МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА МЕРЕЖЕВИХ ТА ІНТЕРНЕТ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
ЗАВІДУВАЧ КАФЕДРИ
МЕРЕЖЕВИХ ТА ІНТЕРНЕТ ТЕХНОЛОГІЙ
_____ Юрій КРАВЧЕНКО
«_____» _____2022 РОКУ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

галузі знань 17 «Електроніка та телекомунікації»
за спеціальністю 172 «Телекомунікації та радіотехніка»
освітньо-професійна програма «Мережеві та інтернет технологій»

на тему:

РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ РОБОТИ З ДОВІДКОВИМИ ДЖЕРЕЛАМИ НА ОСНОВІ LOW-CODE ПЛАТФОРМ

Виконав: студент групи МІТ - 41
Каріна ПІВТОРАК
_____ _____
(ім'я та ПРІЗВИЩЕ) (підпис)

Керівник: асистент кафедри мережевих та інтернет технологій
(посада)
к.т.н., асистент Олена СТАРКОВА
_____ _____
(науковий ступень, вчене звання, ім'я та ПРІЗВИЩЕ) (підпис)

Київ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА МЕРЕЖЕВИХ ТА ІНТЕРНЕТ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
ЗАВІДУВАЧ КАФЕДРИ
МЕРЕЖЕВИХ ТА ІНТЕРНЕТ ТЕХНОЛОГІЙ
_____ Юрій КРАВЧЕНКО
«_____» _____2022 РОКУ

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ

Здобувач вищої освіти: Півторак Каріна Сергіївна

1. Тема роботи: розробка інформаційної системи для автоматизації роботи з довідковими джерелами з використанням low-code платформ затверджена на засіданні кафедри МІТ «24» ___грудня_____ 2021 р. протокол № 8
2. Термін здачі закінченої роботи «30» травня 2022 р.
3. Вихідні дані до проекту (роботи)
   Системи для автоматизації роботи з довідковими джерелами
   Проаналізувати сучасні шляхи та методи проектування інформаційних систем
   Розробити інформаційну систему на базі чат-бота з використанням Low-Code технологій
4. Зміст пояснювальної записки (перелік питань, що їх потрібно розробити, обсяг – 35-50 стор.): Вступ, пошук та аналіз технологій для реалізації, опис основних функціональних можливостей платформи corezoid, розробка інформаційної системи, висновки, додатки
5. Перелік графічного матеріалу 8-10 слайдів: загальна класифікація мов програмування, модель використання платформи Corezoid, Corezoid Domain Model, структура процесів системи, важливі частини коду, розгорнуті частини системи на Google Cloud Platform, знімки екрану з готовою логікою.

Дата видачі завдання:

Керівник роботи:_____к.т.н., асистент Олена СТАРКОВА___
(підпис) (посада, ім`я, ПРІЗВИЩЕ)

Завдання прийняв до виконання_____ Каріна ПІВТОРАК_____
(підпис) (ім'я, ПРІЗВИЩЕ)

КАЛЕНДАРНИЙ ПЛАН ВИКОНАННЯ РОБОТИ

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Підготовчий	10.02.2022	
2	Розділ 1	06.03.2022	
3	Розділ 2	04.04.2022	
4	Розділ 3	08.05.2022	
5	Доповідь та слайди		
6	Пояснювальна записка	15.05.2022	

Здобувач вищої освіти _____Каріна ПІВТОРАК_____

(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник: _____Олена СТАРКОВА_____

(підпис) (ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Розробка інформаційної системи для автоматизації роботи з довідковими джерелами на основі low-code платформ» складається зі вступу, основної частини, що містить 3 розділи, висновків, додатків, списку літератури та джерел.

Загальний обсяг роботи – **66** сторінки. Робота містить 36 рисунків, 0 таблиць, 3 додатка. Список використаних джерел включає 25 джерел.

Об’єкт дослідження – система керування чат-ботом на базі Low-Code.

Мета роботи – розкрити поняття Low-Code платформи, дослідити основи їх проектування, переваги, недоліки та особливості, проаналізувати існуючі платформи, створити інформаційну систему для пошуку, збереження та обробки інформації на базі Low-Code обраної платформи Corezoid.

Предмет дослідження – Low-Code платформи, чат-бот, його функціонування, робота з даними та інтеграція Low-Code платформи із базою даних та зовнішніми API.

Методи дослідження – системний аналіз, евристичний та теоретичний аналіз, методи програмної інженерії, компонентне проектування, UML-проектування.

В ході роботи проведено аналіз технологій для розробки інформаційних систем. Запропоновано оптимальний метод для реалізації. Побудовано інформаційну систему на базі Corezoid.

Практичне значення роботи полягає у розробці інформаційної системи на базі low-code платформи Corezoid.

Результати здійснених у дипломному проекті досліджень можуть бути використані для створення подібних систем.

Ключові слова: Чат-бот, API, Інформаційна система, Corezoid, Low-Code, Mongodb, CRUD операції, Telegram, Viber.

ABSTRACT

Explanatory note to the thesis "Development of information system for automation of reference sources based on low-code platforms" consists of an introduction, the main part containing 3 sections, conclusions, appendices, bibliography and sources.

The total volume of the work is 54 pages. The work contains 36 figures, 0 tables, 3 appendices. The list of used sources includes 25 sources.

The object of research is chatbot technologies and properties of Low-Code platforms.

The purpose of the work is to reveal the basic concepts and types of Low-Code platforms, explore the basics of design, advantages, disadvantages and features, create an information system for searching, storing and processing information based on Low-Code platform Corezoid.

The subject of research is a chat-bot control system based on Low-Code.

Research methods - systems analysis, heuristic and theoretical analysis, software engineering methods, component design, UML design.

In the course of work the analysis of technologies for development of information systems is carried out. The optimal method for implementation is offered. An information system based on Corezoid was built.

The practical significance of the work is to develop an information system based on the low-code platform Corezoid.

The results of the research carried out in the diploma project can be used to create similar systems.

