МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

Кваліфікаційна робота бакалавра

на тему: Розробка чат-боту для онлайн публікацій

Виконав студент групи К-20і
спеціальності 122 Комп'ютерні науки
Леонтьєв Михайло Едуардович

Керівник Штефан
Наталія Зінов'ївна

Консультант д.т.н., проф.
Казакова Надія Феліксівна

Рецензент д.т.н., професор
Мещеряков Володимир Іванович

Одеса 2022

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ 5
ВСТУП 6
1 АНАЛІТИЧНА ЧАСТИНА ПРОЕКТУ 7
1.1 Опис предметної області 7
1.2 Характеристика об’єкту розробки 8
1.3 Аналоги ботів у видавництві 10
1.4 Інструментальні засоби розробки чат-ботів 14
1.4.1 Телеграм API 17
1.4.2 Бібліотека Gson 19
2 ПРОЕКТУВАННЯ ОБ’ЄКТУ РОЗРОБКИ 21
2.1 Діаграми Use-Case 21
2.2 Діаграма активності UML 24
3 ОПИС РЕАЛІЗАЦІЇ 26
3.1 Налаштування бази даних PostgreSQL 26
3.2 Діаграма класів 31
3.3 Опис алгоритму роботи бота 33
3.3.1 Клас «TelegramServer» 37
3.3.2 Базові функції 38
3.4 Налаштування Клієнту 42
3.5 Основна логіка боту 44
3.6 Приклад роботи боту 50
ВИСНОВКИ 53
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ 54

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

API – Application Programming Interface

DOM – Document Object Model

JDBC – Java DataBase Connectivity

JDK – Java Development Kit

MVP – Minimum Viable Product

TDLib –Telegram Database Library

ORM – Object-relational mapping

POM – Project Object Model

SQL – Structured Query Language

UML – Unified Modeling Language

XML – eXtensible Markup Language

Gson – бібліотека на основі Java

Java – мова програмування

JAXB – аналізатора для XML

POJO – простий старий об'єкт Java

PostgreSQL – сама професійна з реляційних баз даних

Use-Case – варіант використання

ВСТУП

Чат-ботом зараз нікого не здивуєш. Їх використовують практично у всіх нішах: від держустанов до інтернет-магазинів. Чат-боти чудово справляються з різними завданнями:

- підбором товарів;
- оформленням замовлень;
- відповідями на часті запитання користувачів тощо.

Для багатьох користувачів месенджери – зручний канал комунікації, ніж спілкування по телефону або наживо, тому тема з чат-ботами трендова.

Чат-бот – корисний інструмент не тільки для служби підтримки, але й для рекламних завдань. Можна вести трафік з пошукових систем та соцмереж на чат-бота, всередині нього реалізувати автоворонку продажів, яка буде підігрівати користувачів корисним та цікавим контентом, а потім конвертувати в клієнтів, продаючи основний продукт. Відкриття повідомлень у месенджерах у рази вище, ніж email-розсилок, тому використання чат-ботів допомагає підвищити ефективність вкладень у маркетинг.

За даними опитування 2018 року, 62% користувачів Інтернету сказали, що їм подобається використовувати чат-боти для підтримки клієнтів. Це більше половини користувачів мережі. Ця статистика показує багатьом брендам і видавцям, що чат-боти мають потенціал, тим більше, що спектр їх використання ширший за підтримку читачів.

Чат-боти відкривають багато нових можливостей для видавців, які вирішують проблеми, з якими борються їхні клієнти. Більше того, ця технологія робить це ефективно та вчасно, без необхідності реєстрації чи входу [1].

1 АНАЛІТИЧНА ЧАСТИНА ПРОЕКТУ

1.1 Опис предметної області

Чат-боти – це програми, що автоматизують спілкування з користувачами. Наприклад, людина робить заявку на замовлення, а бот відповідає за заданим сценарієм. Чат-бот – гарний засіб автоматизувати комунікацію з користувачами у будь-якому бізнесі. Від саппорту та довідкової функції до маркетингу та прямих продажів – з усіма цими завданнями чат-боти добре справляються та показують непогані результати.

51% користувачів очікують від брендів негайної відповіді будь-якої доби. Для цього бізнеси запускають цілодобову службу підтримки та витрачають на це великі гроші. Але більшість питань користувачів є типовими, і на них цілком може відповідати правильно налаштована програма – наприклад, чат-бот.

За даними SalesForce, 69% користувачів віддають перевагу спілкуванню з ботами, тому що вони можуть отримати відповіді від бренду зі зручною для себе швидкістю. Роботу не складно відповісти 100 користувачам одночасно.

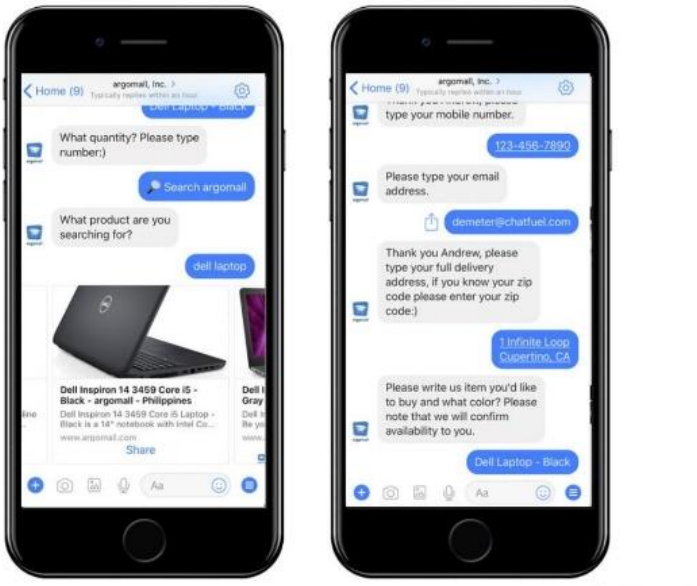

Рисунок 1 – Приклад роботи Чат-боту з продажу ноутбуків

Можливості чат-ботів для бізнесу:

1. Додатковий канал продажів З погляду продажів, чат-бот схожий на програму. У тому ж Telegram можна з виклику оплату з Apple Pay.
2. MVP мобільного додатка. Чат-бот – це інструмент для протестувати, тестування актуальності бізнесу робити мобільний додаток. Одночасно можна протестувати і маркетинг і розробку.
3. Збір фідбеку користувачів. Робот не може відповісти на запит, якого сценарій не готовий. Але ця слабкість допомагає збирати зворотний зв'язок від користувачів та покращувати продукт. Наприклад, у «Railwaybot» творці навчили робота відповідати на «Спасибо» завдяки фідбеку від користувачів.

У проекті «Ілон Маск» творці пішли ще далі – запам'ятовують усі нестандартні повідомлення від користувачів типу «Дякую» та відкритих питань. Потім аналізують їх і навчають на їх основі нейронну мережу, яка підбирає відповідні за змістом відповіді. За рахунок цього, діалог виходить природнішим, а бот намагається привести користувача до угоди [1].

1.2 Характеристика об’єкту розробки

Вважається, що сьогодні понад 40% всього інтернет-трафіку складається з бот-трафіку. У більшості випадків вони відповідають за сканування веб-сайтів, під час яких автоматизований сценарій витягує, аналізує та файлує інформацію з веб-серверів.

Вони присутні і у видавничій справі, і їх значення зростає. Боти, які копіюють поведінку реальних користувачів, постійно фільтрують трафік на видавничих веб-сайтах по всьому світу.

Оскільки вони все більше вдосконалюються та адаптуються, щоб виглядати як справжній відвідувач, інвестиції в автоматичне виявлення ботів і керування ними можуть бути корисними для видавців.

Звичайно, є різні види ботів, перші автоматизують завдання на користь користувачів, а другі, навпаки, створюють хаос, автоматизуючи спам-кампанії, атаки відмови в обслуговуванні або порушуючи безпеку мережі. У випадку видавничої індустрії, видалення поганих ботів є важливим, оскільки вони штучно збільшують кількість трафіку, спотворюючи аналітику поведінки користувачів.

Усунення поганих ботів, які збільшують фальшивий трафік, є важливим кроком у наданні видавцям точних даних, які їм потрібні, щоб покращити реальний досвід користувачів, стимулювати зростання реальних користувачів і отримати більше фінансових переваг від реклами [2].

Чат-боти можна використовувати у видавництві для:

1. Керування підписками.

Кількість видавців, які пропонують підписку, зростає, і чат-боти є одним із інструментів, які можуть полегшити читачам керування. Вони можуть виконувати основні службові завдання, як-от скасування підписок або щось більш просунуте, наприклад, допомагати контролювати час підписки. Передплатники можуть просто запитати у них, скільки часу триває до поновлення підписки або яка зараз ставка підписки, яка їх цікавить.

2. Індивідуальний дайджест.

Чат-боти можуть надавати читачам налаштовані дайджести, щоб отримувати інформацію на певні вибрані теми, як-от політика, бізнес, наука тощо. Читачі можуть отримувати дайджест так часто, як захочуть: раз на тиждень, раз на день або отримувати всю важливу інформацію. новини негайно.

3. Інформаційна підтримка читача.

Боти допомагають надавати читачам останні новини в тих сферах, які їх цікавлять, що підтримує взаємодію. Постачання актуального й оновленого контенту може призвести до зростання підписок.

4. Пропонування новинок.

Боти мають можливість дізнатися, що є потужною функцією.

Виходячи з поведінки користувачів – історій, які вони читають, кількості часу, проведеного з кожною історією, і суми всіх їхніх взаємодій, включаючи лайки та поширення – боти можуть рекомендувати вміст людям. Це можуть бути історії, які вони шукають, подібно до того, як бібліотекар може допомогти людям знайти книги в бібліотеці, і вони, ймовірно, зацікавлять читачів, перш ніж вони почнуть шукати цей вміст. Це також працює в інший бік. Вони можуть виявити відсутність інтересу до вмісту та усунути небажану інформацію.

5. Покращення збору даних.

Чат-боти можуть бути корисними як інструменти для збору даних читачів, таких як ім’я та електронна адреса користувача, моніторингу залучення читачів і збору інформації про них і теми, на які вони підписані або зацікавлені дізнатися більше. Усе це може бути дуже корисним для профілювання користувачів, для перенацілювання та створення індивідуальних потоків розмов для певних типів користувачів.

6. Покращення маркетингу електронною поштою.

Чат-боти можуть допомогти читачам підписатися на розсилку електронних листів. Чат-бот надасть вам індивідуальну інформацію про ваших потенційних клієнтів, у той час як ви можете підготувати більш персоналізований вміст і персоналізовані пропозиції для своїх рекламних акцій електронною поштою, покращуючи тим самим залучення користувачів [3].

1.3 Аналоги ботів у видавництві

Чат-боти використовуються як у газетах, так і у видавництві книг. Я вибираю три приклади відомих брендів, щоб показати вам, як вони використовують цю технологію.

Розглянемо декілька аналогів з предметної області дипломного проекту. Перший аналог – це Чат-бот «Epic Reads» (рис. 2). Харпер Коллінз додала

ботів для рекомендацій книг до Facebook. Це вирішує проблеми, з якими стикаються багато читачів.

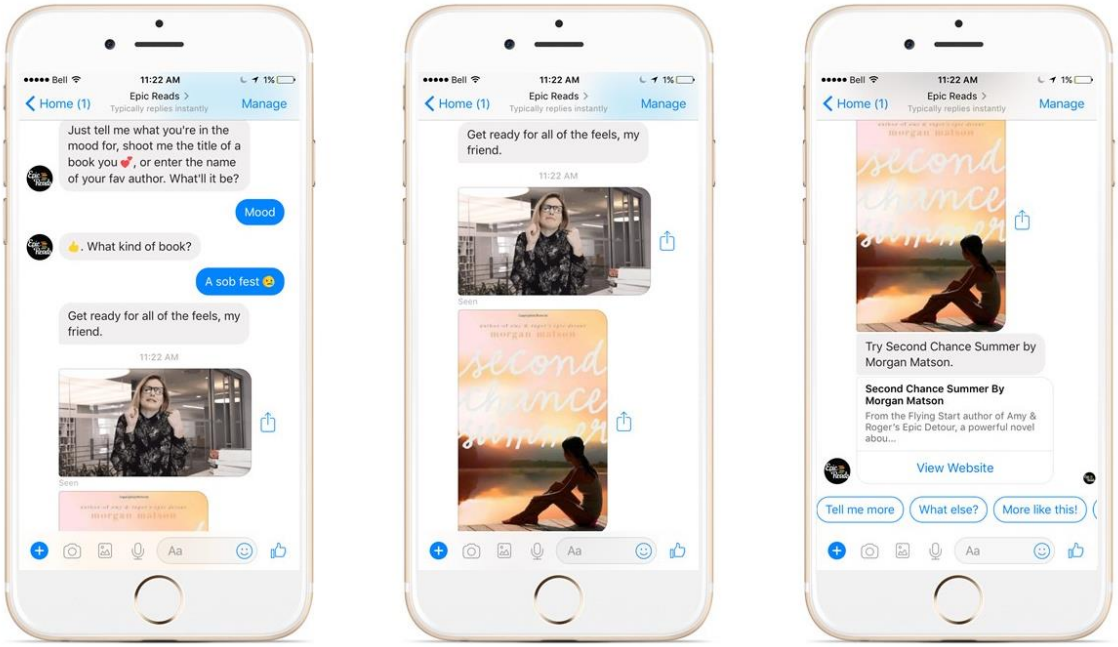

Рисунок 2 – Чат-бот Epic Reads

Harper Collins і «Massively» створили найкращий інструмент відкриття для книголюбів. Цей бот є не просто рекомендаційним механізмом, а 24/7 торговим агентом для одного з найбільших книговидавців у світі. Запущений у Facebook Messenger і Kik, Massively розробив масштабоване рішення, яке дає змогу видавцеві спілкуватися зі своєю спільнотою, залучати своїх шанувальників і заохочувати обмін вмістом.

За допомогою кількох простих запитань про минулі улюблені та жанрові уподобання цей бот зможе знайти книгу, яка відповідає вашим унікальним літературним смакам. Масово створений досвід для книголюбів, який відрізняється від самих книг.

Розробники використали мультимедійний підхід, який включає в себе gif-файли, запропоновані зображення та каруселі з посиланнями на вміст веб-сайту, щоб візуально залучити аудиторію, яка звикла споживати рядки тексту.

