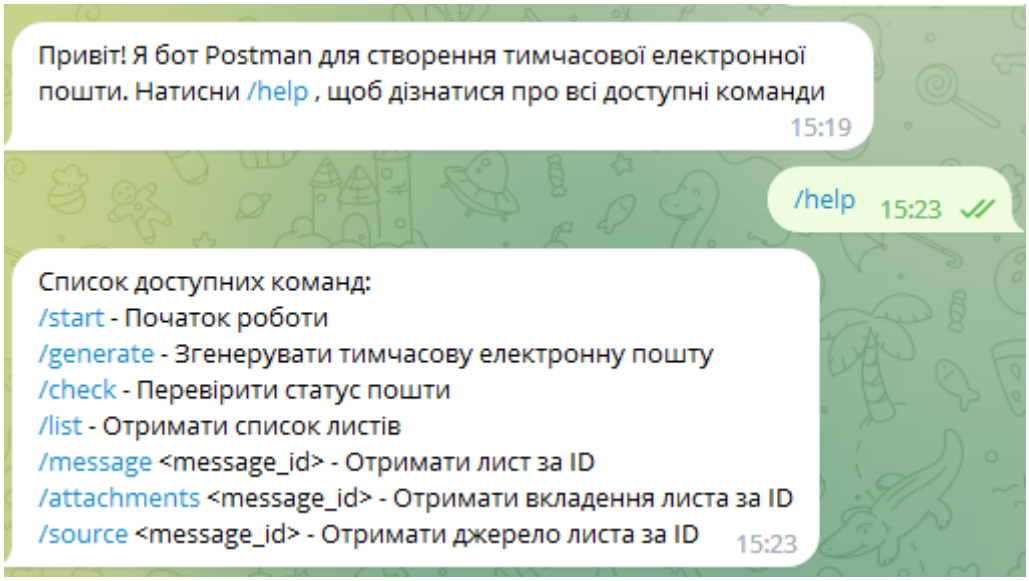


Рисунок 3.5. – Запуск команди «/help»

Для реалізації поставлених цілей потрібно зробити меню в якому після першого запуску «/start» (рис. 3.4) чат-бот буде вітатись, та пропонуватиме запустити команду /help, щоб ознайомитись з функціями які в ньому запрограмовані, як показано на рисунку 3.5.

Далі виконаємо і опишемо роботу основної функції цього бота – створення тимчасової адреси електронної пошти. При виконанні команди /generate бот генерує тимчасову поштову скриньку та її md5 хеш (рис. 3.6).

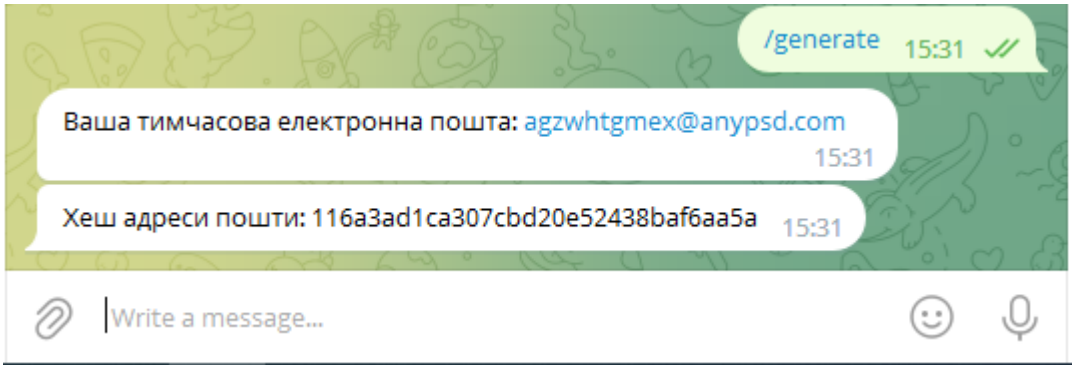


Рисунок 3.6. – Запуск команди «/generate»

Під час запуску цієї команди відбувається виконання функції generate_email, яка генерує тимчасову електронну адресу для користувача і відправляє її у повідомленні. Функція використовує інші допоміжні функції, такі як get_domains і generate_random_string. Код цієї функції зображено на рис. 3.7.

```python
def generate_email(update, context):
    domains = get_domains()

    if domains:
        random_domain = random.choice(domains)
        random_username = generate_random_string(10)
        email = random_username + random_domain

        context.user_data['email'] = email
        context.user_data['email_hash'] = hashlib.md5(email.encode()).hexdigest()

        context.bot.send_message(chat_id=update.effective_chat.id, text=f"Ваша тимчасова електронна пошта: {email}")
        context.bot.send_message(chat_id=update.effective_chat.id, text=f"Хеш адреси пошти: {context.user_data['email_hash']}")
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося отримати список доменів.")
```

Рисунок 3.7. – Код функції generate_email.