Keywords: Chatbot, API, Information System, Corezoid, Low-Code, Mongodb, CRUD operations, Telegram, Viber.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

APEX – Oracle Application Express

API – Application Programming Interface

APaas – Application platform as a Service

CRM – Customer relationship management

HTML – HyperText Markup Language, мова розмітки гіпертексту

HTTP - Hypertext Transfer Protocol

JSON – JavaScript Object Notation

MVP – Minimum viable product, мінімально життєздатний продукт

PaaS – Platform as a Service, платформа як послуга

RAD – Rapid application development, швидка́ розро́бка застосу́нків

ROI – Return on Investment, або окупність, рентабельність інвестицій

SQL – Structured query language, мова структурованих запитів

YAML – Yet Another Markup Language («Ще одна мова розмітки»)

БД – база даних

ПЗ – програмне забезпечення

СУБД – система управління базами даних

ВСТУП

У 50–60 роки минулого століття експоненційне зростання потоку публікацій отримало назву «інформаційний вибух». Кількість інформації збільшується за рахунок її дублювання, ущільнення (кумулятивності), перенесення в документи, що випускаються знову (статті, огляди, монографії), дискретності (переривчастості), коли для розуміння сенсу необхідно прочитати всі попередні роботи. Знаходити інформацію стало складніше, до того ж швидкість читання та сприйняття інформації людиною залишилися на тому ж рівні, внаслідок чого «об'єктивні або суб'єктивні фактори, що перешкоджають отриманню потрібної інформації та ускладнюють використання документів як джерел інформації», дослідники називають «інформаційним бар'єром».

З появою нових способів створення та поширення інформації (зокрема, інтернету) зростання потоку документів збільшується у рази. За даними International Data Corporation (IDC) кількість даних на планеті «подвоюється кожні два роки, причому частка корисної інформації становить всього 35% від усієї згенерованої», а «обсяг загальносвітових даних зросте з 33 зеттабайт у 2018 році та на 175 у 2025». Отже, збільшення потоку документів не завжди є збільшенням нової та корисної інформації.

При аналізі готових рішень було виявлено що не існує окремого застосунку, що б вміщував би у собі всі необхідні функції, такі як пошук, обробка, збереження контенту. Частіше за все користувачі використовують нотатки різного формату, тематичні форуми, соціальні мережі та ін, але всі вони вузько направлені та не забезпечують увесь потрібний функціонал, або створені з іншою метою. Отже, варто сформулювати чіткі функціональні вимоги, до майбутньої інформаційної системи:

1. Збереження та керування різними видами інформації та контентом, знайденим в інтернеті;

2. Облік та керування створеними списками;

3. Вбудований пошук та структуризація знайдених даних;

4. Можливість зберігати різні види довідкових джерел: книги, відео, картинки, курси, файли, тощо;

5. Можливість ділитись та додавати уже готові списки.

Метою дослідження є створення програмного застосунку, за описаними вище вимогами, що дозволить шукати, зберігати, структурувати та обробляти корисну, потрібну та важливу інформацію, яку не хочеться загубити серед великого потоку даних з яким ми зустрічаємось щодня.

1 ПОШУК ТА АНАЛІЗ ТЕХНОЛОГІЙ ТА МЕТОДІВ ДЛЯ РЕАЛІЗАЦІЇ

Вибір методів та технологій для проектування будь-якого продукту є важливим етапом, оскільки він впливає на процес його розробки та на певні архітектурні особливості, притаманні вибраним технологіям

1.1 Вибір методу реалізації

Зараз існує декілька основних методів реалізації інтерфейсу для інформаційної системи. Кожен із них має свої переваги та недоліки. При створенні корпоративних рішень компанії часто не зупиняються на одному із варіантів, що допомагає розширити кількість користувачів та покращити досвід використання.

Настільний додаток — це програмна програма, яку можна запускати на окремому комп’ютері для виконання певного завдання кінцевим користувачем. Вони створюються для роботи на певній операційній системі, наприклад Windows, Mac або Linux [1]. Оновлення настільних програм повинні встановлюватися кінцевими користувачами. Оновлення можуть бути опубліковані через Інтернет, але інсталяція зазвичай виконується вручну кінцевим користувачем. Настільні програми розроблені для роботи в ізольованому середовищі, тому вони мають менше проблем із безпекою. Можливість працювати без підключення до Інтернету є ще однією загальною характеристикою настільних додатків.

Мобільний додаток – це спеціально розроблене під функціональні можливості гаджетів програмне забезпечення[2]. Призначення ПЗ може бути найрізноманітнішим: послуги, магазини, розваги, онлайн-помічники та інше. Ці програми завантажуються та встановлюються самим користувачем через

мобільні маркетплейси. Найбільші майданчики – AppStore, Google Play. Технічно всі програми створюються під конкретну платформу мобільного гаджета. Найбільш популярні операційні системи – iOS, Android, Windows Phone. Мінусом такої реалізації, як і попередній спосіб є необхідність встановлення користувачем додаткового пз, що буде ускладнювати взаємодію користувачів між із нашою інформаційною системою. Також, є потреба створення додатку під декілька операційних систем та необхідність самостійного оновлення додатку користувачем.

Web-сайти - зазвичай носять інформаційний характер. Складаються з веб-сторінок, об'єднаних один з одним у єдиний ресурс. Мають просту архітектуру на основі HTML-коду. Служать як платформу для надання контенту для відвідувачів: можуть містити текст, зображення або музику. Сайти не надають можливості взаємодії із програмою. Користувачі не мають доступу до розміщення своєї інформації, крім заповнення форми для отримання підписки.

Чат-бот - це програма, яка імітує реальну розмову з користувачем. Чат-боти дозволяють спілкуватися за допомогою текстових або аудіо повідомлень на сайтах, месенджерах, мобільних додатках або по телефону [3]. Більшість сучасних месенджерів мають у собі функціонал що дозволяє розробникам створювати власних віртуальних асистентів під свої потреби. Перевагами такого варіанту реалізації нашої інформаційної системи є зручний доступ до чат-бота із будь-якого пристрою де встановлений один із месенджерів. Користувач матиме змогу у будь-який момент додати нове джерело чи книгу у список літератури, а також поділитись ним із іншими.

Найбільш популярними месенджерами в Україні є Telegram та Viber. Вони надають доволі широкий функціонал та великий набір API-запитів. Детальніше про доступні можливості чат-ботів цих месенджерів можна ознайомитись на офіційних сайтах месенджерів: https://core.telegram.org/bots та https://developers.viber.com/docs/all/ відповідно.

1.2 Вибір технологій реалізації

Машинний код є рідною мовою комп’ютерів і не потребує трансформації для розуміння машини. Концепція двійкового коду (0 і 1), на якій базується машинний код, була розроблена Готфрідом Лейбніцем наприкінці 17 століття. Програмування в машинному коді вимагало б від людей запам’ятовувати числові коди та складні модифікації лише для взаємодії з комп’ютерами. В результаті майже ніхто не програмує безпосередньо в машинному коді, тому розробники постійно намагались спростити взаємодію комп’ютерів та людей для створення програм.

Зараз існує три основних підходи до створення інформаційних систем: Pro-code, Low-code і No-code.

Pro code означає написання коду звичайною мовою програмування, наприклад Python, або спеціалізованою мовою програмування, як SQL. Існує багато типів мов програмування та способів їх класифікації.

Існуючі мови програмування можна розділити на дві групи: процедурні та непроцедурні (рисунок 1.1).