Подібна ідея використовується через «Macmillan Publishing» (рис. 3). Після того, як компанія виявилася нагромадженою частими запитами клієнтів про те, яку книгу отримати в подарунок чи просто прочитати, видавець вирішив принести користь своїм клієнтам, надавши їм особистого помічника, щоб спілкуватися та вибирати книгу, і водночас, автоматизувати управління комунікацією з клієнтами.

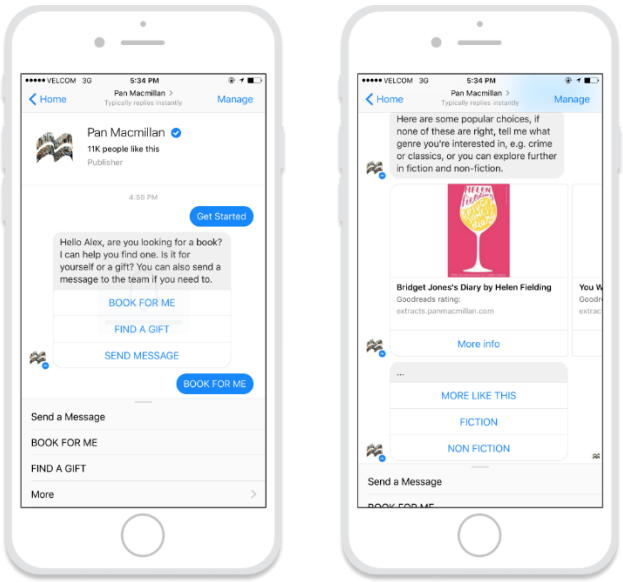

Рисунок 3 – Приклад роботи боту «Macmillan Publishing»

Пан Макміллан придумав єдине рішення, яке надасть мільйонам клієнтів персонального агента та ефективно вирішує повторні запити клієнтів. Так народився чат-бот «Pan Macmillan» для Facebook Messenger [4].

Цей бот не тільки автоматизує службу рекомендацій в інтерфейсі месенджера, впізнаваному клієнтами. Він також персонально ставиться до кожного клієнта та пристосовує свої рекомендації до потреб клієнта.

Бот «Pan Macmillan» – це рішення для чат-ботів, створене в співпраці між BAM Mobile та технологічним агентством «Digiteum». Чат-бот розмовляє з клієнтами, як людина, і допомагає їм вибирати й замовляти книги онлайн за допомогою розмовного інтерфейсу.

API, інтегрований з двома службами рекомендацій та рейтингу книг – GoodReads і Supadu – і підключений до Amazon, він надає клієнтам відповідну інформацію про книги, які їх цікавлять. Як розумний особистий агент, чат-бот спрощує вибір книг і дає рекомендації щодо відповідати навіть на найпростіші запити користувачів, як-от «жах» або «біографія».

Після вибору книги бот пропонує користувачам перенаправити їх на Amazon і зробити онлайн-замовлення без додаткових зусиль. Таким чином чат-бот стає додатковим каналом комунікації між брендом і його споживачами і стимулює продажі, збільшуючи таким чином дохід і прибуток.

Насамперед, чат-бот «Pan Macmillan» застосовує правила, щоб аналізувати введення користувачів і відповідати найкращим можливим сценарієм. Покладаючись на багате сховище тригерних ключових слів, бот розуміє уподобання клієнтів, запам’ятовує їх і надалі використовує цю інформацію, щоб розумно будувати свої рекомендації для книг [5].

Чат-боти також використовують видавці газет, як-от «The Wall Street Journal» (рис. 4). Глобальна інформаційна організація створила чат-бот Messenger, який надає людям доступ до щоденних інформаційних дайджестів журналу, останніх новин і даних про ринки.

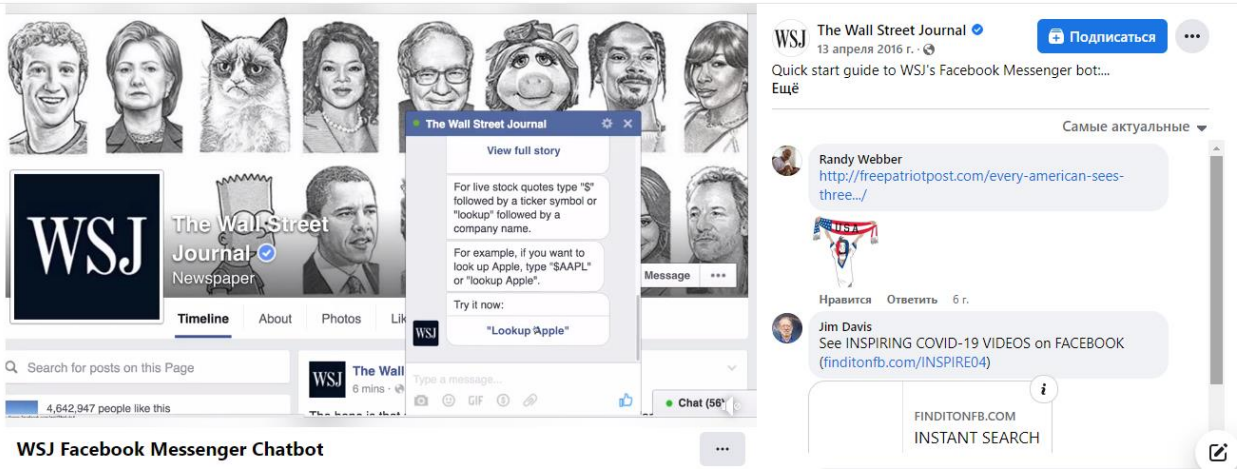

Рисунок 4 – Приклад роботи чат-боту «The Wall Street Journal»

Чат-бот допоміг WSJ залучити молодшу аудиторію та представити журналістику світового рівня новим читачам у всьому світі. 70% щомісячних активних користувачів чат-бота також є щоденними користувачами, і він придбав 130 тисяч цих щоденних активних користувачів за перші 14 місяців.

1.4 Інструментальні засоби розробки чат-ботів

Трендовість теми породила велику кількість сервісів, за допомогою яких можна без знання коду створювати чат-боти різного ступеня складності та використовувати їх для своїх проектів.

«Aimylogic» – російськомовний сервіс, де можна створювати розумні чат-боти (рис. 5). Створені чат-боти працюють на основі алгоритмів машинного навчання та постійно навчаються. Окрім чат-ботів в «Aimylogic» також можна створювати голосові роботи, налаштовувати сценарії розумних обдзвонив та впроваджувати їх у бізнес-процеси.

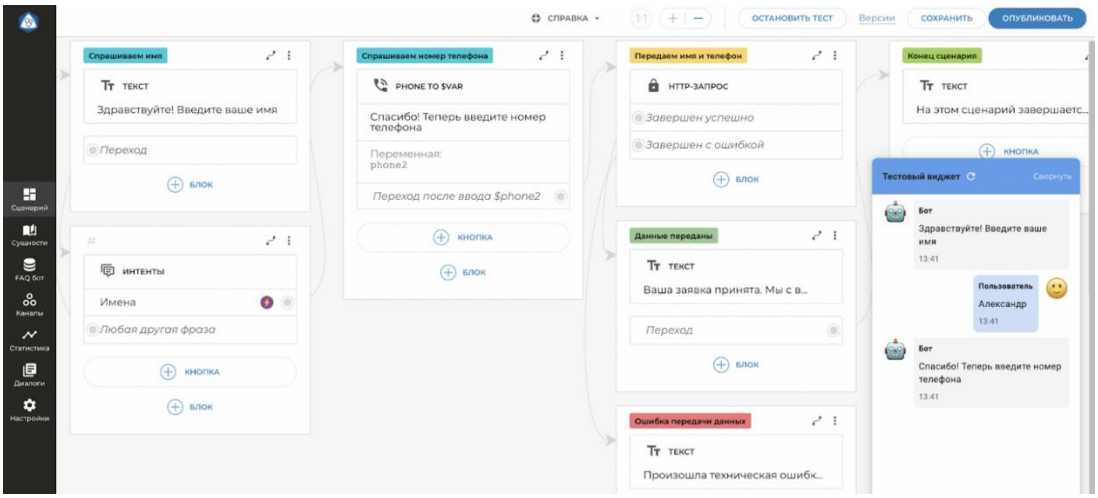

Рисунок 5 – Інтерфейс сервісу «Aimylogic»

Основні можливості чат-ботів:

– природного мовлення та базових смислів;

- за замовчуванням робот розпізнає вітання, прощання, згоду, відмову, подяку, а також нецензурну лексику;
- можна навчати бота на прикладі реальних фраз (наприклад, можна створити інтенти та додати кілька прикладів фраз користувачів, за допомогою машинного навчання бот розпізнаватиме більшість фраз, які відносяться до цієї теми);
- прийом платежів;
- готові інтеграції з популярними месенджерами, голосовими помічниками та іншими сервісами;

Другий аналог – сервіс «Flow XO». Англомовний онлайн-конструктор чат-ботів. Дозволяє вирішувати будь-які завдання бізнесу, такі як збирання лідів, автоматизація підтримки користувачів (відповіді на часті питання), прийом платежів, перемикання між чат-ботом та онлайн-чатом тощо.

Основні можливості:

- створення роботів за допомогою візуального конструктора;
- створення автоматизованих потоків для основних сценаріїв взаємодії з користувачами;
- прийом платежів;
- інтеграція зі 100+ сервісами (сервіси Google, CRM, платіжні системи тощо);
- збір статистики щодо взаємодії з користувачами;
- кроссплатформенна взаємодія з користувачами.

«Botmother» – сервіс позиціонує себе як CMS для чат-ботів (рис. 6). Дозволяє створювати чат-боти для соцмереж та месенджерів за допомогою конструктора та готових блоків.

Основні можливості:

- налаштування сценаріїв для взаємодії з користувачами;
- переключення між чат-ботом та онлайн-чатом (підключення до чату оператора);
- надсилання файлів користувачам (відео, зображення, інші файли);

– зчитування QR-кодів;
– прийом платежів (є інтеграція з Robokassa, ЮKassa та іншими платіжними системами);
– обмін даними з іншими сервісами (Botmother підтримує інтеграцію з сервісами Zapier, Albato та ApiX-Drive, через які можна зв'язати чат-бот практично з будь-яким сервісом – від Google Таблиць до сервісів розсилки та CRM);
– є доступ до діалогів із користувачами. Можна переглядати історію взаємодії з роботом або підключатися до чату в реальному часі, якщо робот не справляється з питаннями людини [6].

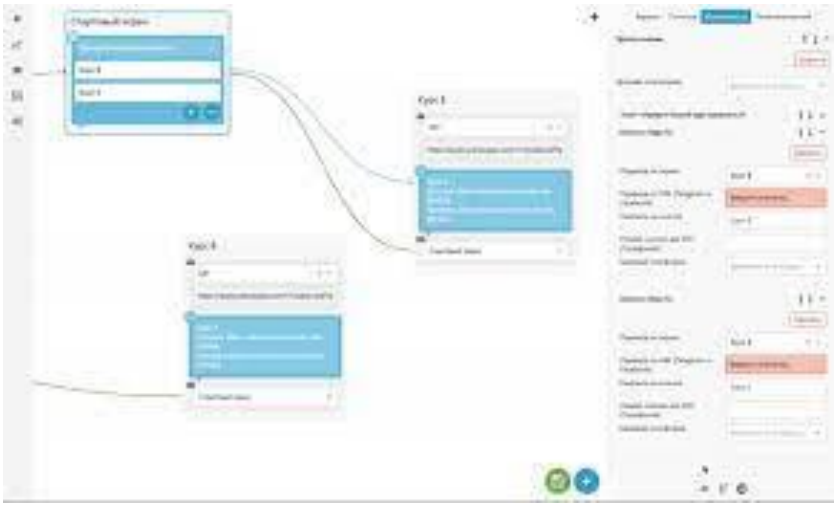

Рисунок 6 – Приклад роботи у сервісі «Botmother»

Останнім був оглянут серсів «BotFather» – найпростіший спосіб для реєстрації, налаштування та керування іншими telegram-ботами. Робота з ним проста і вимагає специфічних навичок. За допомогою «BotFather» можна зареєструвати необмежену кількість нових роботів.

По суті, Telegram Bots – це спеціальні облікові записи, для налаштування яких не потрібен додатковий номер телефону. Користувачі можуть взаємодіяти з ботами двома способами: надсилайти повідомлення та команди

ботам, відкриваючи з ними чат або додаючи їх до груп або надсилайти запити безпосередньо з поля введення, ввівши «@username» бота та запит. Це дозволяє надсилати вміст від вбудованих ботів безпосередньо в будь-який чат, групу чи канал.

Повідомлення, команди та запити, надіслані користувачами, передаються програмному забезпеченню, запущеному на ваших серверах. Сервер-посередник обробляє усе шифрування та спілкування з Telegram API. Користувачі спілкуються з цим сервером через простий HTTPS-інтерфейс, який пропонує спрощену версію API Telegram. Ми називаємо цей інтерфейс нашим API бота.

Боти Telegram унікальні в багатьох відношеннях – користувачу пропонується два види клавіатур, додаткові інтерфейси для команд за замовчуванням і глибокі посилання, а також форматування тексту, інтегровані платежі тощо [7].

Користувачі можуть взаємодіяти з ботом за допомогою вбудованих запитів прямо з поля введення тексту в будь-якому чаті. Все, що їм потрібно зробити, це розпочати повідомлення з іменем користувача вашого бота, а потім ввести запит.

Отримавши запит, бот може повернути деякі результати. Як тільки користувач торкається одного з них, він надсилається в поточний відкритий чат користувача. Таким чином, люди можуть запитувати вміст у вашого бота в будь-якому зі своїх чатів, груп або каналів.

За результатами аналізу аналогів було прийнято рішенн для реалізації бота у дипломному проекті використовувати сервіс «BotFather».

1.4.1 Телеграм API

На даний момент є два основні інструменти API, за допомогою яких можна використовувати сервіси Telegram – Telegram Bot API і Telegram API. Перший служить для розробки чат-ботів, другий дозволяє робити повністю

кастомні телеграм-клієнти. Розробникам також доступна відкрита бібліотека TDLib (Telegram Database Library), за допомогою якої можна створювати свою версію месенджера з унікальними опціями (наприклад, Telegram X, побудований саме на TDLib). Telegram Bot API є надбудовою над Telegram API, тому користуватися Bot API можна без знань про механізм використовуваного протоколу MTProto.

Для його роботи задіяний проміжний сервер з інтерфейсом HTTPS, який шифрує трафік і забезпечує зв'язок з Telegram API. Bot API дозволяє легко створювати програми, які використовують інтерфейс Telegram для виконання коду на локальному сервері. Користувачі можуть взаємодіяти з ботами, надсилаючи їм повідомлення, команди та вбудовані запити.