Функція get_domains відправляє запит на веб-сервер за допомогою модуля requests, щоб отримати список доступних доменів для електронної пошти. Вона встановлює заголовки для запиту, включаючи ключ API та хост, і виконує GET-запит за URL-адресою "https://privatix-temp-mail-v1.p.rapidapi.com/request/domains/". Якщо відповідь має статусний код 200 (OK), то вона повертає список доменів у форматі JSON. У протилежному випадку повертається значення None (рис. 3.8)

```
def get_domains():
    url = "https://privatix-temp-mail-v1.p.rapidapi.com/request/domains/"

    headers = {
        "X-RapidAPI-Key": "7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        domains = response.json()
        return domains
    else:
```

Рисунок 3.8. – Код функції get_domains.

Функція generate_random_string (рис. 3.9) генерує випадковий рядок заданої довжини length. Вона використовує модуль random та string.ascii_lowercase, щоб створити випадкові літери малої латинської абетки. За допомогою random.choices вибирається випадкова літера length разів, і результат об'єднується в одному рядку.

```
46  def generate_random_string(length):
47      letters = string.ascii_letters
48      return ''.join(random.choices(string.ascii_lowercase, k=10))
49
```

Рисунок 3.9. – Код функції generate_random_string.

У функції generate_email спочатку викликається функція get_domains, яка повертає список доменів. Якщо список не є порожнім, вибирається випадковий домен зі списку. Також генерується випадкове ім'я користувача, використовуючи

функцію generate_random_string. За допомогою об'єднання цих значень створюється тимчасова електронна адреса email.

Далі, отримані значення зберігаються у словнику context.user_data, який доступний для використання в майбутньому. Хеш адреси пошти обчислюється за допомогою алгоритму MD5 і зберігається також у словнику context.user_data.

Нарешті, два повідомлення надсилаються за допомогою context.bot.send_message. Перше повідомлення містить тимчасову електронну адресу email, а друге - хеш адреси пошти зі словника context.user_data['email_hash']. Якщо список доменів порожній, надсилається повідомлення, що не вдалося отримати список доменів.

Отже, функції get_domains, generate_random_string і generate_email співпрацюють, щоб створити тимчасову електронну адресу, зберегти її та хеш адреси в словнику context.user_data, і надіслати їх користувачеві через бота.

Наступна важлива команда цього бота, функціонал якої варто розглянути в рамках цієї кваліфікаційної роботи, є команда /check, яка призначена для перевірки статусу тимчасової електронної адреси користувача (рис. 3.10).

```python
def check_email(update, context):
    email_hash = context.user_data['email_hash']

    url = f"https://privatix-temp-mail-v1.p.rapidapi.com/request/keep/id/{email_hash}/"

    headers = {
        "X-RapidAPI-Key": "7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        email_data = response.json()
        if email_data['error'] == "":
            context.bot.send_message(chat_id=update.effective_chat.id, text=f"Статус пошти: {email_data['mail_status']}")
        else:
            context.bot.send_message(chat_id=update.effective_chat.id, text=email_data['error'])
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося перевірити статус пошти.")
```

Рисунок 3.10. – Код функції check_email.

Функція отримує хеш адреси електронної пошти зі словника context.user_data['email_hash']. Цей хеш був збережений під час генерації тимчасової електронної адреси в попередній функції.

З використанням отриманого хешу формується URL-адреса для виконання запиту перевірки статусу електронної пошти. У цьому випадку URL містить хеш адреси у вигляді шаблону {email_hash}.

Задаються заголовки для запиту, включаючи ключ API та хост. Запит відправляється за допомогою модуля requests методом get за вказаною URL-адресою з встановленими заголовками.

Перевіряється статусний код відповіді. Якщо статусний код дорівнює 200 (OK), то відповідь інтерпретується у форматі JSON за допомогою response.json(). Якщо поле error в JSON-відповіді порожнє, то надсилається повідомлення про статус пошти email_data['mail_status'] через бота. У протилежному випадку надсилається повідомлення з текстом помилки email_data['error'].

Якщо статусний код відповіді не дорівнює 200, то надсилається повідомлення про невдалу перевірку статусу пошти через бота.

Підсумовуючи все вищесказане, функція check_email отримує хеш адреси електронної пошти, виконує запит до веб-сервера з використанням цього хешу, отримує відповідь зі статусом пошти та обробляє цю відповідь для надсилання відповідного повідомлення користувачеві через бота.

```python
def get_email_list(update, context):
    context.bot.send_message(chat_id=update.effective_chat.id, text="Введіть хеш адреси електронної пошти:")
    context.bot.register_next_step_handler(update.message, process_email_list)

def process_email_list(update, context):
    email_hash = update.message.text

    url = f"https://privatix-temp-mail-v1.p.rapidapi.com/request/mail/id/{email_hash}/"

    headers = {
        "X-RapidAPI-Key": "7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        email_data = response.json()
        if email_data['error'] == "":
            mail_list = email_data['mail_list']
            if mail_list:
                context.bot.send_message(chat_id=update.effective_chat.id, text="Список листів:")
                for mail in mail_list:
                    context.bot.send_message(chat_id=update.effective_chat.id, text=mail['mail_text'])
            else:
                context.bot.send_message(chat_id=update.effective_chat.id, text="Немає листів.")
        else:
            context.bot.send_message(chat_id=update.effective_chat.id, text=email_data['error'])
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося отримати список листів.")
```

Рисунок 3.11. – Код функції get_email_list.