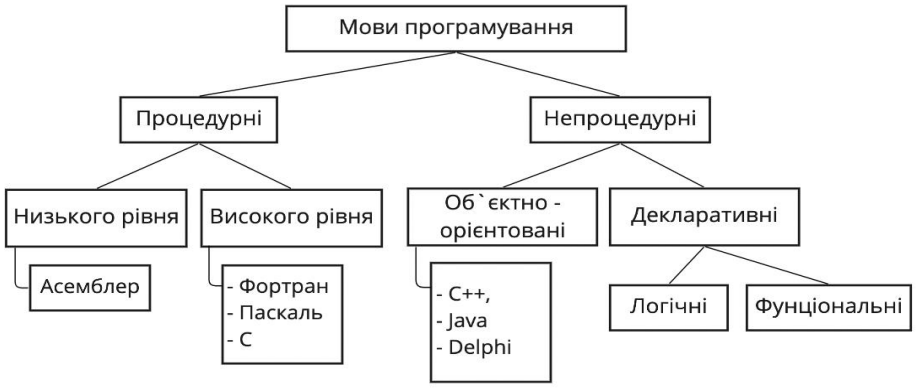

Рисунок 1.1 Загальна класифікація мов програмування

Процедурні (або алгоритмічні) програми є системою команд для вирішення конкретної задачі. Роль комп'ютера зводиться до послідовного виконання цих розпоряджень. Процедурні мови поділяються на мови низького та високого рівня.

Різні типи процесорів мають різні набори команд. Якщо мова програмування орієнтована конкретний тип процесора і враховує його особливості, вона називається мовою програмування низького рівня. Тобто, команди, написані на мові низького рівня близькі до машинного коду і спрямовані на певні команди процесора.

Мови низького рівня (машинно орієнтовані [4] дозволяють створювати програми з машинних кодів, зазвичай у шістнадцятковій формі. З ними важко працювати, але створені з їх допомогою програми займають менше місця у пам'яті та працюють швидше. За допомогою цих мов зручніше розробляти системні програми, драйвери (програми управління пристроями комп'ютера), деякі інші види програм. Мовою низького рівня (машинно-орієнтованим) є Ассемблер , який просто представляє кожну команду машинного коду, але не у вигляді чисел, а за допомогою умовних символьних позначень, які називають мнемоніками.

Мови кодування високого рівня[5] мають вищий рівень абстракції. Це означає, що вони ближчі до людської мови і далі від машинного коду. Такі мови легше вивчати та використовувати, але вони зазвичай пропонують менше функціональних можливостей та прямого контролю над комп'ютером. Вони, як правило, більш автоматизовані, тобто одна команда фактично виконує безліч заздалегідь запрограмованих речей, щоб зробити програмування більш простим та ефективним. Прикладами мов високого рівня є Python, Java, C++, JavaScript та ін.

Основна перевага алгоритмічних мов високого рівня - можливість опису програм вирішення завдань у максимально зручній для сприйняття людиною. Але оскільки кожне сімейство ЕОМ має свою власну, специфічну внутрішню (машинну) мову і може виконувати лише ті команди, які записані цією мовою, то для перекладу вихідних програм машинною мовою використовуються спеціальні програми-транслятори. Робота всіх трансляторів будується за одним із двох принципів: інтерпретація чи компіляція.

Інтерпретація має на увазі пооператорну трансляцію та подальше виконання відтрансльованого оператора вихідної програми. У зв'язку з цим можна відзначити два недоліки методу інтерпретації: по-перше, програма, що інтерпретує, повинна перебувати в пам'яті ЕОМ протягом всього процесу виконання вихідної програми, тобто займати певний обсяг пам'яті; по-друге, процес трансляції одного і того ж оператора повторюється стільки разів, скільки разів має виконуватися ця команда у програмі, що різко знижує продуктивність роботи програми. Незважаючи на зазначені недоліки, транслятори-інтерпретатори набули достатнього поширення, оскільки вони зручні при розробці та налагодженні вихідних програм.

При компіляції процеси трансляції та виконання розділені в часі: спочатку вихідна програма повністю перекладається машинною мовою (після чого наявність транслятора в оперативній пам'яті стає непотрібною), а потім така програма може багаторазово виконуватися. Отже, однієї і тієї ж програми трансляція методом компіляції забезпечує більш високу продуктивність обчислювальної системи при скороченні необхідної оперативної пам'яті.

Low- і zero-code походять із попередніх інструментів швидкої розробки додатків (RAD), таких як Excel, Lotus Notes та Microsoft Access, які також передавали деякі можливості розробки в руки бізнес-користувачів (тобто не спеціалістів з IT). Візуальне програмування, RAD та інші технології існують вже багато років. Інструменти low-code розробки є декларативними, тобто вони дозволяють візуально моделювати користувацькі інтерфейси, бізнес-логіку, алгоритми та обробку даних, які складають типову програму, без необхідності описувати керуючий код. Вони можуть генерувати тисячі рядків коду за лаштунками, до якого, по необхідності, можна отримати доступ і змінити (але таку можливість надають не усі платформи) [6]. Лоукод (low-code) і зерокод (zero-code) платформи тісно пов'язані з поняттям PaaS (platform as a service, «платформа як послуга») [7], що пропонує можливості додатків у вигляді хмарних сервісів, а не набору коду, які пише розробник. Платформи, відомі як "сервіси додатків", або "конструктори додатків" (aPaas, application

platform as a service, «платформа додатків як послуга») [8], забезпечують найвищий у цій категорії рівень абстракції, дозволяючи розробникам моделювати додатки швидше, ніж програмувати їх, не турбуючись при цьому про операційну систему, сховищах чи програмних оновленнях.

І low-code, і zero-code надають засоби для розробки додатків без необхідності написання великої кількості коду та навичок програмування. Натомість користувачі отримують доступ до візуального середовища розробки, де вони можуть вибирати потрібні функціональні елементи з бібліотеки, з'єднувати їх візуально в робочий процес, зазвичай шляхом перетягування квадратиків на екрані, і так створювати власну програму.

Простіше кажучи, low-code розробник робить візуально те, що pro-code розробник зазвичай пише кодом. У low-code додатках знадобиться мінімум коду, у той час як для zero-code використовуються встановлені шаблони та моделі, не потребуючи код взагалі. Давайте розглянемо дані підходи докладніше: що в них спільне, їх плюси, мінуси і відмінності.