Принцип роботи будь-якого робота полягає в тому, що він перманентно надсилає запити на сервер і регулярно отримує оновлення. Отримувати їх можна двома способами. По-перше, можна використовувати вебхуки, коли сервер здійснює зворотний дзвінок на вказаний URL. По-друге, можна просто «закидати» запитами Telegram, отримуючи постійні відповіді.

На відміну від Bot API, де отримувати оновлення можна лише один раз, у Telegram API це обмеження можна обійти, якщо використовувати кілька клієнтів. У такому випадку робот отримуватиме всі оновлення на кожному із запущених клієнтів. Також у Bot API немає можливості розсилки повідомлень усім користувачам одночасно [8].

Telegram API дозволяє програмувати власний бот. Задавати різні команди взаємодії з користувачами, отримувати зручний доступ. Фактично без API програмісти ботів мали б щоразу писати свій власний штучний інтелект. З його використанням все стає набагато простіше.

У цьому API є вже готові функції виведення та введення тексту, відповіді на прописані запитання та інше. Тобто фактично розробнику залишається тільки вписати свій текст, якщо йдеться про зовсім примітивний робот.

1.4.2 Бібліотека Gson

Google Gson – це проста бібліотека на основі Java для серіалізації об'єктів Java у JSON і навпаки. Це бібліотека з відкритим кодом, розроблена Google. До основних характеристик слід віднести наступні:

1. Стандартизований – Gson – це стандартизована бібліотека, керована Google.
2. Ефективний – це надійне, швидке та ефективне розширення стандартної бібліотеки Java.
3. Оптимізовано – бібліотека оптимізована.
4. Підтримка дженериків – забезпечує велику підтримку дженериків.
5. Підтримує складні внутрішні класи – підтримує складні об'єкти із глибокою ієрархією спадкування.

Простота використання – Gson API забезпечує фасад високого рівня для спрощення часто використовуваних сценаріїв використання. Немає необхідності створювати зіставлення – API Gson забезпечує зіставлення за замовчуванням для більшості об'єктів, що серіалізуються.

Продуктивність Gson досить швидкий і займає мало пам'яті. Підходить для великих графіків об'єктів або систем. Чистий JSON – Gson створює чистий та компактний результат JSON, який легко читається.

Немає залежності – бібліотека Gson не вимагає будь-якої іншої бібліотеки, крім JDK. Open Source – бібліотека Gson із відкритим вихідним кодом; це вільно доступно [9].

Gson пропонує три альтернативні способи обробки JSON:

1. Потоковий API – він читає та записує вміст JSON як окремі події. JsonReader та JsonWriter зчитують/записують дані як токен, званий JsonToken.

Це найпотужніший підхід із трьох підходів до обробки JSON. Він має мінімальні накладні витрати та досить швидкий в операціях читання/запису. Це аналог парсера Stax для XML.

2. Модель дерева.

Він готує уявлення дерева JSON у пам'яті. Він будує дерево вузлів JsonObject. Це гнучкий підхід, аналогічний до аналізатора DOM для XML.

3. Прив'язка даних.

Він перетворює JSON на POJO (простий старий об'єкт Java) і назад, використовуючи метод доступу до властивостей. Gson читає/пишає JSON, використовуючи адаптери типів даних. Це аналог аналізатора JAXB для XML [10].

2 ПРОЕКТУВАННЯ ОБ’ЄКТУ РОЗРОБКИ

2.1 Діаграми Use-Case

Формулювання того, як клієнт буде взаємодіяти з продуктом або системою, необхідно для збору вимог та спілкування із зацікавленими сторонами на високому рівні. Діаграма моделі варіантів використання – це візуальне уявлення користувачів продукту, того, як вони взаємодіятимуть із продуктом і що продукт робить.

Варіант використання – це опис способів взаємодії користувача із системою або продуктом. Варіант використання може встановлювати сценарії успіху, сценарії відмови та будь-які критичні варіанти чи винятки. Варіант використання можна написати або візуалізувати за допомогою інструмента моделі варіанта використання.

Діаграма Use-Case використовує текст та фігури для представлення відносин між користувачем та системою. Насамперед діаграми моделей варіантів використання використовуються для:

– візуалізуйте потік та поведінку системи;
– проілюструвати функціональність системи;
– подання ключових взаємодій системи та користувача.

Залежно від системи діаграма моделі варіанта використання може різнитися за складністю, відображаючи основні асоціації або розширюючись відображення кількох винятків.

Use-Case визначають очікувану поведінку (що), а не точний спосіб її здійснення (як). Use-Case після вказівки можуть бути позначені як текстовим, так і візуальним представленням (тобто діаграма варіантів використання).

Ключова концепція моделювання варіантів використання полягає в тому, що воно допомагає нам проектувати систему з точки зору кінцевого користувача. Це ефективна техніка для передачі поведінки системи в термінах користувача шляхом визначення всієї зовнішньої поведінки системи [11].

На першому етапі моделювання системи побудуємо діаграму варіантів використання, яка демонструє процес створення власного боту у телеграм-каналі (рис. 7). На діаграмі присутні три актора: Службовий бот телеграм, Кліент (який намагається створити свого бота) та сам сервіс BotFather, якого було обрано після аналізу існуючих засобів розробки.

Головний варіант використання – це отримання токену – унікального ключу, за допомогою якого і реєструються нові боти у системі. Коли Клієнт вступає у діалог в приватному чаті з BotFather, він отримає пошагову інструкцію як налаштувати свого бота для подальшого запуску у телеграм-канал.

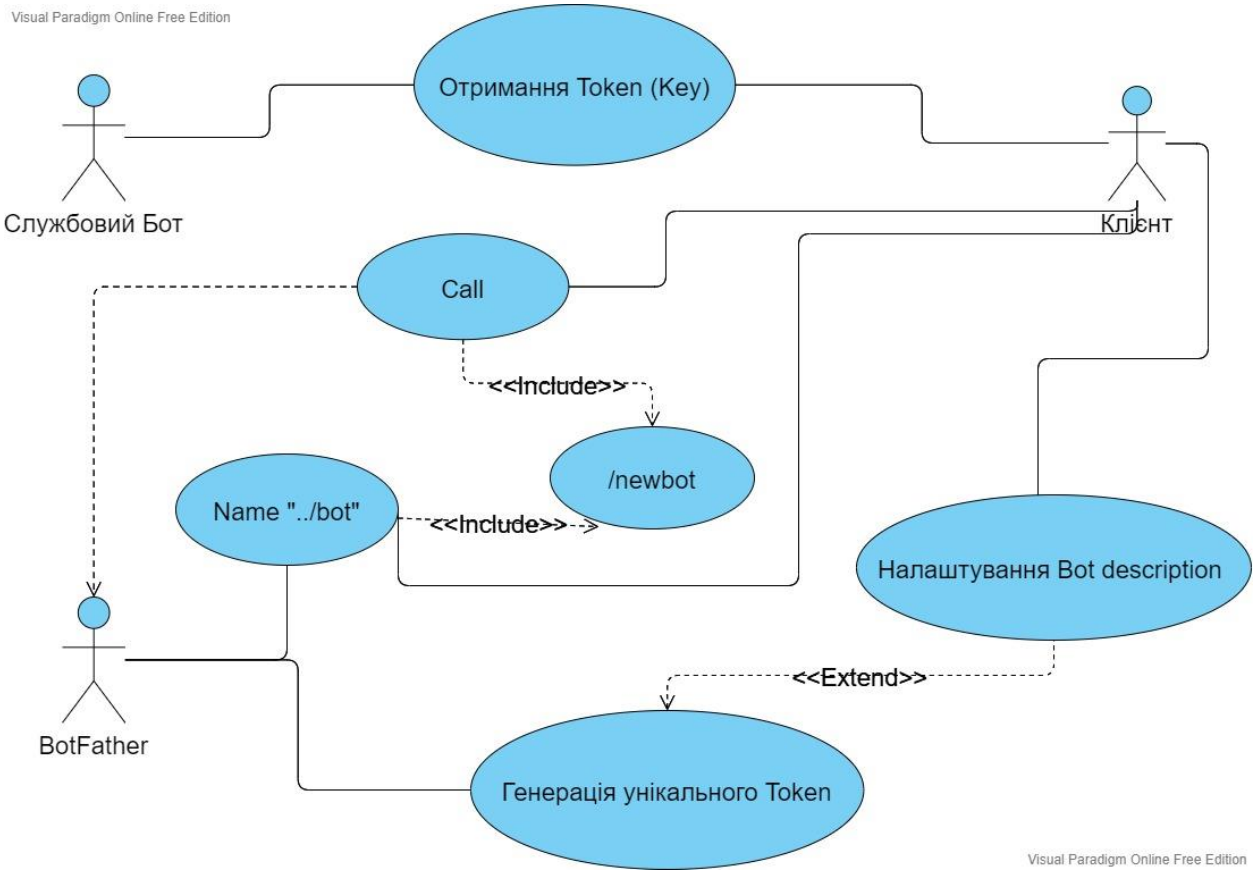

Рисунок 7 – Діаграма варіантів використання для процесу реєстрації бота

Наступна діаграма варіантів використання відображає основні дії по реєстрації нових авторів та етапи розміщення статті у чаті (рис. 8).

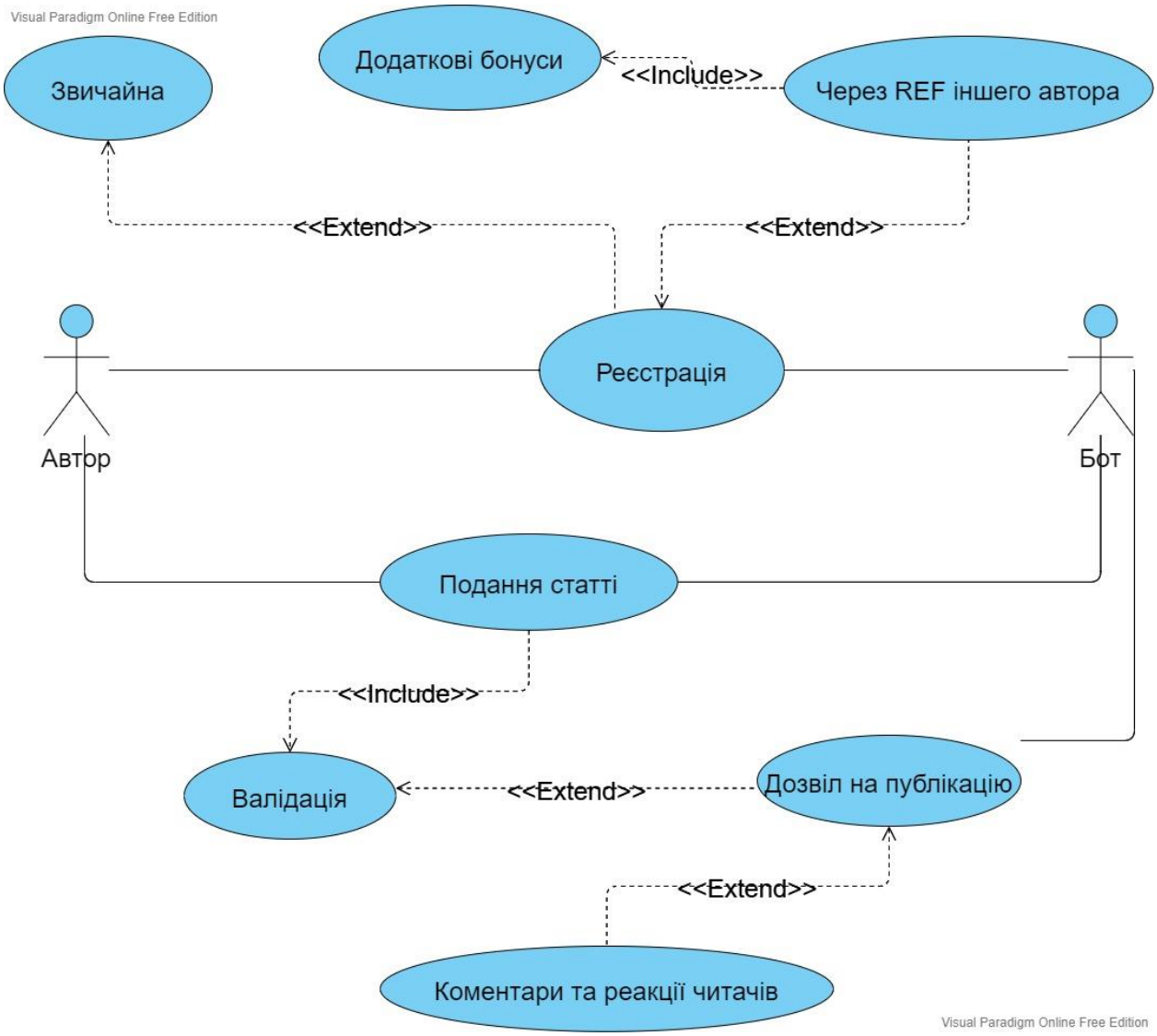

Рисунок 8 – Діаграма Use-Case реєстрації та публікації статті

Так, є два види реєстрації нового користувача (майбутнього автору): проста та через реферальне посилання іншого автора, який вже є у системі. Якщо реєстрація йде за другим варіантом, це надає додаткові бонуси власнику реферального посилання та новому учаснику, такі як збільшення кількості символів у пості, частота публікації у чаті.

Такий підхід допомагає як можна більше людей зацікавити телеграм-каналом на наростити собі аудиторію.

2.2 Діаграма активності UML

Діаграма діяльності (activity) є ще однією важливою діаграмою поведінки в діаграмі UML для опису динамічних аспектів системи. Діаграма діяльності, по суті, є розширеною версією блок-схеми, яка моделює перехід від однієї діяльності до іншої.

Потік управління переходить від однієї операції до іншої. Цей потік може бути послідовним, розгалуженим або одночасним. Діаграми діяльності розглядають усі типи керування потоком за допомогою різних елементів, таких як розгалуження, з'єднання тощо

Діаграми діяльності описують, як діяльність координується для надання послуги, яка може бути на різних рівнях абстракції. Як правило, подія повинна бути досягнута деякими операціями, особливо якщо операція призначена для досягнення ряду різних речей, які потребують координації, або як події в одному випадку використання пов'язані одна з одною, зокрема, випадки використання, коли дії можуть перетинатися і вимагати координації.