Наступним, ми опишемо код (рис. 3.11), що містить дві функції: get_email_list та process_email_list, які співпрацюють, щоб отримати та обробити список листів для певної електронної адреси.

Перша функція get_email_list надсилає повідомлення користувачеві через бота з проханням ввести хеш адреси електронної пошти, список листів якої необхідно отримати. Вона використовує context.bot.send_message для надсилання повідомлення, а потім реєструє process_email_list як наступний обробник кроку, що чекає на відповідь користувача.

Функція process_email_list обробляє введений користувачем хеш адреси електронної пошти. Вона отримує текст повідомлення від користувача через update.message.text і зберігає його в змінній email_hash.

Далі, з використанням отриманого хешу, формується URL-адреса для отримання списку листів за цим хешем. Заголовки для запиту встановлюються так само, як і в попередніх описаних функціях.

Виконується GET-запит за URL-адресою з встановленими заголовками. Якщо статусний код відповіді дорівнює 200, то відповідь інтерпретується у форматі JSON за допомогою response.json(). Якщо поле error в JSON-відповіді порожнє, то отримується список листів email_data['mail_list']. Якщо цей список не порожній, надсилається повідомлення "Список листів:", а потім для кожного листа у списку надсилається повідомлення з текстом листа mail ['mail_text']. Коли список листів порожній, надсилається повідомлення "Немає листів". Якщо поле error в JSON-відповіді не порожнє, надсилається повідомлення з текстом помилки email_data['error'].

У випадку, коли статусний код відповіді не дорівнює 200, надсилається повідомлення про невдале отримання списку листів через бота.

Функції get_email_list та process_email_list співпрацюють для отримання від користувача хешу адреси згенерованої ботом раніше чи будь-якої іншої тимчасової електронної пошти, виконання запиту до сервера для отримання списку листів за цим хешем, та надсилання цього списку листів користувачеві через бота. Функція get_email_list отримує список листів з тимчасової поштової

скриньки на основі введеного хешу адреси електронної пошти, а також отримує update і context як параметри, які передаються від фреймворка Telegram для обробки повідомлення. За допомогою context.bot.send_message, бот надсилає повідомлення користувачу з проханням ввести хеш адреси електронної пошти.

Хендлер context.bot.register_next_step_handler реєструє наступну функцію process_email_list як обробник наступного кроку. Це означає, що після введення користувачем хешу адреси електронної пошти, фреймворк Telegram автоматично викличе функцію process_email_list для подальшої обробки. Ця функція отримує введену користувачем хеш адресу електронної пошти з update.message.text. Далі формується URL для запиту до API Temp Mail з використанням отриманого хешу адреси електронної пошти.

Встановлюються необхідні заголовки, включаючи ключ API та хост. Далі виконується GET-запит до API Temp Mail з використанням requests.get(). Отримана відповідь зберігається в змінній response. Перевіряється код стану відповіді (response.status_code). У разі успішного запиту, дані про пошту отримуються з відповіді у форматі JSON за допомогою response.json() та зберігаються в змінну email_data.

Перевіряється наявність помилок у відповіді. Якщо поле error у email_data дорівнює порожньому рядку, це означає, що немає помилки і функція повертає запитані користувачем дані.

```python
def get_email_message(update, context):
    email_hash = context.user_data['email_hash']
    message_id = context.args[0]

    url = f"https://privatix-temp-mail-v1.p.rapidapi.com/request/mail/id/{email_hash}/message/{message_id}"

    headers = {
        "X-RapidAPI-Key": "7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        email_message = response.json()
        if email_message['error'] == "":
            context.bot.send_message(chat_id=update.effective_chat.id, text=f"Лист #{message_id}:")
            context.bot.send_message(chat_id=update.effective_chat.id, text=email_message['mail_text'])
        else:
            context.bot.send_message(chat_id=update.effective_chat.id, text=email_message['error'])
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося отримати лист.")
```

Рисунок 3.12. – Код функції get_email_message.

Функція get_email_message (рис. 3.12) отримує повідомлення з тимчасової поштової скриньки на основі хешу адреси електронної пошти та ідентифікатора повідомлення.

Отримується хеш адреси електронної пошти з context.user_data['email_hash'] та ідентифікатор повідомлення з context.args[0]. Далі формується URL для запиту до API Temp Mail з використанням отриманого хешу адреси електронної пошти та ідентифікатора повідомлення і встановлюються необхідні заголовки, включаючи ключ API та хост.

Після цього виконується GET-запит до API Temp Mail з використанням requests.get(). Отримана відповідь зберігається в змінній response. Перевіряється код стану відповіді (response.status_code).

У разі успішного запиту, дані про повідомлення отримуються з відповіді у форматі JSON за допомогою response.json() та зберігаються в змінну email_message. Перевіряється наявність помилок у відповіді. Якщо поле error у email_message дорівнює порожньому рядку, це означає, що немає помилок.