Low-Code це відносно новий напрямок у сфері IT, який розвинувся після переосмислення фреймворків і патернів програмування, що дозволяли пришвидшити написання і мінімізувати необхідність дублювання коду [9]. Перевагами такого способу розробки застосунків чи веб-сторінок є швидкість і умовна простота (він більш інтуїтивний для користувача на базовому рівні, але може бути більш заплутаним та складнішим за звичний нам код, коли потрібно створити кастомний функціонал, якого немає у стандартному наборі інструментів). Частіше за все у таких випадках рятують мови програмування, що підтримуються подібними платформами і забирають на себе частину логіки.

Low-code означає написання високорівневих конфігурацій і шаблонів, які налаштовують широкий діапазон поведінки для програмного компонента або системи. JSON і YAML[10], зараз є найпопулярнішими варіантами синтаксису для цього. Даний метод створення інформаційних систем не завжди простий — конфігурації та шаблони, як правило, мають спеціальні

атрибути, операції та схеми для кожного програмного компонента, який розробники повинні вивчити. І чим складнішою стає система конфігурації/шаблона, тим більше вона стає власною мовою програмування з високим кодом .

До основних переваг low-code підходу для бізнесу відносяться:

— швидкість та гнучкість. Функція drag-and-drop (перетягування елементів), логічні схеми і моделі даних, попередньо змодельовані бізнес-процеси та набір інших автоматизованих інструментів дозволяють швидко розробляти повноцінні кросплатформні додатки;

— зниження вартості розробки. Про це говорить сайт https://intellect.icu. Можливість створення більшої кількості програм за короткий час, а також зниження потреб у наймі додаткових розробників;

— зниження залежності від IT. Команди програмістів отримують можливість зосередитись на вирішенні першочергових завдань, а не витрачати ресурси на створення простих додатків та дрібні редагування;

— підвищення продуктивності. При low-code розробці не потрібно чекати, поки інший розробник закінчать свій етап створення проекту, а час більше не є стримуючим фактором для інновацій;

— зниження ризику/підвищення ROI (Return on Investment, або окупність, рентабельність інвестицій) [11]. Інтеграції даних, процеси забезпечення безпеки та крос платформна підтримка вже вшиті у платформу і можуть легко налаштовуватися у майбутньому.

— швидке розгортання та простота обслуговування. Внесені зміни можна скасувати в один клік, і ви зможете налаштувати програму відповідно до нових вимог у будь-який момент [12].

До недоліків даного підходу:

— нестача кастомізації, обумовлена модульною структурою low-code платформ, та обмежені можливості інтеграції. Чим більше візуальних інтерфейсів, тим більше обмежень. Якщо ви використовуєте дуже потужну low-code платформу, то вона за складністю не поступатиметься коду.

— прив'язка до постачальника ПЗ. Багато користувачів переживають, що вони так чи інакше будуть прив'язані до платформи. Деякі з постачальників використовують відкритий код та фреймворки, підтримка яких реалізована як у рамках платформи, так і за її межами, тоді як інші обмежують можливості для редагування програм після того, як ви перестанете використовувати цей інструмент.

— ще один, доволі вагомий мінус, це набагато менша кількість розробників, що працюють у цьому напрямку і, відповідно спільнота, тому знайти рішення чи відповідь на запитання буде набагато складніше.

No-code (або zero-code) це рішення призначені для непрофесійних розробників (Citizen Developers [13] - користувачі всередині компанії, які мають доступ до візуального середовища розробки) і не вимагають ні написання коду вручну, ні знань та досвіду в галузі програмування. Будучи візуальною мовою програмування, zero-code дозволяє створювати програми візуально, перетягуючи елементи з бібліотеки і вимагає мінімального навчання. Не використовуючи код, розробник швидше вирішує, що робить програму, а не як вона це робить: декларативний UI описує те, що користувач повинен бачити в результаті, а не послідовність досягнення цього результату. Zero-code часто називають майбутнім програмування: очікується, що найближчими роками 80% завдань та програм будуть реалізовані без необхідності написання коду.

"Майбутнє кодингу - це відсутність коду зовсім" - Кріс Ванстрат, CEO GitHub

У zero-code платформ є багато спільного з low-code рішеннями:

— швидкість розробки та застосування. Відсутність коду дозволяє настроювати всі компоненти програми через візуальні інтерфейси, дозволяючи рядовим розробникам комбінувати модулі на власний розсуд. У випадку з low-code платформою деякі модулі та функції можуть бути недоступні «з коробки» або шаблону, тому вам знадобиться хоча б мінімальний код. Але у випадку з інструментами зерокод-розробки більшість

елементів форми і компонентів дизайну є частиною платформи і можуть оброблятися всього в кілька кліків;

— поява aPaaS-рішень, що дозволяють рядовим розробникам створювати кросплатформні програми для різних систем. APaaS можна використовувати через браузер незалежно від того, в якій операційній системі працюють користувачі;

— і zero-code, і low-code платформи допомагають бізнесам створювати та керувати власними програмами , тоді як професійні розробники можуть зосередитися на створенні більш складних та першочергових програмних рішень;

— переваги платформ zero-code значною мірою збігаються з перевагами low-code підходу: краща адаптивність, зниження вартості розробки, зменшення залежності від IT та підвищення продуктивності, простота в обслуговуванні та гнучкість.

Є й інші позитивні сторони zero-code:

— прості, доступні та ефективні платформи для людей, що не мають спеціальних технічних знань – майбутнє у побудові додатків;

— швидкість створення програми, що дозволяє швидко запускати ідеї за лічені дні, а не місяці;

— красиво оформлені шаблони - функціональність та дизайн «з коробки»;

— простий у використанні інтерфейс завдяки функції drag-and-drop, вам потрібно тільки розмістити елементи на формі і все, можна запускати. Не потрібно нічого тестувати та дебатувати.

Незважаючи на очевидні переваги zero-code розробки, власникам бізнесу необхідно розуміти приховані недоліки та ризики, пов'язані з її використанням:

— відсутність гнучкості. Zero-рішення не настільки гнучкі, як написання коду з нуля, а жорсткі шаблони обмежують те, що можна

побудувати: ваша платформа може не містити необхідних компонентів «з коробки» для вирішення конкретних прикладних завдань;

— користувачі повинні чітко розуміти свої вимоги та визначати, чи збігаються вони з можливостями та обмеженнями конкретного інструменту;

— ризики виникнення «тіньового IT» , коли офіційний IT-відділ втрачає видимість процесів розробки, що відбуваються в організації, а проблеми з безпекою виникають через відсутність контролю;

— володіння вихідним кодом. Проблеми, пов'язані із залежністю від постачальників ПЗ, які можуть виникнути, якщо ви вирішите перестати використовувати zero-code платформу, обмежений доступ або відсутність доступу до вихідного коду, відсутність чіткої документації тощо.

Експерти галузі прогнозують, що в майбутньому low-code буде продовжувати застосовуватися підприємствами, особливо для швидкого розвитку та конкретних потреб бізнесу, хоча low-code та no-code не повністю замінять традиційну розробку додатків.