Така діаграма також підходить для моделювання того, як набір варіантів використання координується для представлення робочих процесів бізнесу.

Діяльність – це конкретна операція системи. Діаграми діяльності використовуються не тільки для візуалізації динамічної природи системи, але вони також використовуються для побудови виконуваної системи за допомогою методів прямого та зворотного проектування. Єдине, чого не вистачає на діаграмі діяльності, – це частина повідомлення [12].

На рисунку 9 представлена діаграма активності, яка демонструє основний алгоритм роботи у телеграм-каналі. Показано перелік дій, які необхідно виконати щоб створити власного боту та зареєструвати нових користувачів/авторів.

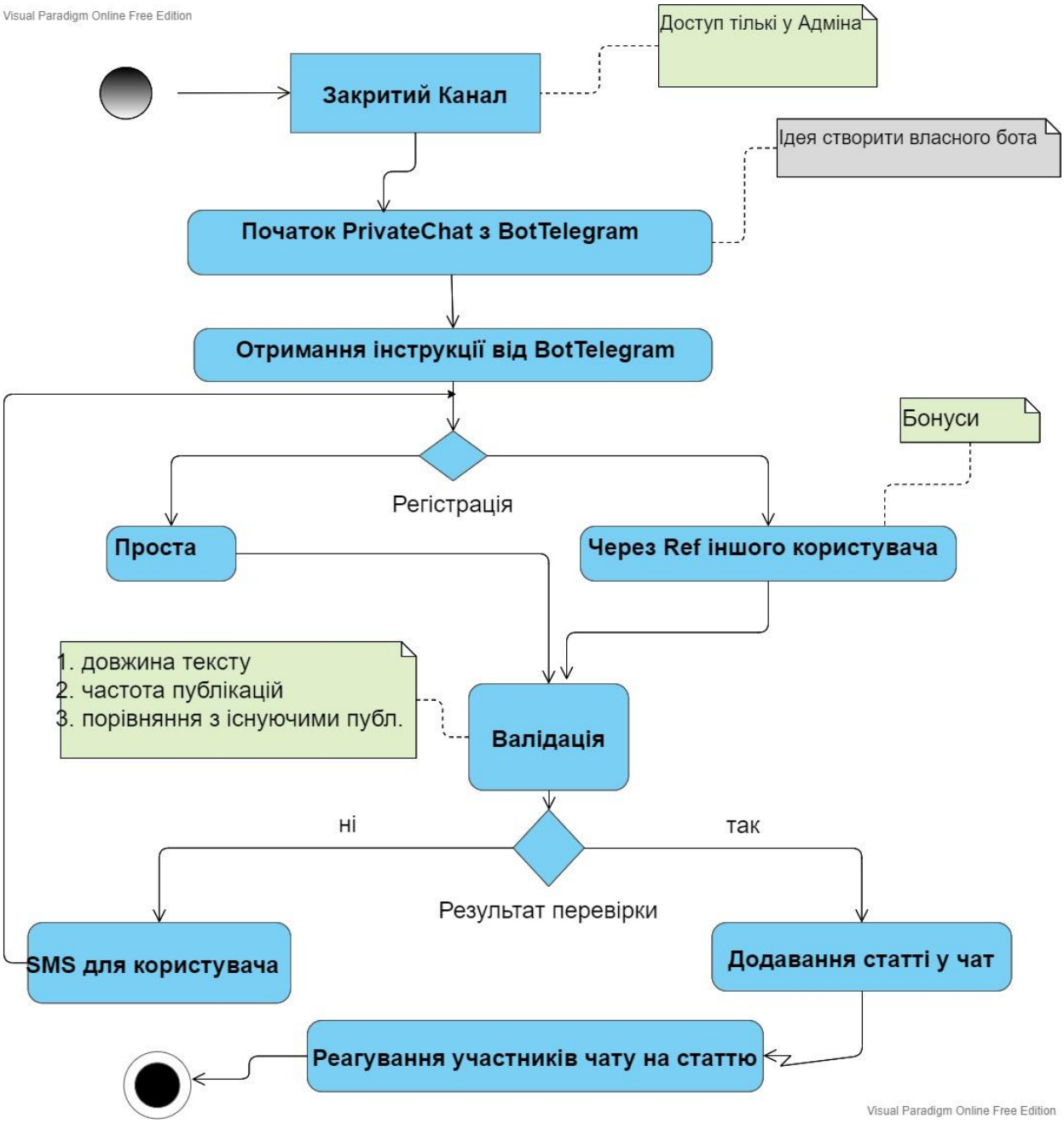

Рисунок 9 – Діаграма активності

3 ОПИС РЕАЛІЗАЦІЇ

3.1 Налаштування бази даних PostgreSQL

В якості БД був вибраний PostgreSQL. PostgreSQL – сама професійна з реляційних баз даних з відкритим вихідним кодом, кілька разів отримана нагороду «Система баз даних року». Це дуже надійна, стабільна, масштабована та безпечна система, яка існує вже більше двох десятиліть. Таким чином, вона зарекомендувала себе як великий гравець у світі баз даних з відкритим вихідним кодом і кидає виклик крупним гравцям, таким як Oracle, Sybase та IBM. PostgreSQL – це професійно підтримуване та розроблене програмне забезпечення, здатне запускати складні додатки, керовані даними [13].

Переваги PostgreSQL :

1. Відкритий вихідний код – він знаходиться у вільному доступі під ліцензію з відкритим вихідним кодом. Це дає вам можливість вільно використовувати, модифікувати та вносити його відповідно до потреб вашого бізнесу.
2. Скорочення витрат.

Будучи справжнім продуктом з відкритим вихідним кодом. Розробнику більше не потрібно споживатися про ліцензійні витрати, проблеми з контрактами та мати справу з високобюджетним продуктом. Він доступний для вас, коли ви цього хочете і так, як ви цього хочете.

3. Надійність.

Декілька компаній і окремих осіб вносять свій вклад у проект і внедряють інновації вже більше 25 років. Сільне співтовариство гарантує, що помилки будуть виправлені без промедлення. PG підтримує широкий спектр розширення, а також нескількими моделями даних SQL і NoSQL.

4. Безпека. Існує безліч функцій для підвищення безпеки завдяки простому розширенню: однак, якщо розробник використовує звичайні (TDE, маскування даних), вінотримує дуже безпечну базу даних

5. Масштабованість – база даних PostgreSQL може розвиватися разом з вами. Існує кілька технічних варіантів масштабування PostgreSQL. Ось чому база даних PostgreSQL може розвиватися разом із вами і може бути настільки великим, наскільки вам потрібно [14].

Першим етапом по роботі з БД є її створення у системі pgAdmin 4 за допомогою графічного редактору (рис. 11):

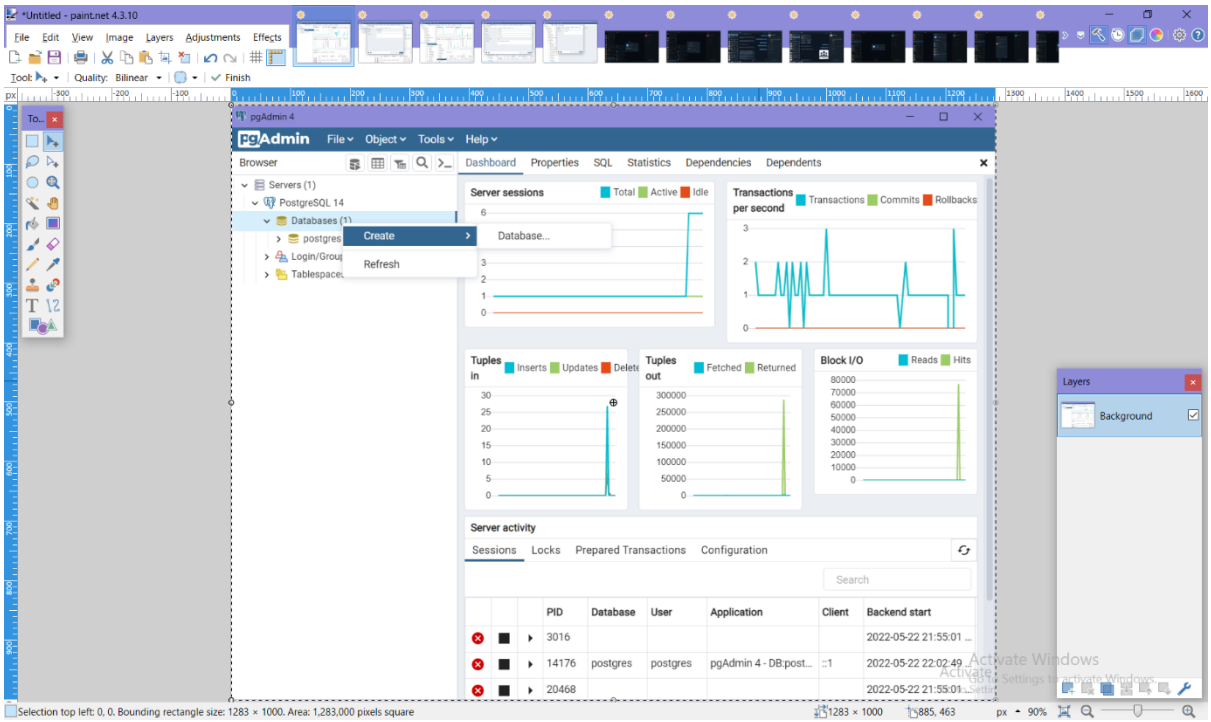

Рисунок 11 – Створення БД у pgAdmin 4

Після цього відкривається окно «Создание бази данных», де необхідно указати назву створюваної бази даних (рис. 12). І цього достатньо, щоб створити базу даних за умовчанням. В результаті в браузері з'явиться нова база даних.

Самі таблиці розміщені у схемах. Ми використали стандартну схему public. В відмінності від деяких інших СУБД, в PostgreSQL немає столбцов зі своїм auto_increment. Замість цього в постгресі використовуються послідовності (послідовності).

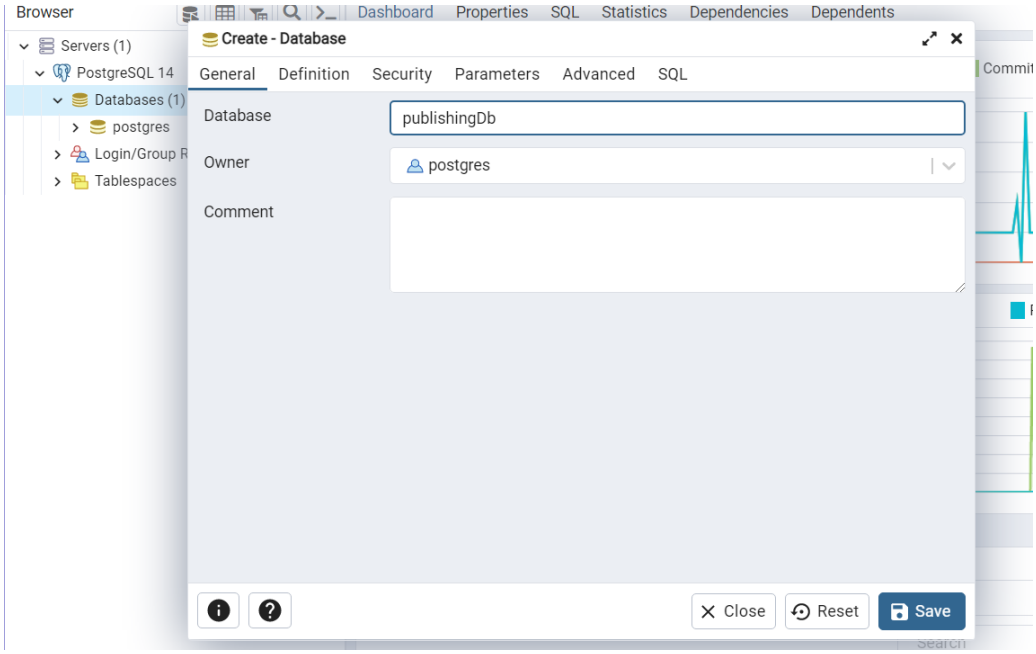

Рисунок 12 – Завдання параметрів для БД

Наступний етап – це створення таблиць до нашої БД. Для цього також використовуємо графічний редактор, що дозволить прискорити час на виконання всієї роботи. Приклад Завдання атрибутів до таблиці «Publication» представлено на рисунку 13:

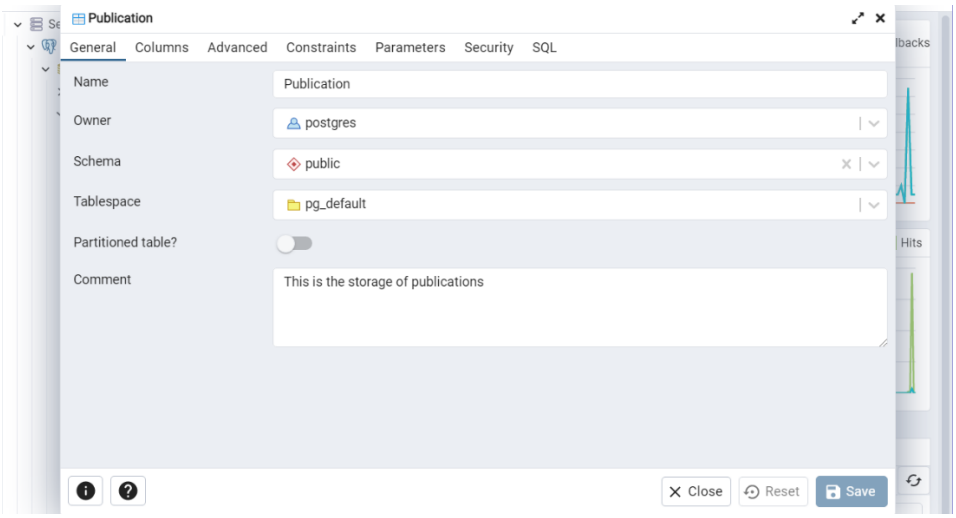

Рисунок 13 – Створення таблиці БД

Кожен стовбець задається окремо (рис. 14), при цьому слід вказати тип поля та обмеження, якщо в цьому є необхідність.