Наступним кроком виконання функції є відправка повідомлення з текстом листа за допомогою context.bot.send_message(). Першим повідомленням надсилається інформація про номер листа, а потім сам текст листа. У разі наявності помилки, надсилається повідомлення з текстом помилки. А у разі невдалого запиту, надсилається повідомлення про невдале отримання листа.

Функція get_email_attachments (рис. 3.16) призначена для отримання вкладень повідомлення з тимчасової поштової скриньки на основі хешу адреси електронної пошти та ідентифікатора повідомлення.

На початку виконання функції отримується хеш адреси електронної пошти з context.user_data['email_hash'] та ідентифікатор повідомлення з context.args[0]. На основі цих даних формується URL для запиту до API Temp Mail. Цей URL містить тег /source в кінці, що вказує на отримання джерела повідомлення з вкладеннями.

```python
def get_email_attachments(update, context):
    email_hash = context.user_data['email_hash']
    message_id = context.args[0]

    url = f"https://privatix-temp-mail-v1.p.rapidapi.com/request/mail/id/{email_hash}/message/{message_id}/source"

    headers = {
        "X-RapidAPI-Key": "7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        email_source = response.text
        if email_source != "":
            context.bot.send_message(chat_id=update.effective_chat.id, text=f"Вкладення листа #{message_id}:")
            context.bot.send_document(chat_id=update.effective_chat.id, document=email_source)
        else:
            context.bot.send_message(chat_id=update.effective_chat.id, text="У листа немає вкладень.")
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося отримати вкладення листа.")
```

Рисунок 3.16. – Код функції get_email_attachments.

Перевіряється код стану відповіді (response.status_code). Якщо код стану 200, це означає успішний запит. У разі успішного запиту, отримується текст джерела повідомлення з response.text та зберігається в змінну email_source.

Також здійснюється перевірка наявності тексту джерела повідомлення (email_source). Якщо він не дорівнює порожньому рядку, це означає, що є вкладення.

Після цього відправляється повідомлення з інформацією про номер листа та текстом "Вкладення листа". Також надсилається сам документ (текст джерела повідомлення) за допомогою context.bot.send_document().

При відсутності вкладень, надсилається повідомлення з текстом "У листа немає вкладень" або у разі невдалого запиту, надсилається повідомлення про невдале отримання.

Функція get_email_source (рис. 3.16) призначена для отримання джерела (source) повідомлення з тимчасової поштової скриньки на основі хешу адреси електронної пошти та ідентифікатора повідомлення. Основний принцип роботи цієї функції полягає в наступному:

Отримується хеш адреси електронної пошти з context.user_data['email_hash'] та ідентифікатор повідомлення з context.args[0].

```python
def get_email_source(update, context):
    email_hash = context.user_data['email_hash']
    message_id = context.args[0]

    url = f"https://privatix-temp-mail-v1.p.rapidapi.com/request/mail/id/{email_hash}/message/{message_id}/attachments"

    headers = {
        "X-RapidAPI-Key": "7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        email_attachments = response.json()
        if email_attachments['error'] == "":
            context.bot.send_message(chat_id=update.effective_chat.id, text=f"Джерело листа #{message_id}:")
            context.bot.send_message(chat_id=update.effective_chat.id, text=email_attachments['source'])
        else:
            context.bot.send_message(chat_id=update.effective_chat.id, text=email_attachments['error'])
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося отримати джерело листа.")
```

Рисунок 3.16. – Код функції get_email_source.

\

На основі них формується URL для запиту до API Temp Mail з використанням отриманого хешу адреси електронної пошти та ідентифікатора повідомлення. URL містить /attachments в кінці, що вказує на отримання джерела повідомлення.

Після встановлення заголовків, включаючи ключ API та хост виконується GET-запит до API Temp Mail з використанням requests.get(). Отримана відповідь зберігається в змінній response.

Перевіряється код стану відповіді (response.status_code). Якщо код стану 200, це означає успішний запит. У разі успішного запиту, отримується JSON-об'єкт з даними джерела повідомлення з response.json() та зберігається в змінну email_attachments.

Далі перевіряється, чи є помилка в об'єкті джерела повідомлення (email_attachments['error']). Якщо помилка відсутня, то відправляється повідомлення з інформацією про номер листа та текстом "Джерело листа", а також надсилається саме джерело повідомлення за допомогою context.bot.send_message().

Останнє на чому хотілося б зупинитися в описі функціонування нашого бота, це функція main, яка виконує основний код програми для запуску бота і обробки команд (рис. 3. 17).

```python
def main():
    updater = Updater("6163279777:AAGSN64HsCEtJGuVf4E_gqvg78-ux_66uog", use_context=True)
    dp = updater.dispatcher

    dp.add_handler(CommandHandler("start", start))
    dp.add_handler(CommandHandler("generate", generate_email))
    dp.add_handler(CommandHandler("check", check_email))
    dp.add_handler(CommandHandler("list", get_email_list))
    dp.add_handler(CommandHandler("message", get_email_message, pass_args=True))
    dp.add_handler(CommandHandler("attachments", get_email_attachments, pass_args=True))
    dp.add_handler(CommandHandler("source", get_email_source, pass_args=True))
    dp.add_handler(CommandHandler("help", help))

    updater.start_polling()
    updater.idle()
```

Рисунок 3.17. – Код функції main.