За оцінками аналітиків Gartner, ринок low-code/no-code виріс на 23% у 2020 році, досягнувши 11,3 мільярда доларів, і виросте до 13,8 мільярдів доларів у 2021 році і майже до 30 мільярдів доларів до 2025 року [14]. Gartner також прогнозує, що розробка програм з використанням даних методів розробки становитиме 65% від усієї діяльності з розробки додатків до 2024 року, в основному для малих і середніх проектів. Між тим, за даними досліджень компанії Forrester, близько половини компаній сьогодні використовують платформу з low-code/no-code [15], але до кінця 2022 року ця кількість може зрости до 75%. Однак підприємства продовжуватимуть практику традиційної розробки додатків, які потребують ширшої функціональності додатків, керування даними та розгортання в певних архітектурах чи середовищах.

1.3 Вибір Low-code платформи

Кожна із існуючих платформ, як і звичні нам мови програмування, створені для вирішення певних задач, хоч вони настільки гнучкі як звичайний код. Нижче наведені приклади таких платформ:

OutSystems — це платформа для розробки мобільних і веб-додатків підприємства, які працюють у хмарі, локальному чи гібридному середовищі [16].

Mendix Studio — універсальна low-code платформа для розробки корпоративних програм та автоматизації бізнес-процесів. Дозволяє користувачам без навичок програмування швидко створювати MVP, впроваджувати та тестувати його, отримуючи результат для бізнесу за короткий термін [17].

Microsoft Power Platform — це перелік продуктів, що інтегровані між собою, і призначені для бізнес-аналітики, автоматизації бізнес-процесів, написання додатків мовою лоу-код, або використання готових шаблонів додатків [18].

Appian — це хмарна платформа основні функції якої обертаються навколо управління бізнес-процесами (BPM), автоматизації роботизованих процесів (RPA), керування справами, управління контентом та інтелектуальної автоматизації [19].

Oracle Application Express (APEX) — це low-code платформа для розробки від Oracle. APEX дозволяє своїм користувачам створювати масштабовані, безпечні корпоративні програми, які можна розгорнути в будь-якому місці [20].

Corezoid — це Cloud Process Engine, який дозволяє створювати, розміщувати та запускати алгоритми будь-якої складності в хмарі: чат-боти, сценарії спілкування, функціональність CRM, підтримка клієнтів, багатоканальні маркетингові кампанії, моніторинг обладнання, рішення для боротьби з шахрайством, фінанси, інструменти управління тощо [21].

Оскільки інтерфейсом для нашої інформаційної системи буде чат-бот, Corezoid чудово підійде його реалізації завдяки наявному фреймворку, що

спрощує роботу із API месенджерів, а також високій гнучкості завдяки можливості змінювати структуру проекту та кастомізувати існуючу логіку під свої потреби.

1.4 Вибір СУБД

Система управління базами даних (СУБД) – це сукупність мовних і програмних засобів, які призначені для створення, ведення і сумісного використання БД багатьма користувачами. Системи управління базами даних дозволяють поєднувати великі обсяги інформації і обробляти їх, сортувати, робити вибірки за визначеними критеріями тощо. Простота використання СУБД дозволяє створювати нові бази даних, не звертаючись до програмування, а користуючись тільки вбудованими функціями. СУБД забезпечують правильність, повноту і несуперечливість даних, а також зручний доступ до них.

При виборі СУБД дуже важливо обрати ту базу даних, яка в найбільшій мірі відповідає вимогам до інформаційної системи. Відповідно до DB-Engines Ranking - Top 10 Most Popular Databases - March 2021 (URL: https://db-engines.com/en/ranking) найбільш популярними є:

1. Oracle - Relational, multi-model
2. MySQL - Relational, multi-model
3. Microsoft SQL Server - Relational, multi-model
4. PostgreSQL - Relational, multi-model
5. MongoDB - Document, multi-model

При порівнянні різних популярних баз даних, слід враховувати, чи зручна для користувача і чи масштабована конкретна СУБД, а також переконатися, що вона буде добре інтегруватися з іншими продуктами, які

вже використовуються. Крім того, під час вибору слід взяти до уваги вартість системи та підтримки, яку надає розробник.

Розглянемо перелічені СУБД. Серед них є чотири реляційних баз даних: Oracle, MySQL, Microsoft SQL Server, PostgreSQL та одна NoSql бд - MongoDB. Oracle та Microsoft SQL Server це ліцензійні продукти компаній Oracle та Microsoft відповідно. PostgreSQL та MySQL це реляційні системи управління базами даних з відкритим кодом. Проведемо порівняння бд у цих двох категоріях, а потім порівняємо обрані в кожній із категорій СУБД між собою.

MySQL – це реляційна система управління базами даних з відкритим кодом, що розробляється компанією Oracle. Цей код можна отримати безкоштовно, використовуючи ліцензію GNU, а також комерційні версії MySQL доступні під різними угодами [22]. PostgreSQL - це об'єктно-реляційна СУБД, розроблена Global Group. Вона також має відкритий вихідний код [23]. Основні відмінності між цими двома моделями СУБД:

- керування;
- підтримка платформ;
- методи доступу;
- секціонування;
- реплікація.

Управління цих двох моделях баз даних одна із найбільш істотних відмінностей. MySQL керується Oracle, тоді як Postgres доступний за ліцензією з відкритим кодом від Global Group. Таким чином, спостерігається підвищення інтересу до PostgreSQL за останні кілька років. Обидві СУБД можуть працювати на Linux, OS X, Solaris та Windows. MySQL також підтримує ОС FreeBSD. PostgreSQL підтримує операційну систему HP-UX, створену компанією Hewlett Packard.

Технології, які є спільними для MySQL і PostgreSQL — ADO.NET, JDBC і ODBC. ADO.NET є набором інтерфейсів додатків (API), який програмісти використовують для доступу до даних на основі XML. JDBC —

це API для мови Java, а ODBC — це стандартний для всіх інших мов. MySQL і PostgreSQL істотно відрізняються за власними способами виділення розділів, які визначають, як дані зберігаються на різних вузлах бази даних. MySQL використовує власну технологію під назвою MySQL Cluster для виконання горизонтальної кластеризації, яка складається зі створення кількох кластерів з одним екземпляром кластера у кожному вузлі. PostgreSQL не реалізує справжнього поділу, хоча він може забезпечити аналогічні можливості успадкування таблиці. Це завдання включає використання окремої під таблиці для управління кожним розділом. MySQL завжди був орієнтований на велику продуктивність, у той час як Postgresql був націлений на велику кількість налаштувань та стандартів. Але з часом ця ситуація змінилася і Postgres став більш продуктивним [24].