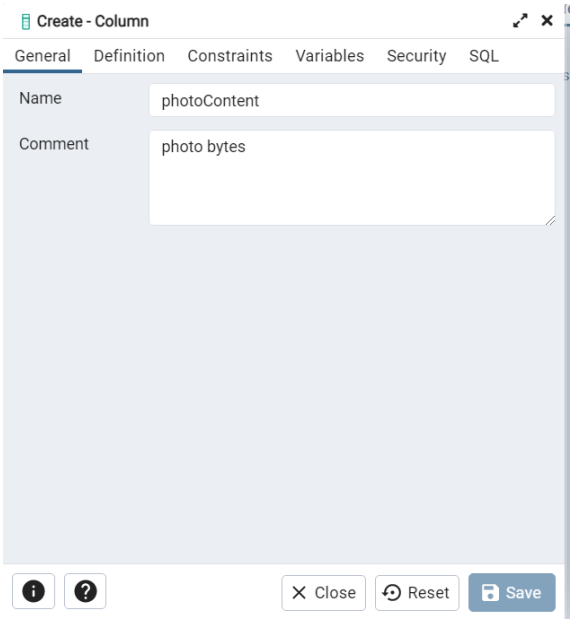

Рисунок 14 – Завдання стовпців для таблиці

Для роботи з БД було встановлено драйвер для роботи з PostgreSQL, а також ORM Hibernate – v5.2.12. ORM (англ. Object-relational mapping) – це відображення об'єктів будь-якої об'єктно-орієнтованої мови у структурі реляційних баз даних. Саме об'єктів, таких, які вони є, з усіма полями, значеннями, відносинами між собою.

ORM-рішенням для мови Java, є технологія Hibernate, яка не тільки дбає про зв'язок Java класів з таблицями бази даних (і типів даних Java в типи даних SQL), але також надає засоби для автоматичної побудови запитів та вилучення даних і може значно зменшити час розробки, яке зазвичай витрачається на ручне написання SQL та JDBC коду.

Hibernate генерує SQL виклики і звільняє розробника від ручної обробки результуючого набору даних та конвертації об'єктів, зберігаючи програму портується у всі SQL бази даних [15].

Наступний крок – налаштування Maven-файлу конфігурацій pom.xml (рис. 15):

```
<dependencies>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.2.12.Final</version>
    </dependency>
    <dependency>
        <groupId>com.google.code.gson</groupId>
        <artifactId>gson</artifactId>
        <version>2.9.0</version>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.3.3</version>
    </dependency>
```

Рисунок 15 – Налаштування Maven-файлу конфігурацій pom.xml

Це спеціальний XML-файл, який завжди зберігається в базовій директорії проекту та називається pom. xml. Файл POM містить інформацію про проект та різні деталі конфігурації, які використовуються Maven для створення проекту.

Цей файл також містить завдання та плагіни. Під час виконання завдань, Maven шукає файл pom.xml у базовій директорії проекту. Він читає його та отримує необхідну інформацію, після чого виконує завдання [16].

Перед тим, як створювати pom.xml, нам необхідно перш за все визначити групу проекту (groupId), його ім'я (artifactId) та його версію. Все це допоможе нам уніфікувати проект для простої його ідентифікації у репозиторії.

Також необхідно налаштувати деякі параметри для роботи з БД. Це стосується параметрів драйвера.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" version="2.1"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
      <persistence-unit name="publishingDb">
```

Тут йде опис БД:

```
<description>database for storing publishing
   data</description>
  <properties>

<property name="hibernate.dialect"
        value="org.hibernate.dialect.PostgreSQL94Dialect" />
    <property name="javax.persistence.jdbc.driver"
        value="org.postgresql.Driver" />
```

Посилання на нашу БД:

```
          <property name="javax.persistence.jdbc.url"
value="jdbc:postgresql://localhost:5432/postgres" />
```

Креди к БД:

```
    <property name="javax.persistence.jdbc.publishingUser"
              value="postgres" />

     <property name="javax.persistence.jdbc.password"
              value="128asdadXDS" />

      </properties>
  </persistence-unit>
</persistence>
```

3.2 Діаграма класів

Діаграма класів UML (рис. 17) – це графічне позначення, яке використовується для побудови та візуалізації об’єктно-орієнтованих систем.

Діаграма класів на уніфікованій мові моделювання (UML) – це тип діаграми статичної структури, яка описує структуру системи, показуючи:

– класи,

– їх атрибути,

– операції (або методи),

– і відносини між об'єктами.

Якщо використовується правильно, UML точно передає, як слід реалізувати код за допомогою діаграм. Якщо його точно інтерпретувати, реалізований код буде правильно відображати наміри дизайнера.

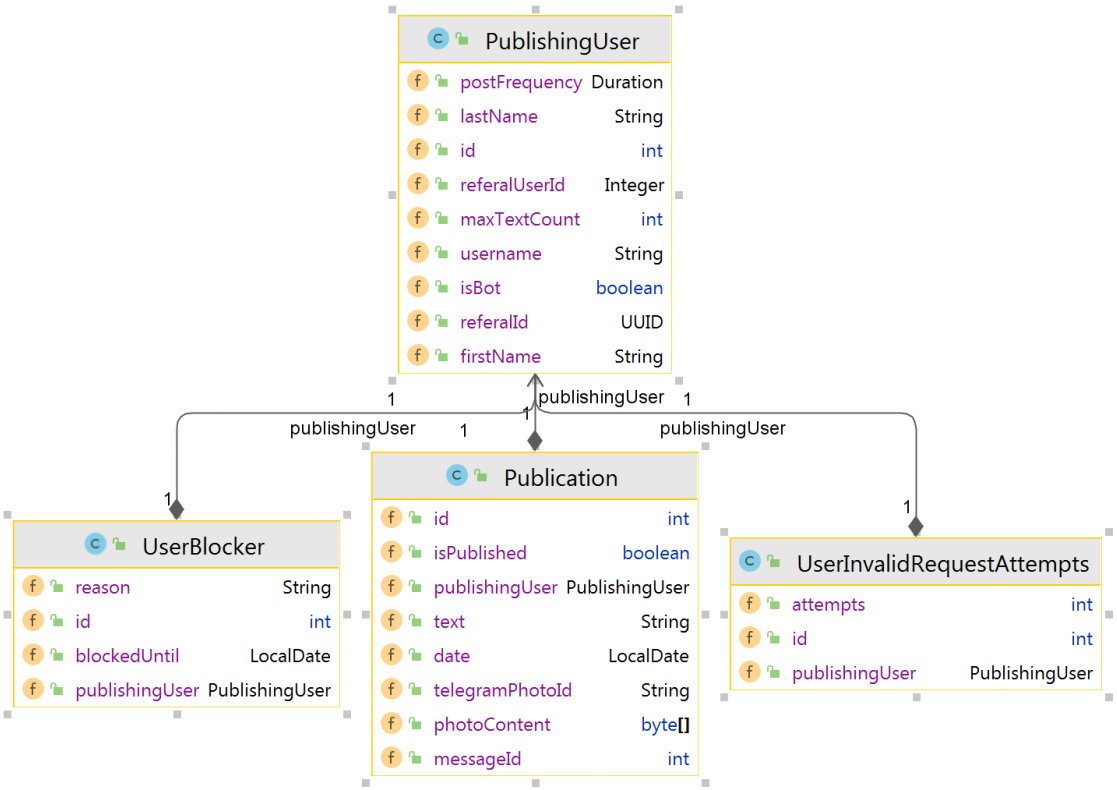

Рисунок 17 – Діаграма класів

Як серіалізатор для Json була взята бібліотека Gson, яка дозволяє робити багато – працювати з токенами джейсона, а також серіалізувати в рядок і десеріалізувати в об'єкт.

3.3 Опис алгоритму роботи бота

Телеграм надає кілька типів API. Саме у цьому проекті використовуватимемо API робота. Якщо коротко, то в проекті буде канал з публікаціями, де користувачі можуть залишати свої публикації, однак для цього вони повинні пройти валідацію за різними критеріями. Тому є бот, який одночасно присутній як на каналі, так і в приватному чаті з кожним учасником каналу, який хоче щось постити.

Основна ідея така – канал закритий для публікацій, крім адміну. Будь-хто хоче щось запостити, повинен спочатку створити приватний чат з ботом. Бот надасть докладну інструкцію щодо того, як зареєструватися.

Типів реєстрації є два – простий, просто подаємо заявку і бот приймає її, або можна зареєструватися по реферальному ВД. Це означає, що хтось уже з існуючих користувачів передав свій реферальний ВД і саме по ньому користувач реєструється. Це дає бонуси в обидві сторони.

Зареєструвавшись, користувач може надіслати свою посаду. Бот відразу починає його валідувати. На даний момент валідація стосується лише довжини тексту, частота публікацій, а також публікація звіряється з усіма публікаціями на подібність за рахунок алгоритму Дамеро-Лавештейн. Надалі можна використовувати сторонні утиліти для валідації контенту посту та зображення, прикріпленого до посту.

Якщо публікація не пройшла валідацію, робот докладно повідомить, що не так. Якщо ж валідація пройшла успішно, публікація автоматично буде відправлена ботом на канал, де інші учасники можуть відреагувати на повідомлення. Зараз оцінка якості посту будуватиметься на реакціях. Програмою вже передбачено читати коментарі під постом, надалі на їх основі буде побудований додатковий аналіз. Бонусами або просуванням у цій системі є підвищенням розміру тексту публікації та частоти публікацій.

Для роботи з Телеграм Бот АПІ було використано офіційну документацію Телеграма та не було використано жодної бібліотеки для роботи

з ним. Оф. бібліотек немає, а АПІ постійно оновлюється, тож краще самому зіпсувати цю справу.

Основою всіх повідомлень є модель «Message» (class Message).Ідентифікатор повідомлення. Він унікальний у межах чату.

```
@SerializedName("message_id")
public int messageId;
```

Користувач, від якого було надіслано листа та дата листа.

```
@SerializedName("from")
        public User from;
@SerializedName("date")
       public int date;
```

Сутність чату, з якого було надіслано листа:

```
@SerializedName("chat")
public Chat chat;
@SerializedName("forward_from")
```

Ця сутність відповідає за те, чиє листом було відцитовано:

```
public User forwardFrom;
@SerializedName("forward_date")
public int forwardDate;
@SerializedName("reply_to_message")
public Message replyToMessage;
@SerializedName("text")
```

Текст повідомлення

```
public String text;
@SerializedName("entities")
public MessageEntity[] entities;
```

Далі описуються сутності, що представляє відправлене в повідомленні аудіо та що представляє відправлені в повідомленні фото:

```
@SerializedName("audio")
public Audio audio;
@SerializedName("photo")
public PhotoSize[] photo;
```

Сутність, що представляє відправлене в повідомленні стікер:

```
@SerializedName("sticker")
public Sticker sticker;
@SerializedName("video")
public Video video;
@SerializedName("voice")
public Voice voice;
```

Якщо було надіслано повідомлення як пост, то тексту не буде, натомість прийде заголовок:

```
@SerializedName("caption")
public String caption;
@SerializedName("contact")
public Contact contact;
@SerializedName("location")
public Location location;
@SerializedName("venue")
public Venue venue;
@SerializedName("new_chat_member")
public User newChatMemeber;
@SerializedName("left_chat_member")
public User leftChatMemeber;
@SerializedName("new_chat_title")
public String newChatTitle;
```

Тепер розпишемо абстракіцю щодо роботи з Телеграмом. Для цього було створено класи «NonDataTelegramRequest» та «DataTelegramRequest», які представляють зручний інтерфейс для поділу запитів до телеграма. Перший не має параметрів, другий з параметром у вигляді моделі. Задається роут

(відносний шлях запиту) та вказується тип повернення - що запит вірить і у що серіалізувати:

```
    public class NonDataTelegramRequest<T_ResponseData>{
    public final String route;
    public final Class<T_ResponseData> resposeDataType;
    public         NonDataTelegramRequest(String         route,
Class<T_ResponseData> resposeDataType){
    this.route = route;
    this.resposeDataType = resposeDataType;}}
```

Цей клас розширив «NonDataTelegramRequest», додатково додавши «T_RequestData» для визначення вхідної моделі:

```
    public   final   class   DataTelegramRequest<T_RequestData,
T_ResponseData>
    extends NonDataTelegramRequest<T_ResponseData>{
    public DataTelegramRequest(String route,
Class<T_ResponseData> resposeDataType){
    super(route, resposeDataType);}}
```

І потім зручно створюємо набір статичних константих об'єктів цих двох класів і задаємо всі необхідні функції роботи з телеграмом. Для отримання оновлень із телеграма використовуємо метод «NonDataTelegramRequest», а для запису файлу «DataTelegramRequest»:

```
    public class TelegramRequests{
    public final static NonDataTelegramRequest<Update[]>
GET_UPDATES = New NonDataTelegramRequest<>("getUpdates",
Update[].class);
    public final static DataTelegramRequest<FileId, File>
GET_FILE = New DataTelegramRequest<>("getFile", File.class);
```

Наступні два метода необхідні для надсилання текстового повідомлення та повідомлення з фото(для постів):

```
    public final static DataTelegramRequest<SendTextMessage,
Message> SEND_TEXT_MESSAGE = New
DataTelegramRequest<>("sendMessage", Message.class);
```

```
    public final static DataTelegramRequest<SendPhotoMessage,
Message> SEND_PHOTO_MESSAGE = New
DataTelegramRequest<>("sendPhoto", Message.class);
```

Для отримання реакцій на заданому пості:

```
    public final static DataTelegramRequest<GetMessageReaction,
MessageReactionType[]> GET_MESSAGE_REACTION = новий
DataTelegramRequest<>("getReaction",
MessageReactionType[].class);}
```

```
    MessageReactionType є енам з таким набором значеньpublic
enum MessageReactionType{ SMILE, GOOD, BAD, FIRE, HEART}
```

3.3.1 Клас «TelegramServer»

Тепер опишемо код класу «TelegramServer», який займається відправкою запитів у телеграм. Почнемо з двох основних методів з надсилання запитів по Http.

```
    private static String executePostString(String targetURL){
    HttpURLConnection connection = null;
    try{
```

Створюємо об'єкт URL та відкриємо зв'язок. Далі необхідно встановити метод POST та тип контенту як json:

```
    URL url = new URL(targetURL);
    connection = (HttpURLConnection) url.openConnection();
    connection.setRequestMethod("POST");
    connection.setRequestProperty("Content-Type",
                                  "application/json");
    connection.setRequestProperty("Content-Length",
                                  Integer.toString(0));
    connection.setRequestProperty("Content-Language", "en-US");
    connection.setUseCaches(false);
```

Встановлюємо параметр читання результату true, щоб читати відповідь. Після цього слід витягнути з коннекшина стриму для читання та прочитати відповідь. Буфферізацію виконуємо через білдер рядків:

```
connection.setDoOutput(true);
     InputStream is = connection.getInputStream();
     BufferedReader rd = New BufferedReader(New
                                    InputStreamReader(is));
    StringBuilder response = новий StringBuilder();
   String line;
```

Читаємо стрім построчно і записуємо в білдер. Обовязково перевіряємо та виводимо результат.