Давайте розглянемо кожен крок цієї функції більш детально:

- Ініціалізація бота та диспетчера:

Створюється об'єкт Updater з переданим токеном бота, отриманим від BotFather. Встановлюється параметр use_context=True, щоб використовувати нову модель контексту для взаємодії з ботом. Далі, створюється об'єкт dp (диспетчер) для реєстрації обробників команд.

- Реєстрація обробників команд:

Використовуючи метод add_handler об'єкта диспетчера dp, реєструються обробники команд для різних команд бота. Наприклад, команда "/start" реєструється з обробником start, команда "/generate" - з обробником generate_email, і т.д. Кожен обробник команди виконує відповідну функцію для обробки команди.

- Запуск бота:

Використовуючи метод start_polling, запускається процес отримання та обробки оновлень бота. Бот починає відслідковувати нові повідомлення та виконувати відповідні обробники команд. Функція updater.idle() відслідковує, коли бот знаходиться в режимі очікування і не виконує додаткових команд.

В загальному підсумку функція main ініціалізує бота, реєструє обробники команд та запускає бота для обробки вхідних повідомлень і виконання відповідних функцій.

# ОХОРОНА ПРАЦІ

## 5.1. Аналіз небезпеки під час роботи комп'ютера

З'ясувалося, що під час роботи з комп'ютером найбільшому ризику піддаються зорова, опорно-рухова, нервово-психічна система, достовірно невідомо, що саме порушує її – випромінювання або постійна статична поза.

Дисплей – головне джерело небезпеки. Він випускає випромінювання декількох видів: рентгенівське, ультрафіолетове, інфрачервоне, електромагнітне. Для кожного з цих випромінювання розроблені гранично допустимі норми, проте вони досить умовні й різняться в кожній країні. Норми передбачають, що опромінюється весь організм людини, тоді як на ділі впливу піддається лише верхня частина тулуба. Згадані норми встановлені з розрахунку на кожен вид опромінення в окремо, хоча реально всі поля діють одночасно, а їх комплексний вплив досі не досліджено.

Крім того, відео-дисплейний термінал порушує рівновагу між позитивно і негативно зарядженими іонами в повітрі. Електростатичне поле дисплея притягає негативні іони, порушуючи тим самим загальний баланс атмосфери. Це також шкодить здоров'ю. Вже через годину роботи біля монітора спостерігається майже повне зникнення негативних іонів. Ось чому необхідно, щоб до робочого місця за комп'ютером проникав свіже повітря. У зв'язку з усіма цими небезпеками досить чітко регламентовані розміри столу і стільця для роботи з комп'ютером. Отже непорушна постава шкідливо впливає на скелетно-м'язову систему. Стіл повинен бути просторим, зі спеціальною підставкою для ніг, а робочий стілець повинен мати відрегульовану висоту, певний кут нахилу сидіння і спинки.

Джерел випромінювання є два. Системний блок і монітор.

1. Системний блок створює тільки електромагнітне поле (випромінювання). Правда є ще й шум від вентиляторів, але ця тема всім

зрозуміла і не вимагає пізнань електроніки. Шкода від електромагнітного поля однозначно є при високому рівні поля. Однак поле комп'ютер створює набагато менше, ніж мобільний телефон.

2. Монітор має два основних шкідливих фактора. Бета-випромінювання (а простіше, потік електронів), яке власне кажучи створює картинку на екрані, і висока напруга (як і в будь-якому телевізорі, воно досягає 16-20 кіловольт), викликає іонізацію повітря. Бета-випромінювання поширюється монітором в двох напрямках – вперед і назад. У старих телевізорах і моніторах випромінювання досягало одного або двох метрів від екрану (всі пам'ятають рекомендаційне сидіти ближче двох, а то й трьох метрів від телевізора). Тобто виходив отакий потужний прожектор, що стріляє в нас шквалом електронів. По дорозі вибиваючи електрони з молекул повітря, перетворюючи їх на позитивні іони, так шкідливі для людини. На даний момент монітори мають дуже низький рівень бета-випромінювання, тобто електрони вилітають за межі екрану на пару сантиметрів. Основне випромінювання монітора направлено назад. Тому «зона ураження» поширюється на метр-півтора. Ось її і слід уникати. Висока напруга примудряється відхоплюватиму молекул повітря електрони, також перетворюючи молекули під шкідливі позитивні іони. До виробників моніторів і телевізорів пред'являються все більш жорсткі вимоги щодо використання високих напруг, і це не може не радувати.