Обидва проекти мають відкритий вихідний код, але розвиваються по-різному. MySQL під керівництвом компанії Oracle гальмує у розвитку. Postgresql розвивається групою програмістів та кількома компаніями. Нові версії виходять досить часто та мають нові функції. Основна відмінність MySQL від СУБД промислового рівня полягає в тому, що MySQL призначена для вирішення вузького кола завдань. У свою чергу, СУБД промислового рівня не мають обмежень у застосуванні, починаючи від простої бази даних, що обслуговує сайт або невелику компанію, і закінчуючи величезними потужними сховищами даних. Отже, у цій парі СУБД варто надати перевагу Postgresql.

База даних Oracle розроблена корпорацією Oracle і використовується як система для збору даних, що розглядаються як єдине ціле. Він використовується для зберігання та вилучення даних, а сервер бази даних використовується для керування загальними функціями. Але цей крок є загальним для всіх систем баз даних, будь то MS SQL або ще щось. Він використовується для обробки онлайн-транзакцій, зберігання даних, а також змішаних робочих навантажень бази даних. Вся більшість програмних систем,

веб-сайтів, корпорацій і т. д. використовують одну форму або інші системи баз даних для обробки своїх даних.

SQL Server було створено компанією Microsoft і є однією з найпопулярніших систем управління базами даних (СУБД) у світі. Ця СУБД підходить для різних проектів: від невеликих додатків до великих високонавантажених проектів. SQL Server довгий час був виключно системою керування базами даних для Windows, проте починаючи з версії 16 ця система доступна і на Linux. SQL Server характеризується такими особливостями як:

– продуктивність (SQL Server працює дуже швидко);

– надійність;

– безпека (SQL Server надає шифрування даних);

– простота.

З цієї СУБД щодо легко працювати та вести адміністрування. Для управління SQL Server використовується management studio - великий і повноцінний клієнт, достатній для виконання переважної більшості операцій будь-якого рівня - від простого користувача до адміністратора. У свою чергу через велику кількість платформ oracle пропонує web- консолі для роботи та адміністрування СУБД.

Обидві СУБД використовують свої діалекти мови запитів. І хоча вони засновані на тому самому стандарті ansi sql, діалекти вийшли різні, а багато їхніх функцій – специфічними і несумісними. Слід зазначити, що PL/SQL потужніший за T-SQL.

Порівнявши характеристики двох передових СУБД: MS SQL Server і Oracle було встановлено, що СУБД Oracle має переваги такі, як: висока надійність і захищеність, можливість роботи на платформі будь-якої операційної системи. З іншого боку, СУБД MS SQL Server має більш низьку ціну володіння. Розглянувши переваги та недоліки цих СУБД, можна дійти висновку, що СУБД Oracle більше підходить для використання у великих підприємствах і організаціях, що і підтверджується статистикою.

MongoDB - система управління базами даних, яка працює з документо орієнтованою моделлю даних. Належить до класу NoSQL і є Schema-less. На відміну від реляційних СУБД, MongoDB не потребує таблиці, схеми або окремої мови запитів. Інформація зберігається як документів чи колекцій.

Кожна база даних містить колекції, які містять документи. Кожен документ може бути різним із різною кількістю полів. Розмір та зміст кожного документа можуть відрізнятися один від одного.

Структура документа більше відповідає тому, як розробники конструюють свої класи та об'єкти відповідними мовами програмування. Розробники часто кажуть, що їхні класи не є рядками та стовпцями, а мають чітку структуру з парами ключ-значення.

Рядки (або документи, що викликаються MongoDB) не обов'язково повинні мати заздалегідь певну схему. Проте, поля можуть бути створені на льоту. Модель даних, доступна в MongoDB, дозволяє вам представляти ієрархічні відносини, простіше зберігати масиви та інші складніші структури.

Щоб розібратися у тому, чим відрізняється реляційна база даних від нереляційної, необхідно окремо розглянути їх особливості. У реляційних базах даних зберігаються структуровані дані, які можуть представляти об'єкти з навколишнього світу. Так, у побудованій за такою моделлю базі можуть бути відомості про реальну людину або про те, які товари покупець склав у кошик у торговому центрі. Ці дані обов'язково групуються у таблицях у заданому форматі. Нереляційні бази даних улаштовані по-іншому. Типи даних у яких безпосередньо залежить від виду самої БД. Так, якщо йдеться про документно-орієнтовану базу, дані в ній будуть міститися у форматі ієрархії і можуть описувати різні об'єкти з довільними характеристиками. У цьому, мабуть, одна з найважливіших переваг NoSQL – база дозволяє зберігати величезні обсяги інформації у вигляді єдиної сутності. Такий обсяг за умов реляційні бази даних довелося б розбити кілька окремих, хоч і взаємозалежних, таблиць.

Перевагами MongoDB серед СУБД є:

– Зрозуміла структура кожного об’єкта;

– Легко масштабується;

– Для зберігання даних використовується внутрішня пам’ять, що дозволяє отримувати більш швидкий доступ;

– Дані зберігаються у вигляді JSON документів;

– MongoDB підтримує динамічні запити документів (document-based query);

– Немає необхідності маппінгу об’єктів додатків в об’єкти БД.

До переваг MongoDB також відносять відсутність схеми даних, але це може бути як перевагою, так і недоліком, в залежності від проекту та його потреб.

Платформа Corezoid дозволяє взаємодіяти із будь-якими базами даних. Це можна зробити двома способами: за допомогою логіки API call (цей спосіб потребує створення Application Programming Interface для роботи із базою даних, наприклад на основі Python та Flask) або з використанням DB Call, що з'явився у версії Corezoid 5.5.1 [25]. Але, завдяки структурі даних, що використовується у Corezoid, документоорієнтовані бази даних більш зручні у використанні, оскільки дані зберігаються у такому ж форматі, як і у самому Corezoid, що значно прискорить створення інформаційної системи. Враховуючи переваги та недоліки описаних вище СУБД для створення дипломного проекту було обрано MongoDB.

1.5 Висновки до розділу

У даному розділі було розглянуто технології що використовуються для створення інформаційних систем: настільний додаток, мобільний додаток, веб-сайт, чат-бот та проаналізовано їх переваги та недоліки, а також обрано технологію для реалізації нашої системи. Крім цього, був проведений аналіз методів реалізації, проаналізовано особливості розробки на базі трьох

основних підходів до створення інформаційних систем: Pro-code, Low-code і No-code та знайдено оптимальний для створення інформаційної системи на базі обраної технології.

Відносно обраної технології та методу реалізації було проаналізовано існуючі СУБД та обрану ту, що найкраще підходить для реалізації нашої системи.