```
while ((line = rd.readLine()) != null){
response.append(line);
response.append('\r');}
rd.close();
return response.toString();}
catch (Exception exc) {
exc.printStackTrace();
return null;
}
  finally{
       if (connection! = null) {
    connection.disconnect();}}}
```

3.3.2 Базові функції

«sendTextMessage» – служить для надсилання повідомлення (спрощена модель). Спочатку створюється модель з даними для відправлення повідомлення. Далі задається текст та ВД чату. Після цього йде виклик більш базового методу з прокидуванням моделі:

```
public static Message sendTextMessage(BotData data, String
                                    text, long chatId){
SendTextMessage sendTextMessage = новий SendTextMessage();
sendTextMessage.text=text;
sendTextMessage.chatId = chatId;
return sendRequest(
```

```
data,
TelegramRequests.SEND_TEXT_MESSAGE,
sendTextMessage);}
```

Базова функція для надсилання запиту без моделі:

```
public static <T_ResponseData> T_ResponseData sendRequest(
```

Даний клас містить дані про боті, такі як токена бота, сутність запиту. За її допомогою Будуємо посилання та надсилаємо запит. За документацією це «базове посилання телеграма + токен бота + відносний роут»:

```
BotData data,
NonDataTelegramRequest<T_ResponseData> request){
try{
String result = executePostString("https://api.telegram.org/bot"
+ data.token + "/" + request.route);
```

Просимо та повертаємо результат за допомогою gson

```
JsonObject jObject = new
JsonParser().parse(result).getAsJsonObject();
Gson gson = new Gson ();
return gson.fromJson(jObject.get("result"),
request.resposeDataType);}
catch (Exception exc) {
throw new RuntimeException(exc);}}
```

Наступна функція необхідна для надсилання запиту на стрибку файлів:

```
public static byte[] getFile(BotData data, String path){

    try{
    return executePostBytes("https://api.telegram.org/file/bot"
    + data.token + "/" + path);}

    catch (Exception exc) {

    throw new RuntimeException(exc);}}
```

Схема класів Telegram API модуля представлено на рисунку 18:

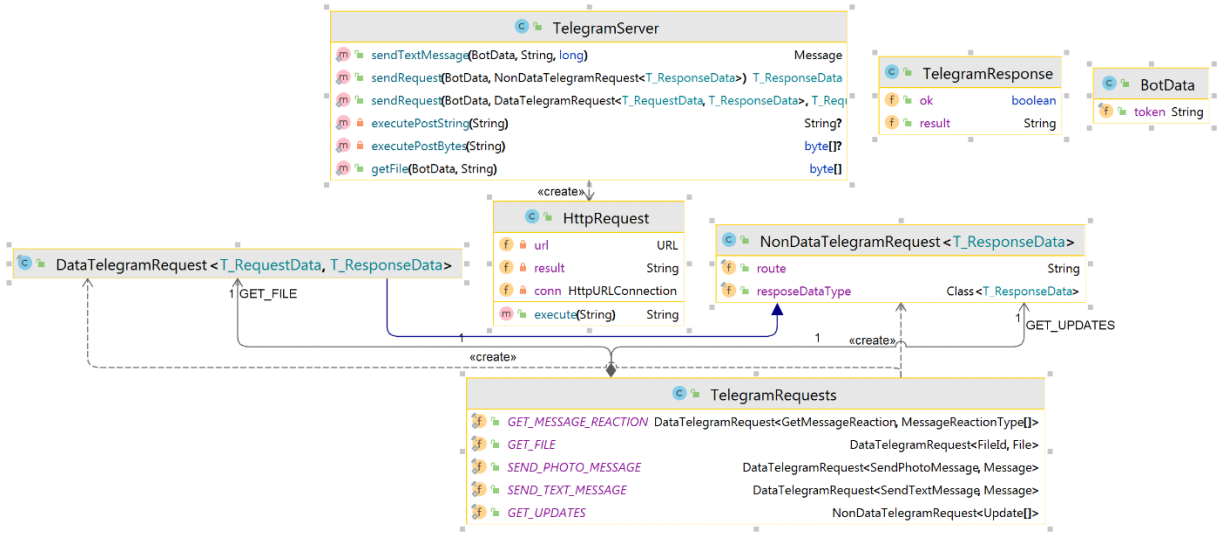

Рисунок 18 – Класи Telegram API модуля

Скорочена схема взаємодії API і моделей представлено на рисунку 19:

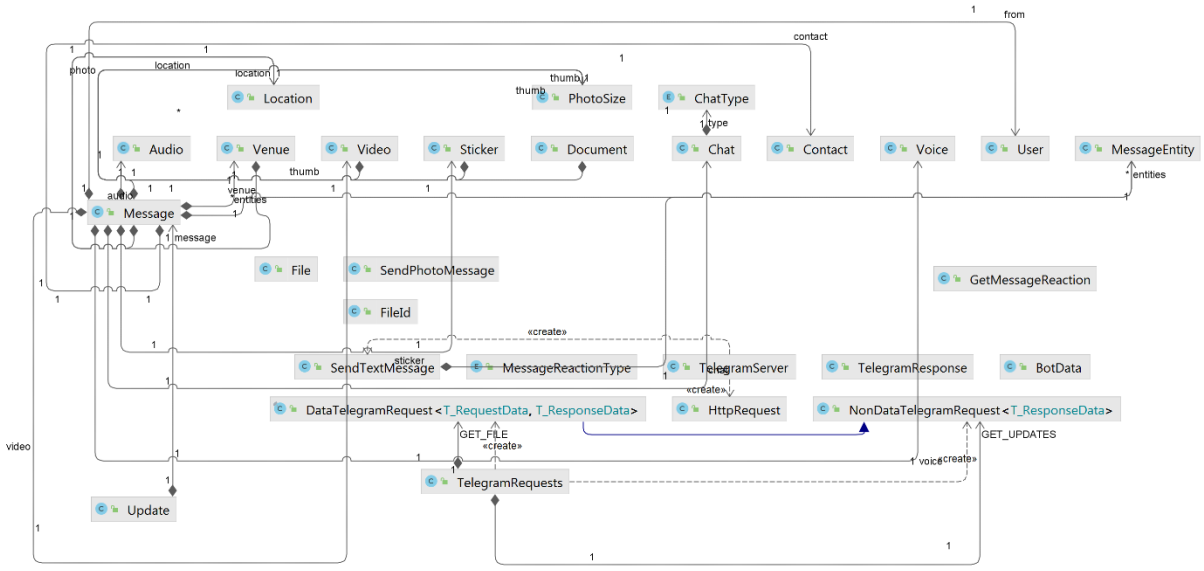

Рисунок 19 – Скорочена модель

Діаграма класів моделей Telegram API представлено на рисунку 20:

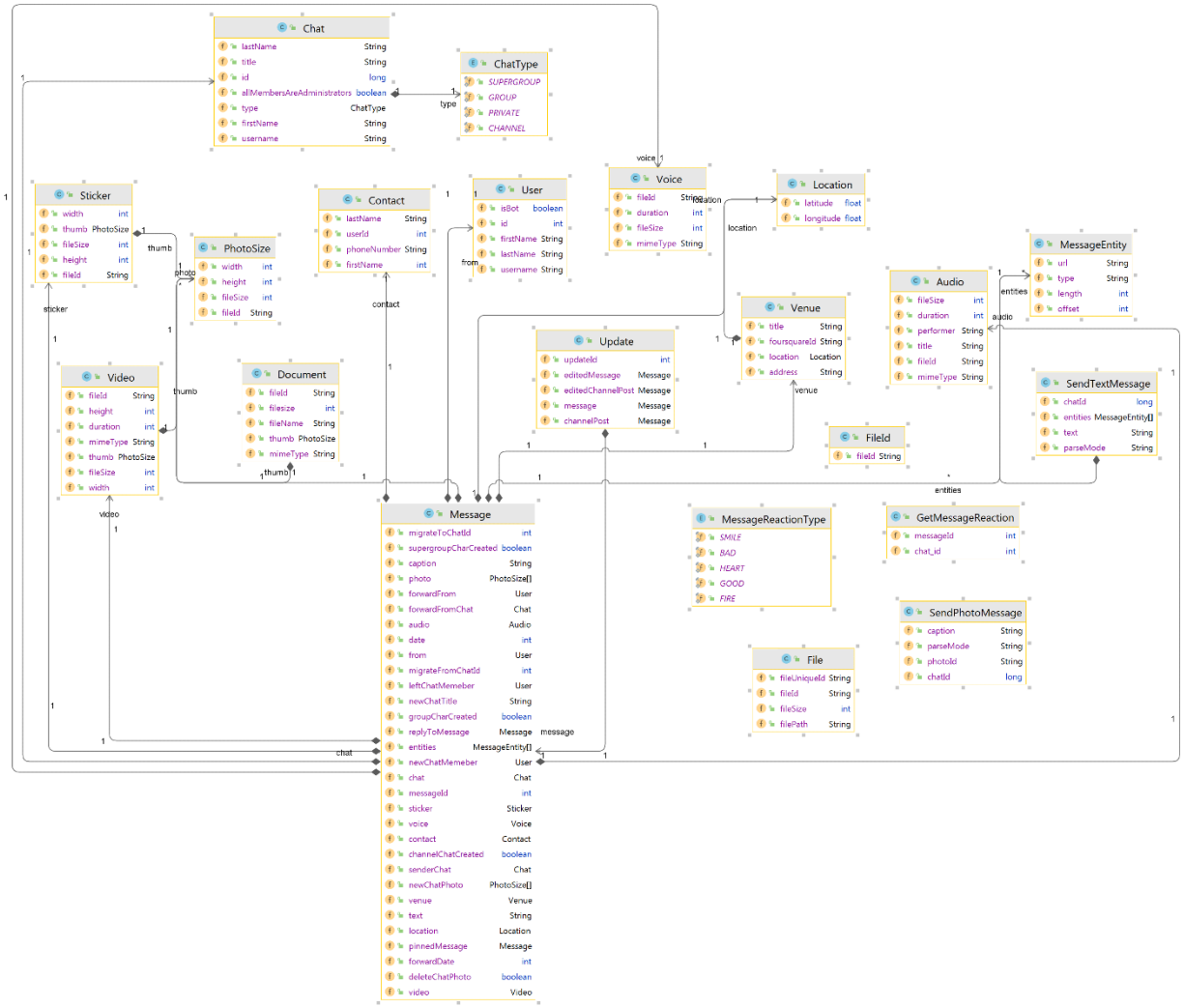

Рисунок 20 – Повна діаграма класів

Всього у системі 21 клас. Основний клас «Message» містить поля для зберігання інформації щодо публікованих постів, та методи для обробки надходжених повідомлень.

3.4 Налаштування Клієнту

Створюємо канал с назвою «Publications» та налаштовуємо його як приватний (рис. 21):

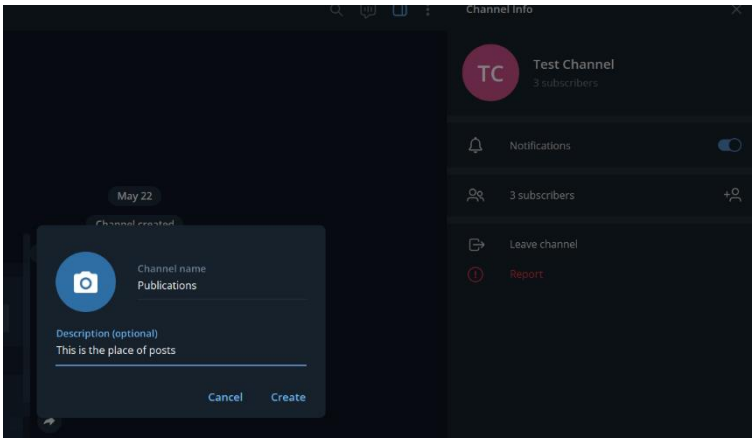

Рисунок 21 – Створення нового каналу

Наступний крок – це додавання чату з BotFather для створення та налаштування нашого боту (рис. 22):

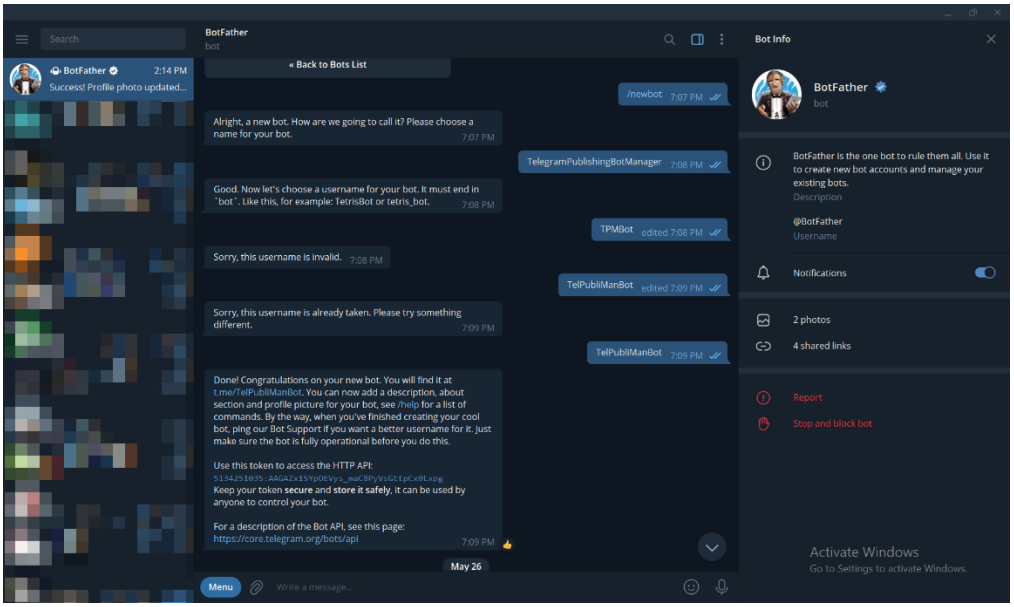

Рисунок 22 – Додавання чату з BotFather

Крім цього, слід додати фото для бота (рис.23) та створюємо групу обговорень для постів у каналі (рис. 24).

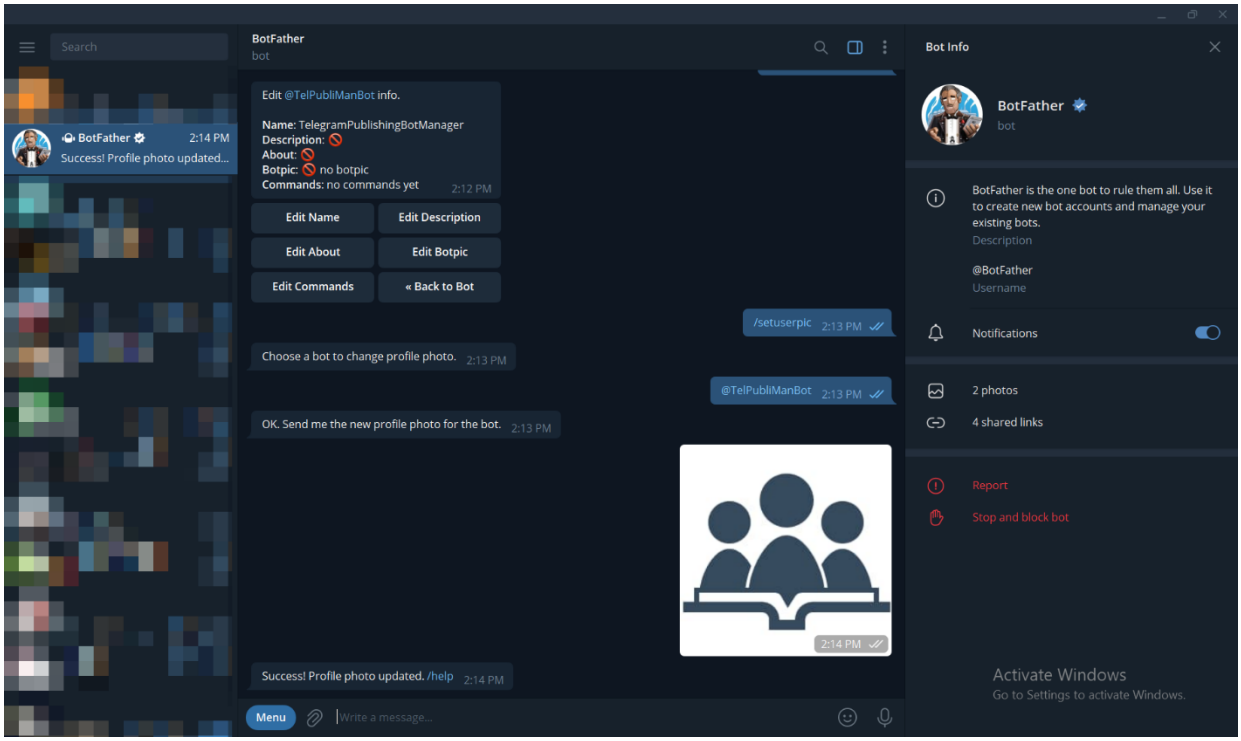

Рисунок 23 – Відображення фото боту у чаті

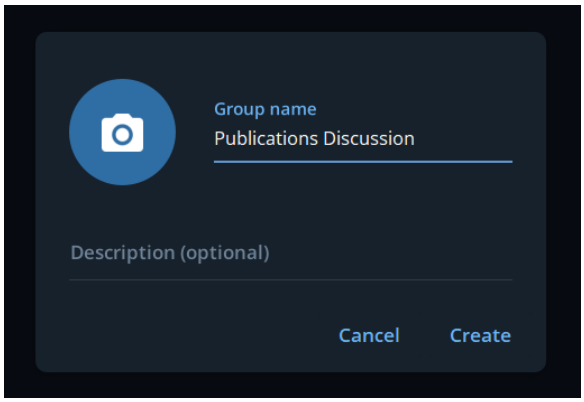

Рисунок 24 – Створення нової групи

У вікні для налаштувань каналу слід підключити тільки що створену групу (рис. 25).

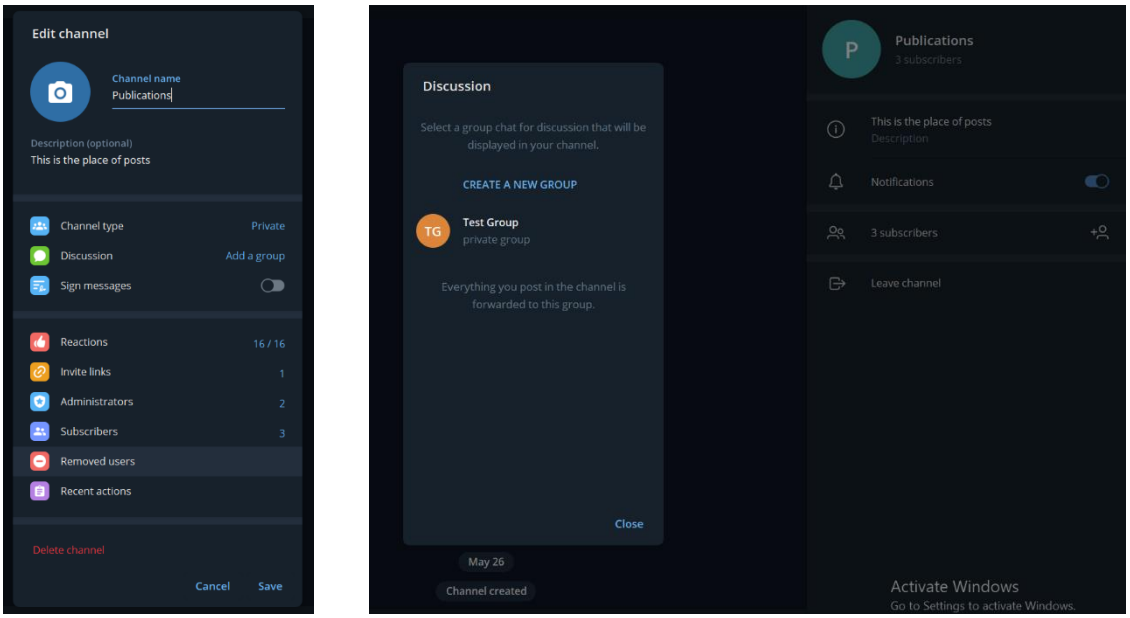

Рисунок 25 – Підключення групи до каналу

3.5 Основна логіка боту

Розглянемо основну логіку роботи робота (клас «Bot»). В ньому створюємо дані бота (токен), ID каналу для публікацій, карту ІД оброблених апдейтів та масив команд бота.