## 5.2. Освітлення та вентиляція в робочому приміщенні

За правилами, світло при роботі з комп'ютером повинне падати зліва, а відстань від очей до екрана має бути близько 50 сантиметрів. Крім того, крісло слід відрегулювати так, щоб очі були на одному рівні з центром монітору. Фахівці говорять, що сааме очі найбільш страждають при роботі з комп'ютером. Виявляється, коли довго дивишся на екран, перестаєш моргати. Тому очі червоніють, сльозяться, а значить, знижується зір. Невелику відстань до екрану, дрібний шрифт, мерехтіння, різне освітлення призводять, у кінцевому рахунку,

до короткозорості. Якщо очі червоніють, сльозяться, з'являється печіння, починає боліти голова – це вже ознаки того, що очі втомилися, і треба відпочити. Але краще, звичайно, до такого стану себе не доводити.

## 5.3. Інструкція з охорони праці під час роботи за комп'ютером

Персонал, що працює на комп'ютері зобов'язаний дотримуватися вимог інструкції, розробленої на підставі Санітарних норм і правил, а також нести особисту відповідальність за дотримання вимог безпеки своєї праці та за створення небезпечного або шкідливого виробничого фактора для інших працюючих і поломки комп'ютера.

При роботі з комп'ютером шкідливими і небезпечними факторами є:

• електростатичні поля;

• електромагнітне випромінювання;

• наявність потужних іонізуючих випромінювання;

• локальне стомлення, загальне стомлення;

• стомлюваність очей;

• небезпека ураження електричним струмом;

• пожежонебезпека.

Режими праці та відпочинку при роботі з комп'ютером повинні організовуватися в залежності від виду та категорії трудової діяльності. Види трудової діяльності поділяються на3 групи:

• Група А – робота з зчитування інформації з екрана комп'ютера з попереднім запитом;

• Група Б – робота з введення інформації;

• Група В – творча робота в режимі діалогу.

За основну роботу з комп'ютером слід приймати таку, що займає не менше 50% часу протягом часу роботи за комп'ютером. Для видів трудової діяльності встановлюється 3 категорії тяжкості і напруженості роботи з комп'ютером, які визначаються:

для групи А – по сумарному числу прочитуються знаків за час роботи з комп'ютером, але не більше 60 000 знаків;

для групи Б – по сумарному числу зчитуються або вводяться знаків за час роботи з комп'ютером, але не більше 40000 знаків;

для групи В – по сумарному часу безпосередньої роботи з комп'ютером, але не більше 6 годин за час роботи з комп'ютером. Для забезпечення оптимальної працездатності і збереження здоров'я протягом часу роботи з комп'ютером повинні встановлюватися регламентовані перерви.

Перед початком роботи необхідно переконатися, що монітори комп'ютера мають анти блокове покриття (крім групи А) з коефіцієнтом відображення не більше 0,5. Покриття повинне також забезпечувати зняття електростатичного заряду з поверхні екрана, іскріння і накопичення пилу. Корпус монітора повинен забезпечувати захист від іонізуючих та неіонізуючих випромінювань.

Необхідно перевірити робоче положення комп'ютера відстань між стіною з віконними прорізами і столом повинно бути не менше 0,8 м. Відстань між робочими столами повинна бути не менше 1,2 м. Не допускається знаходження другого робочого місця з боку задньої сторони.

# ВИСНОВКИ ТА ПРОПОЗИЦІЇ

Проведено аналіз ринку месенджерів, та встановлено, що сучасні месенджери мають безліч схожих функцій, але на особливу увагу заслуговує месенджер Telegram, як кросплатформенний додаток, яким можна користуватись на телефонах, планшетах та ПК.

Здійснено огляд сервісів для отримання тимчасових адрес електронної пошти, який дав можливість, серед усіх досліджених вибрати декілька варіантів, які дозволяють за рахунок використання власного API створювати сторонні додатки та боти на популярних месенджерах.

Проаналізувавши функціонал аналогічних діючих чат-ботів для створення тимчасових адрес електронної пошти на вибраній нами платформі, ми дійшли висновку, що не зважаючи на доволі різноманітний асортимент активних чат-ботів зі схожою функціональністю, на нашу думку є потреба у подальшому удосконаленні таких ботів, додаванням нових команд, які дозволять більш продуктивно використовувати тимчасові електронні скриньки.

Описано особливості розробки чат-ботів та вибрано засоби розробки, серед яких мова програмування Python та фреймворк Python Telegram Bot для роботи з API месенджера Telegram. Також вибрано середовище розробки Visual Studio Code і платформу API для взаємодії з сервісом тимчасових поштових скриньок Telegram. Ці заходи дозволили нам приступити до розробки свого власного чат-бота.

Здійснено проектування чат-боту Postman, яке полягало у реєстрації чат боту для API Telegram, а також описано особливості використання сервісу, який надає тимчасові поштові скриньки.

Реалізовано клієнтську частину чат-боту. Детально описано процес його розробки та функціонування, а саме продемонстровано процес спілкування бота і користувача і пояснено роботу кожної важливої функції бота.

# СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Chatbot Conversations to deliver $8 billion in Cost savings by 2022 [Електронний ресурс]. – Режим доступу: https:// www. juniperresearch. com/analystxpress/july-2017/chatbot-conversations-to-deliver-8bn-cost-saving.