Важливою частиною розділу є обґрунтування вибору технологій для розробки. З метою пришвидшення та спрощення розробки системи була обрана технологія чат-бот, та платформа Corezoid як метод реалізації для програмування логіки передачі та обробки даних. Варто зазначити, що їх використання відповідає обраній архітектурі. MongoDB була обрана завдяки тому, що вона має схожу до платформи структуру даних, що забезпечує простоту інтеграції.

2 ОПИС ОСНОВНИХ ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ ПЛАТФОРМИ COREZOID

У даному розділі ми розглянемо функціональні можливості та конкурентні переваги хмарної платформи Corezoid, її архітектурні особливості та наведемо приклади застосування.

2.1 Модель використання

Corezoid — це відносно нова оригінальна багатоцільова хмарна платформа для розробки для обслуговування, управління, виконання, моніторингу та оптимізації програмного забезпечення для автоматичних або автоматизованих бізнес-процесів[26].

Corezoid слідує сучасній концепції Platform as a Service, основною мовою програмування Corezoid є Erlang, яка забезпечує найвищий ступінь паралельності та масштабованості.

Хмарна платформа Corezoid виконує основні функції, показані на рисунку 2.1

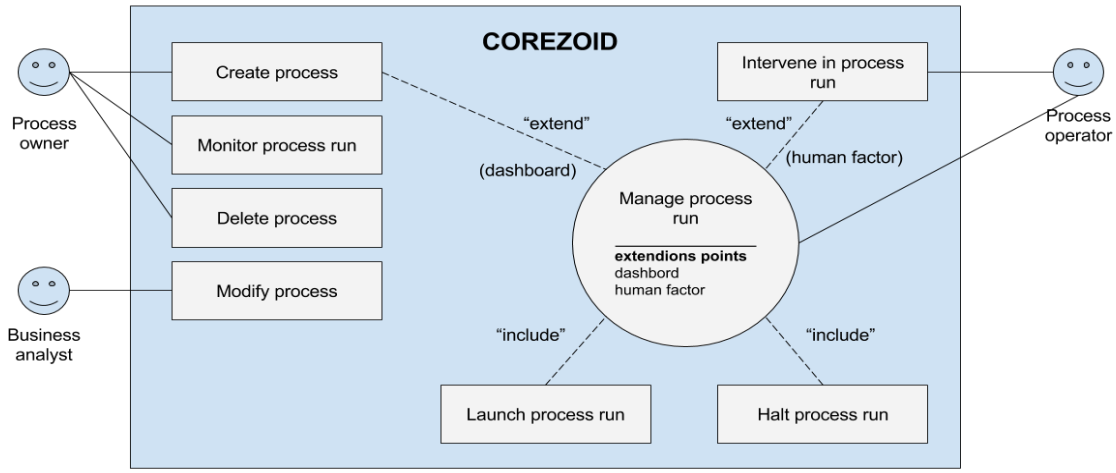

Рисунок 2.1 Модель використання платформи Corezoid

Кожна система, що буде створюватись на основі платформи складатиметься із основних складових: тобто зовнішнє обладнання, програмне забезпечення та користувачі, які взаємодіють із системою. Користувачів системи можна розділити на три категорії:

— Власник процесу: категорія користувачів з повноваженнями створювати та видаляти бізнес-процеси (процеси). Це повноцінні розробники, що створюють корову логіку інформаційної системи. Як правило, власники процесів несуть відповідальність за ефективність та ефективність своїх процесів, і тому моніторинг процесу вважається для них важливим результатом;

— Оператор процесу: користувачі з обмеженим доступом до створених процесів, який в залежності від прав доступу до конкретного процесу чи папки, що налаштовуються власником, можуть переглядати, моніторити чи редагувати уже створену логіку;

— Бізнес-аналітик: категорія користувачів, які займаються проектуванням і вдосконаленням бізнес-процесів.

На верхньому рівні можна розрізнити вісім варіантів використання, два з яких мають центральну роль.

1. Зміна процесу — розширений варіант використання, що дозволяє визначити бізнес-процес. Визначення бізнес-процесу представлено набором діаграм у спеціальному графічному редакторі. Як правило, визначення процесу як готового здійснюється кількома ітераціями, причому кожна наступна ітерація змінює та доповнює діаграми, побудовані на попередній ітерації.

1.2. Створення і видалення процесів — прості параметри використання, необхідні для розрізнення привілеїв користувача.

2. Керування запуском процесу — комбінований варіант використання, що дозволяє запускати виконання бізнес-процесу. Один і той самий бізнес-процес може виконуватися в кількох випадках як послідовно, так і одночасно. Опція поєднує ряд конкретних випадків використання.

2.1 Втручання в процес — усі види ручного втручання в процес, якщо це передбачено нормативними актами.

2.2 Моніторинг виконання процесів — однією з найсильніших конкурентних переваг платформи Corezoid є наявність вбудованих інструментів для моніторингу виконання бізнес-процесів.

2.3 Запуск і зупинка виконання процесу — важливі варіанти використання. Запуск процесу включає початкове завантаження даних, тоді як зупинка та завершення процесу може включати завантаження даних, якщо необхідно [27].

2.2 Модель предметної області

Модель предметної області — це скінченний набір сутностей і понять, що використовуються в предметній області, а також набір відносин між цими поняттями. Модель домену для платформи Corezoid показана на рисунку 3.2.