```
public class Bot{
public final BotData data = New
BotData("5134251035:AAGAZx15YpOEVys_maC8PyVsGttpCx0Lxpg");

private final int publicationsChannelId = -1072489184;

private final HashMap<Integer, LocalDate> handledUpdateIds =
new HashMap<>();
private final IPrivateCommand[] privateCommands = new
IPrivateCommand[]{
New OnStartCommunation(),
new RegisterUser()};
```

Далі використовуємо бескінцевий цикл для запиту телеграм на пошук апдейтів та опрацьовуємо кожен апдейт:

```
public void init() throws Exception{
while (true) {
Update[] updates = TelegramServer.sendRequest(data,
TelegramRequests.GET_UPDATES);
```

Якщо Ід вже опрацьовано, тоді оновлюємо дату по даному Ід. Опрацьовуємо апдейт та додаємо Ід у карту і до поточної дати додаємо 1 день, для того щоб автоматично видаляти оброблені повідомлення через 1 день.

```
for (Update update: updates){

if (handledUpdateIds.containsKey(update.updateId)){
handledUpdateIds.replace(update.updateId, LocalDate.now());
continue;}
else {
handleUpdate(update);
handledUpdateIds.put(update.updateId,
LocalDate.now().plusDays(1));}}
```

Прибираємо непотрібні рекорди з ІД ардейтів і очікуємо 10 секунд:

```
clearRedundantUpdateIds();
Thread.sleep(10000);}}
```

Вираховуємо поточну годину, створюємо масив Ід для видалення та обходимо свою карту. Якщо поточна дата більше ніж вказана в карті, то такий апдейт застарів й має бути видалень. У такому разі видаляються з мапі всі апдейти. Дивлячись на те що саме прийшло в апдейті вибираємо стратегію опрацювання апдейту.

Обов’язково перевіряємо чи це пост у нашому каналі. Якщо текст починається зі '/' значить це відноситься до команд, інакше сприймаємо сповіщення як пост. Слід обходити всі команди й намагатись виконати одну з них.

Якщо жадної команди не знайшлось, то відкриваємо доступ до репозиторію юзерів. Тут необхідно знайти його по ІД з телеграму. У разі відсутності юзера, виводимо сповіщення, що такого юзера немає. Далі йде

валідація поста юзера та пошук його останньої публікації. Дозвіл на публікацію дається у разі, якщо «дата останньої публікації + частота публікацій» юзера перекриває сьогоднішню дату.

Створюємо й зберігаємо в Бд публікацію та відправляємо сповіщення про те що публікація успішно оброблена. Після цього публікація йде в канал.

Функція для валідації контенту публікації «validateMessagePost»: спершу перевіряємо юзера на валідність, отримуємо кількість невірних спроб публікації

```
private void validateMessagePost(PublishingUser
publishingUser, Message message) throws IOException{
     UserValidator userValidator = new
UserValidator(message.from);
     try (Repository<UserInvalidRequestAttempts>
userAttemptsRepository =
DbConnections.USER_INVALID_REQUEST_ATTEMPTS.open()){

          userValidator.validateUser();

          UserInvalidRequestAttempts userAttempts =
userAttemptsRepository.getEntities()
               .stream()
               .filter((ua) -> ua.userId ==
publishingUser.id)
               .findFirst()
               .orElse(null);
```

Якщо кількість спроб рівна 5, знаходимо блокер юзера. якщо такого блокера немає, то створюємо:

```
     if (userAttempts != null && userAttempts.attempts == 5){
          try  (Repository<UserBlocker>  userBlockerRepository  =
DbConnections.USER_BLOCKERS.open()){

     UserBlocker userBlocker =
userBlockerRepository.getEntities().stream().filter((ub) ->
ub.userId == publishingUser.id)

     if (userBlocker == null){
          userBlocker = new UserBlocker();
          userBlocker.userId = publishingUser.id;
        userBlocker.blockedUntil = LocalDate.now().plusMonths(1);
```

```
     userBlocker.reason = "Invalid posts total attempts count has
reached"
     userBlockerRepository.add(userBlocker);}
     LocalDate blockedUntil = userBlocker.blockedUntil;
```

Віправляємо повідомлення про те що юзер заблокований й треба чекати. Якщо ж все таки кількість спроб меньше, або взагалі не було, то перевіряємо що картинка є у публікації. Також перевіряємо що є текст і розмір тексту не перевищено по заданому юзеру. Намагаємось знайти публікацію яка хоча б трохи подібна до нової. У разі якщо така публікація знайдена, тоді в залежності чи належить ця публікація заданому юзеру чи ні відправляється відповідне повідомлення.

Відкриваємо доступ до таблиці помилок юзера та шукаємо сутність в БД по юзеру. Якщо такої сутності немає, то створюємо

```
          try (Repository<UserInvalidRequestAttempts> repository
     = DbConnections.USER_INVALID_REQUEST_ATTEMPTS.open()){
               UserInvalidRequestAttempts userAttempts =
     repository.getEntities()
                    .stream()
                    .filter((ua) -> ua.userId ==
     publishingUser.id)
                    .findFirst()
                    .orElse(null);
               if (userAttempts == null){
                    userAttempts = new
     UserInvalidRequestAttempts();
                    repository.add(userAttempts);}
```

Збільшуємо лічильник невірних спроб та відправляємо повідомлення юзеру про те, що він відправив щось невірне й вказується яка кількість помилок в нього ще залишилось.

Розглянемо кілька прикладів команд. Команда для отримання всіх публікацію по заданому користувачеві:

```
public class GetUsersPublications implements
IPrivateCommand{
@Override
```

```
public boolean tryHandle(BotData data, Message message)
throws IOException{
```

Перевіряємо чи текст рівний «/getPublications» та шукаємо всі публікації за заданим користувачем. Якщо публікацій немає, то виводи сповіщення про те що немає публікацій.

```
 if ( ! message.text.equalsIgnoreCase("/getPublications"))
return false;
UserValidator userValidator = New
UserValidator(message.from);
try{
PublishingUser publishingUser=userValidator.validateUser();
try (Repository<Publication> publicationRepository =
DbConnections.PUBLICATIONS.open()){
Publication[] publications = publicationRepository
.getEntities()
.stream()
.filter((p) -> p.userId == publishingUser.id)
.toArray(Publication[]::new);
if (publications.length == 0)
TelegramServer.sendTextMessage(data, "You haven't any
publication yet", message.chat.id);
```

Команда для початку комунікації з ботом:

```
public class OnStartCommunation implements IPrivateCommand{
public boolean tryHandle(BotData data, Message message)
throws IOException{
```

Якщо текст рівний /start, намагаємось дістати по Ід користувача. Якщо такого користувача немає – Відправляємо сповіщення користувачеві в приватний чат про те як зареєструватися.