2. Global number of mobile messaging users 2016-2022. [Електронний ресурс]. – Режим доступу:https://www.statista.com/statistics/483255/number-of-mobile-messaging-users-worldwide/.

3. JSON: [Електронний ресурс]. – Режим доступу: https://www.json.org/json-ru.html

4. M. Murgia. Can Facebook Messenger Kill Off Apps [Електронний ресурс]/ The Telegraph, 15 Nov. 2015. – Режим доступу: www.telegraph.co.uk/technology/facebook/11996896/Can-Facebook- Messenger-kill-off-apps.html (дата звернення: 20.10.2019).

5. Messaging Apps & Brands: Telegram Messenger [Електронний ресурс]. – Режим доступу:https://www.messengerpeople.com/telegram-messenger/.

6. Messenger Platform [Електронний ресурс]. – Режим доступу: https://developers.facebook.com/docs/messenger-platform/.

7. Mindhacking [Електронний ресурс] – Режим доступу: https://github.com/owocki/mindhacking/blob/master/README.md (дата звернення 01.11.2019)

8. Python Requests : [Електронний ресурс]. – Режим доступу: https://2.python-requests.org/en/master/

9. Telegram Bot API. Tutorials [Електронний ресурс] – Режим доступу: https://core.telegram.org/bots/api/tutorials

10. Telegram Bot API: [Електронний ресурс]. – Режим доступу: https://core.telegram.org/bots/api

11. Telegram FAQ [Електронний ресурс]. – Режим доступу:https://telegram.org/faq

12. What is REST: [Електронний ресурс]. – Режим доступу: https://restfulapi.net/

13. Алгоритмізація та програмування: Практикум [Електронний ресурс]: навч. посіб. для здобувачів ступеня бакалавра за спеціальністю 122 “Комп’ютерні науки” / Л. І. Кублій; КПІ ім. Ігоря Сікорського. — Електронні текстові дані (1 файл: 28,15 Мбайт). — Київ: КПІ ім. Ігоря Сікорського, 2019. — 209 с.

14. Бот для Telegram на Python. Heroku сервер [Електронний ресурс]. – Режим доступу: https://tproger.ru/translations/telegram-bot-create-and-deploy/

15. Введення в JSON [Електронний ресурс] / JSON – Режим доступу: https://www.json.org/json-ua.html

16. Марк Лутц, Learning Python :2019, 720с

17. Огляд протоколу HTTP : [Електронний ресурс]. – Режим доступу: https:// developer. mozilla.org /ru/docs/ Web/HTTP/Overview

18. Основи програмування. Python. Частина 1 [Електронний ресурс]: підручник для студ. спеціальності 122 "Комп’ютерні науки", спеціалізації "Інформаційні технології в біології та медицині" / А. В. Яковенко ; КПІ ім. Ігоря Сікорського. – Електронні текстові данні (1 файл: 1,59 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 195 с.

19. Тестування телеграм-бота: [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/322816/

20. Топ-10 корисних Telegram-ботів для українських користувачів: [Електронний ресурс]. – Режим доступу: https://www.epravda.com.ua/publications/2017/08/7/627822

21. Трофименко О. Г., Прокоп Ю. В., Задерейко О. В. Алгоритмізація та програмування : навчально-методичний посібник. Одеса : Фенікс, 2020. 310 с.

22. Юрченко І.В., Сікора В.С. Програмування мовою Python: навчальний посібник.– Чернівці: Чернівецький національний університет, 2022.– 104 с.

# ДОДАТКИ

## Код Telegram бота Postman для створення тимчасових адрес електронної пошти

```
import hashlib
import random
import string
import requests
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters


def start(update, context):
    context.bot.send_message(chat_id=update.effective_chat.id, text="Привіт! Я бот Postman для створення тимчасової електронної пошти. Натисни /help , щоб дізнатися про всі доступні команди")


def generate_email(update, context):
    domains = get_domains()

    if domains:
        random_domain = random.choice(domains)
        random_username = generate_random_string(10)
        email = random_username + random_domain

        context.user_data['email'] = email
        context.user_data['email_hash'] = hashlib.md5(email.encode()).hexdigest()

        context.bot.send_message(chat_id=update.effective_chat.id, text=f"Ваша тимчасова електронна пошта: {email}")
        context.bot.send_message(chat_id=update.effective_chat.id, text=f"Хеш адреси пошти: {context.user_data['email_hash']}")
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося отримати список доменів.")


def get_domains():
    url = "https://privatix-temp-mail-v1.p.rapidapi.com/request/domains/"

    headers = {
        "X-RapidAPI-Key": "7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
```

```
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        domains = response.json()
        return domains
    else:
        return None


def generate_random_string(length):
    letters = string.ascii_letters
    return ''.join(random.choices(string.ascii_lowercase, k=10))


def check_email(update, context):
    email_hash = context.user_data['email_hash']

    url = f"https://privatix-temp-mail-
v1.p.rapidapi.com/request/keep/id/{email_hash}/"

    headers = {
        "X-RapidAPI-Key":
"7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        email_data = response.json()
        if email_data['error'] == "":
            context.bot.send_message(chat_id=update.effective_chat.id, text=f"Статус
пошти: {email_data['mail_status']}")
        else:
            context.bot.send_message(chat_id=update.effective_chat.id,
text=email_data['error'])
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося
перевірити статус пошти.")


def get_email_list(update, context):
```

```
    context.bot.send_message(chat_id=update.effective_chat.id, text="Введіть хеш адреси електронної пошти:")
    context.bot.register_next_step_handler(update.message, process_email_list)

def process_email_list(update, context):
    email_hash = update.message.text

    url = f"https://privatix-temp-mail-v1.p.rapidapi.com/request/mail/id/{email_hash}/"

    headers = {
        "X-RapidAPI-Key": "7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        email_data = response.json()
        if email_data['error'] == "":
            mail_list = email_data['mail_list']
            if mail_list:
                context.bot.send_message(chat_id=update.effective_chat.id, text="Список листів:")
                for mail in mail_list:
                    context.bot.send_message(chat_id=update.effective_chat.id, text=mail['mail_text'])
            else:
                context.bot.send_message(chat_id=update.effective_chat.id, text="Немає листів.")
        else:
            context.bot.send_message(chat_id=update.effective_chat.id, text=email_data['error'])
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося отримати список листів.")



def get_email_message(update, context):
    email_hash = context.user_data['email_hash']
    message_id = context.args[0]
```

```
    url = f"https://privatix-temp-mail-
v1.p.rapidapi.com/request/mail/id/{email_hash}/message/{message_id}"

    headers = {
        "X-RapidAPI-Key":
"7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        email_message = response.json()
        if email_message['error'] == "":
            context.bot.send_message(chat_id=update.effective_chat.id, text=f"Лист
#{message_id}:")
            context.bot.send_message(chat_id=update.effective_chat.id,
text=email_message['mail_text'])
        else:
            context.bot.send_message(chat_id=update.effective_chat.id,
text=email_message['error'])
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося
отримати лист.")


def get_email_attachments(update, context):
    email_hash = context.user_data['email_hash']
    message_id = context.args[0]

    url = f"https://privatix-temp-mail-
v1.p.rapidapi.com/request/mail/id/{email_hash}/message/{message_id}/source"

    headers = {
        "X-RapidAPI-Key":
"7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        email_source = response.text
        if email_source != "":
```

```
            context.bot.send_message(chat_id=update.effective_chat.id,
text=f"Вкладення листа #{message_id}:")
            context.bot.send_document(chat_id=update.effective_chat.id,
document=email_source)
        else:
            context.bot.send_message(chat_id=update.effective_chat.id, text="У листа
немає вкладень.")
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося
отримати вкладення листа.")


def get_email_source(update, context):
    email_hash = context.user_data['email_hash']
    message_id = context.args[0]

    url = f"https://privatix-temp-mail-
v1.p.rapidapi.com/request/mail/id/{email_hash}/message/{message_id}/attachments"

    headers = {
        "X-RapidAPI-Key":
"7996f7029amsh66b269faaba1b5cp11c71bjsnbc28d78edc13",
        "X-RapidAPI-Host": "privatix-temp-mail-v1.p.rapidapi.com"
    }

    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        email_attachments = response.json()
        if email_attachments['error'] == "":
            context.bot.send_message(chat_id=update.effective_chat.id, text=f"Джерело
листа #{message_id}:")
            context.bot.send_message(chat_id=update.effective_chat.id,
text=email_attachments['source'])
        else:
            context.bot.send_message(chat_id=update.effective_chat.id,
text=email_attachments['error'])
    else:
        context.bot.send_message(chat_id=update.effective_chat.id, text="Не вдалося
отримати джерело листа.")


def help(update, context):
    help_text = """Список доступних команд:
/start - Початок роботи
```

```
/generate - Згенерувати тимчасову електронну пошту
/check - Перевірити статус пошти
/list - Отримати список листів
/message <message_id> - Отримати лист за ID
/attachments <message_id> - Отримати вкладення листа за ID
/source <message_id> - Отримати джерело листа за ID """

    context.bot.send_message(chat_id=update.effective_chat.id, text=help_text)


def main():
    updater = Updater("6163279777:AAGSN64HsCEtJGuVf4E_gqvg78-ux_66uog",
use_context=True)
    dp = updater.dispatcher

    dp.add_handler(CommandHandler("start", start))
    dp.add_handler(CommandHandler("generate", generate_email))
    dp.add_handler(CommandHandler("check", check_email))
    dp.add_handler(CommandHandler("list", get_email_list))
    dp.add_handler(CommandHandler("message", get_email_message,
pass_args=True))
    dp.add_handler(CommandHandler("attachments", get_email_attachments,
pass_args=True))
    dp.add_handler(CommandHandler("source", get_email_source, pass_args=True))
    dp.add_handler(CommandHandler("help", help))

    updater.start_polling()
    updater.idle()


if __name__ == '__main__':
    main()
```