```
if (message.text.equalsIgnoreCase("/start")){
try (Repository<PublishingUser> userRepository =
DbConnections.USERS.open()){
PublishingUser user = userRepository.findFirst((pu) -> pu.id
== message.from.id);
```

```
if (user == null){

SendTextMessage sendTextMessage = новий SendTextMessage();

sendTextMessage.text = "Якщо ви маєте подібний ID,
скористайтеся реєстратором, використовуючи цей
формат:\n/register:<referalId>\nIf ти не будеш ні, ні
register як that:\n/register:none" ;

sendTextMessage.chatId = message.chat.id;

TelegramServer.sendRequest(data,TelegramRequests.SEND_TEXT_M
ESSAGE,sendTextMessage);}
```

Діаграму класів ядра та спрощену діаграму взаємодії ядра, команд і моделей представлено на рисунку 26:

Рисунок 26 – Діаграму класів ядра

3.6 Приклад роботи боту

Розглянемо приклад реєстрації нового юзеру та відправку повідомлень, як показано на рисунку 27:

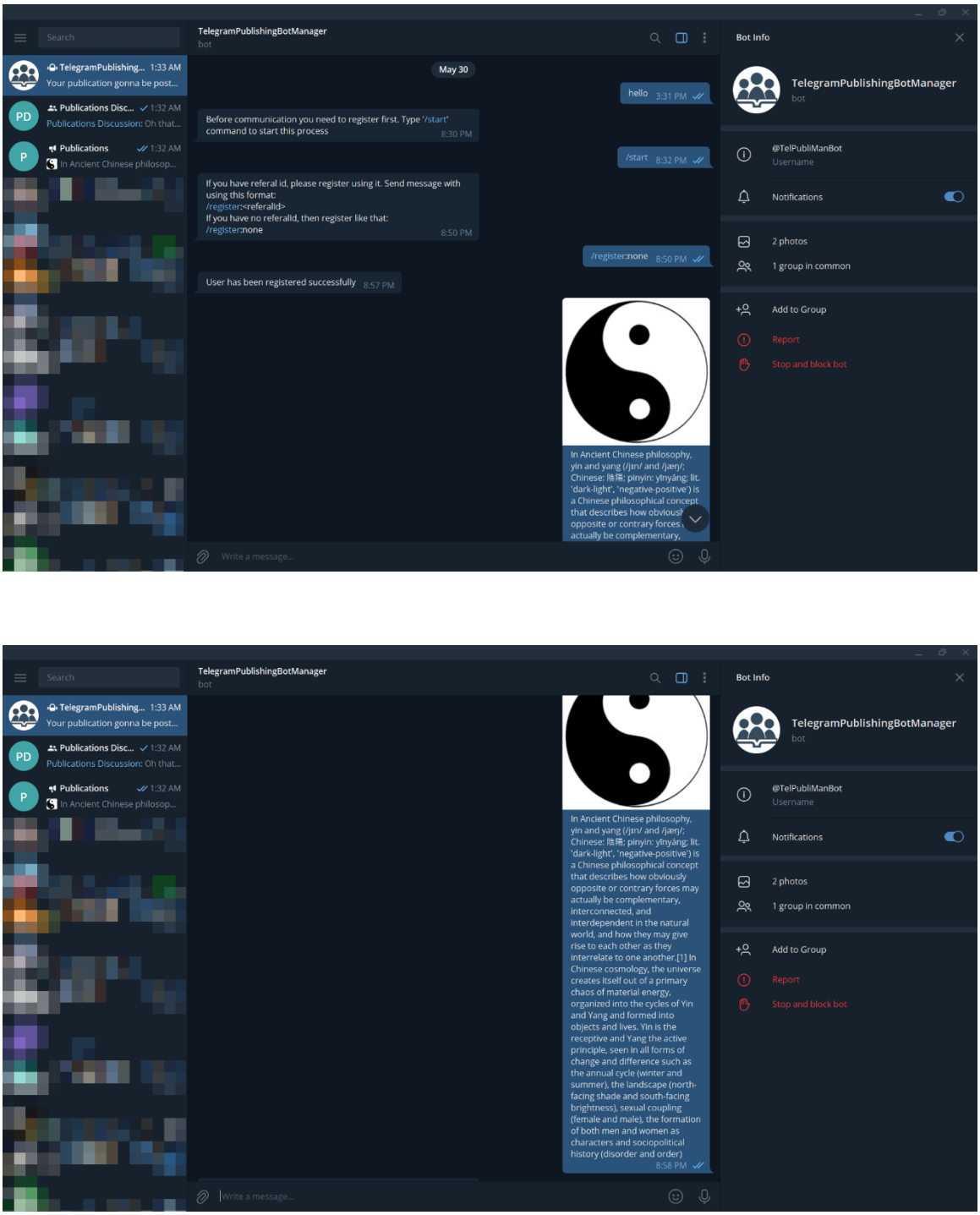

Рисунок 27 – Приклад роботи бота

Якщо користувач оформив новий пост та відправив його на публікацію, пост перевіряється на валідацію ботом. У разі успішної перевірки пост з’явиться у чаті (рис. 28). Якщо перевірку не пройдено, наприклад з причини превищення кількості символів у пості, бот відправляє повідомлення у приватний чат з користувачем (рис. 29).

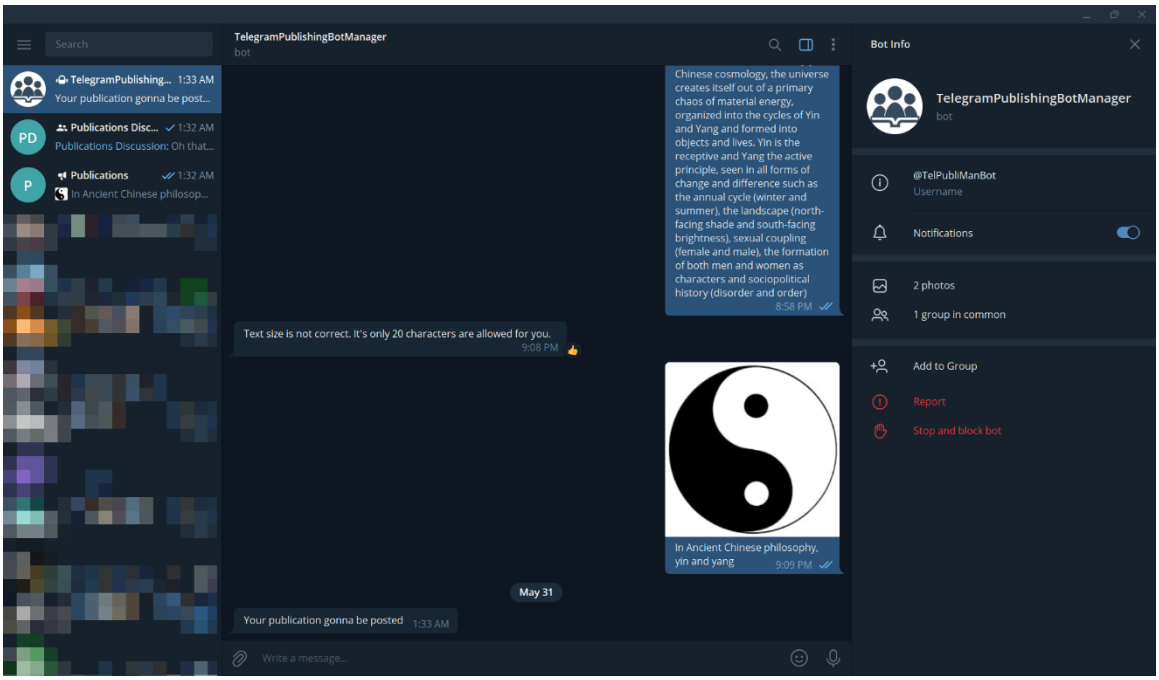

Рисунок 28 – Приклад процесу перевірки нового поста

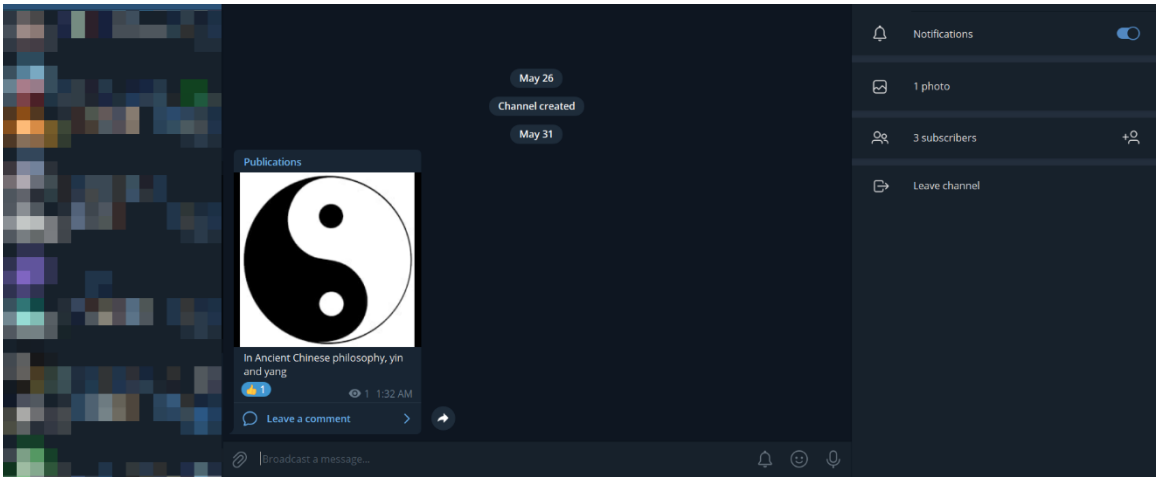

Рисунок 29 – Скрін приватного чату з ботом

Приклад обговорення нового посту у чаті, та накопичування реакцій від читачів представлено на рисунку 30:

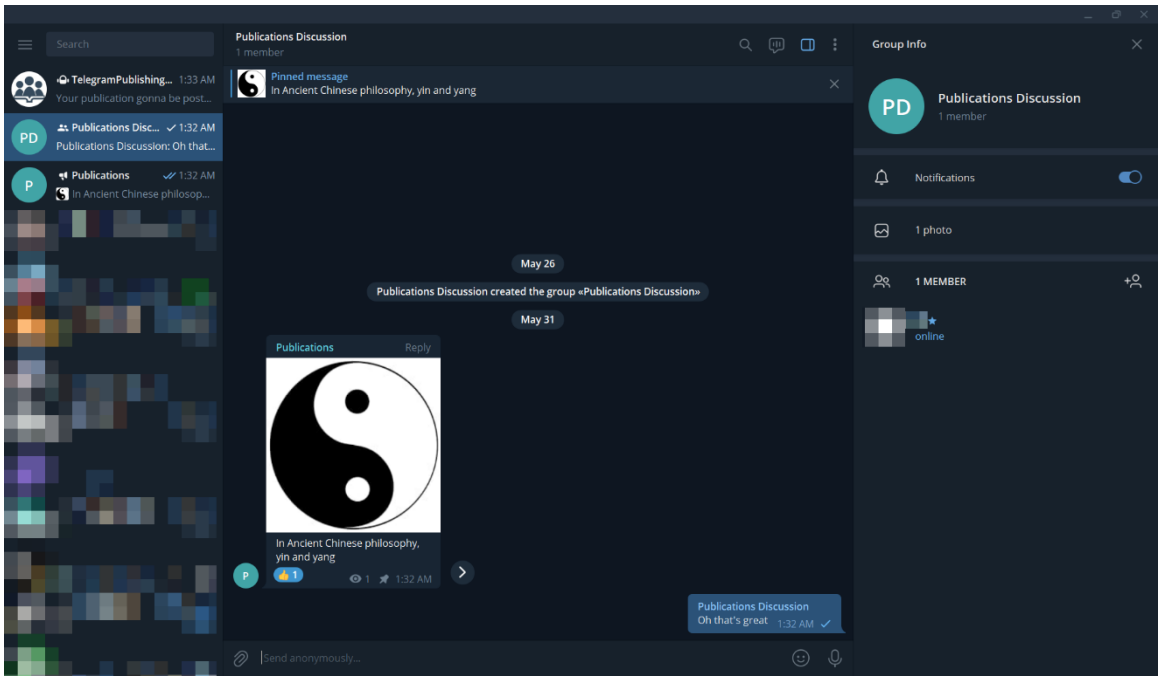

Рисунок 29 – Приклад активного посту

ВИСНОВКИ

Чат-боти не замінять приватне спілкування автора з читачами, вміст домашньої сторінки, електронні листи, які читачі погодилися отримувати, або навіть комунікації в соціальних мережах. Є багато причин, чому видавцям слід мати всі ці канали та піклуватися про них. Однак технологія чат-ботів унікальна і має можливість допомогти видавцям взаємодіяти зі своїми читачами іншим, новим способом.

Вони не просто інструмент передачі інформації. Чат-боти, по суті, є інструментом розмови, який використовує знання про потреби та ідеї клієнтів. Мистецтво — відповідати на них вчасно, наскільки це можливо. Це веде до створення більш привабливого досвіду, який збільшує ймовірність того, що люди натискатимуть, коли ви цього бажаєте.

Чат-боти – це тренд, який буде розвиватися в майбутньому разом із наданням ще більш персоналізованого досвіду читання.

У ході виконання дипломного проекту реалізовано чат-бот для каналу телеграм. В його логіку додано алгоритми валідації текстової інформації. Також, реалізована БД для користувачів для контролю їх активності у чаті публікацій.

Під час проектування виконано аналітичний огляд предметної області та обрані програмні засоби розробки. На етапі моделювання побудовані UML-діаграми варіантів використання та активності. Діаграми класів отримані у результаті сборки повного проекту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. What are bots used for in the publishing industry? URL: https://www.stateofdigitalpublishing.com/digital-platform-tools/bots-in-publishing/ (дата звернення 06.05.2021).
2. Сколько стоит чат-бот и в чем его преимущества для бизнеса. URL: https://www.unisender.com/ru/blog/kuhnya/chat-boty-vnedrenie/ (дата звернення 06.05.2021).
3. What is Digital Publishing? Short Guide. URL: https://publuu.com/knowledge-base/what-is-digital-publishing-short-guide/ (дата звернення 06.05.2021).
4. Epic Reads. URL: https://massively.ai/case-studies-2/epicreads/ (дата звернення 10.05.2021).
5. Pan Macmillan Chatbot. URL: https://www.digiteum.com/portfolio-panmacmillan-chatbot/ (дата звернення 11.05.2021).
6. 13 крутых онлайн-сервисов для создания чат-ботов. URL: https://habr.com/ru/company/click/blog/567446/ (дата звернення 13.05.2021).
7. Bots: An introduction for developers. URL: https://core.telegram.org/bots#inline-mode (дата звернення 14.05.2021).
8. Telegram API: наглядный разбор с примерами. URL: https://highload.today/telegram-api/ (дата звернення 16.05.2021).
9. Gson – Краткое руководство. URL: https://coderlessons.com/tutorials/java-tekhnologii/vyuchit-gson/gson-kratkoe-rukovodstvo (дата звернення 18.05.2021).
10. Обзор Gson – работаем с JSON в Java. URL: http://www.javenue.info/post/gson-json-api (дата звернення 22.05.2021).

11. What is Use Case Diagram? URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата звернення 23.05.2021).
12. UML – Activity Diagrams. URL: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm (дата звернення 23.05.2021).
13. ADVANTAGES OF POSTGRESQL. URL: https://www.cybertec-postgresql.com/en/postgresql-overview/advantages-of-postgresql/ (дата звернення 25.05.2021).
14. pgAdmin 4. URL: https://filetogo.net/app/pgadmin-4 (дата звернення 27.05.2021).
15. Основы Hibernate. URL: https://habr.com/ru/post/29694/ (дата звернення 27.05.2021).
16. Руководство по Maven. POM. URL: https://proselyte.net/tutorials/maven/pom (дата звернення 278.05.2021).