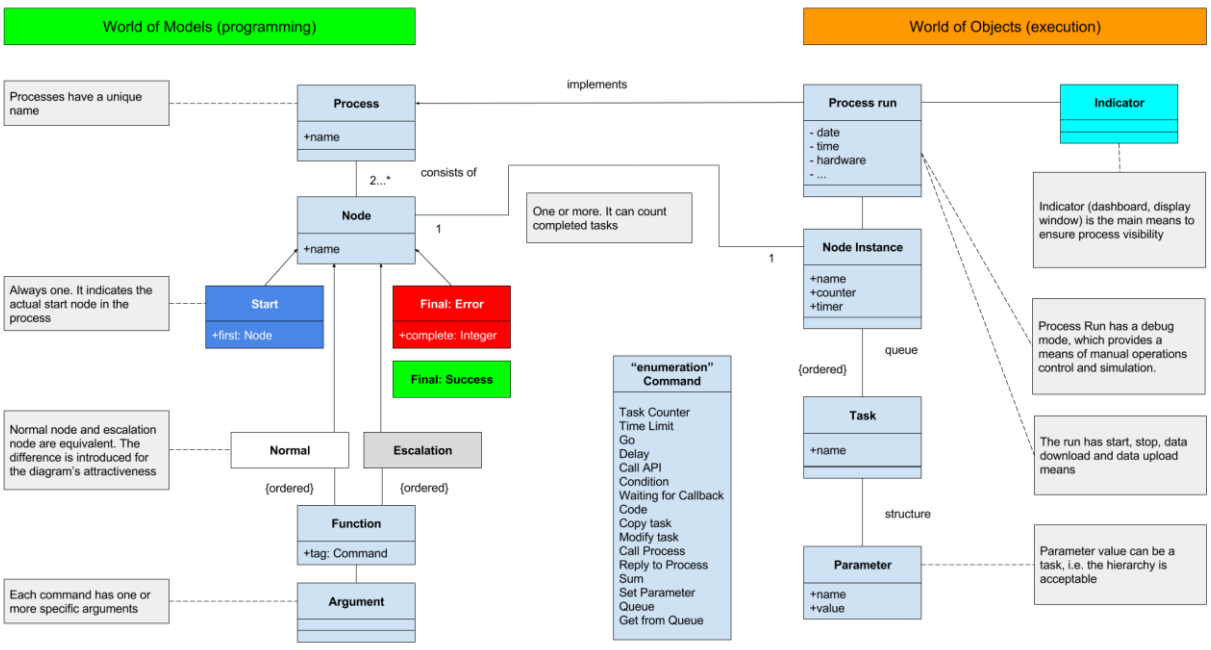

Рисунок 3.2 Corezoid Domain Model

Бізнес-процес (скорочено процес) — це загальна концепція платформи, що описується. Взагалі, бізнес-процес — це частково впорядкований набір дій (іноді їх називають діяльністю чи завданнями), спрямованих на досягнення певної мети (виробництво товарів, надання послуги). У цьому випадку дії передбачають прості обчислення та перетворення частин даних (або записів у термінах традиційних систем управління базами даних), які в цьому контексті називаються об’єктами.

Сама по собі платформа Corezoid дозволяє маніпулювати об’єктами, обчислювати їх кількість, підсумовувати числові значення в об’єктах тощо. У деяких простих, але, тим не менш, важливих випадках, наприклад, стосовно багатьох фінансових бізнес-процесів, цього вже достатньо. Однак сфера застосування платформи Corezoid цим не обмежується, оскільки одним з найважливіших інструментів платформи є вільна взаємодія з будь-якими зовнішніми додатками через API (інтерфейс програмного забезпечення).

Вузол (або нода) [28] є основною концепцією платформи, яка описується. Вузол визначає, що потрібно зробити з об'єктом за допомогою лінійно впорядкованого набору функцій . Вузли можуть бути чотирьох типів.

Початковий вузол (рисунок 3.3) завжди є одним для процесу і використовується для визначення вузла, який буде запущений першим, а також служить орієнтиром для налагодження та відстеження процесу.

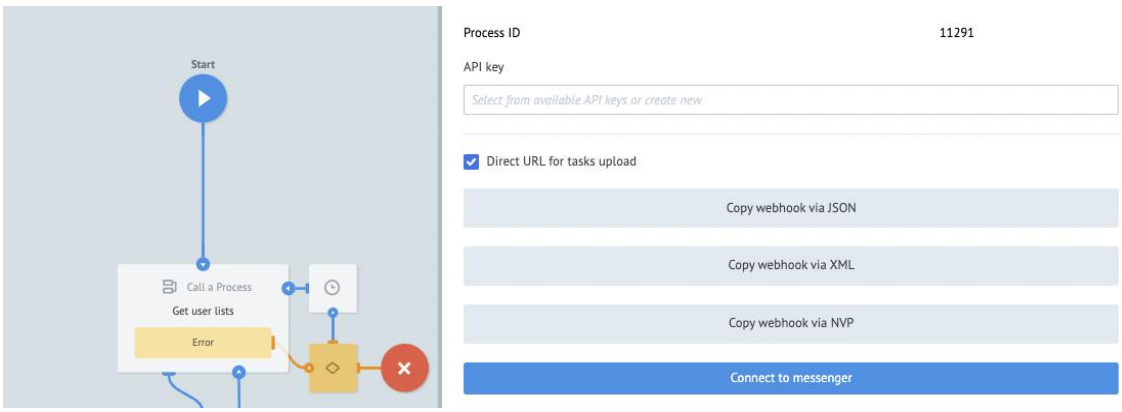

Рисунок 3.3 – Початковий вузол процесу в Corezoid

Кінцевий вузол у більшості випадків повинен бути присутнім у процесі, але їх може бути більше одного. Як і початковий вузол, кінцевий вузол не

виконує жодних дій, а лише фіксує факт завершення обробки об’єкта у вузлі. Для зручності кінцевих вузлів існує два види (рисунок 3.4), що відрізняються лише візуально і використовуються для позначення успішного та виняткового результату виконання процесу.

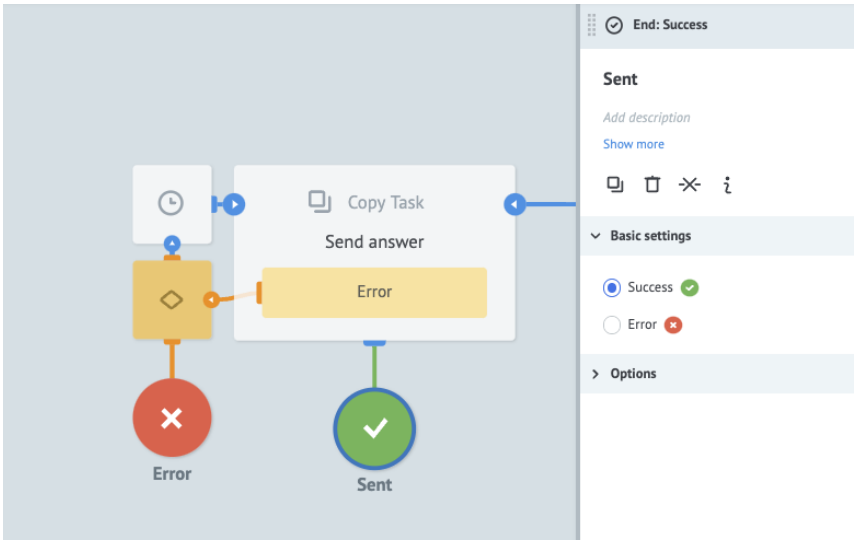

Рисунок 3.4 – Фінальні ноди Corezoid процесу

Звичайний вузол і вузол ескалації семантично еквівалентні. Їх відмінності відносні і введені з описових причин: ці вузли пропонується позначати різними кольорами на діаграмах. Таким чином, передбачається, що розробник процесу використовує звичайні вузли для опису «нормального» перебігу процесу, а для опису реакції на виняткові умови використовує вузли ескалації. Зазначені концепції показані в лівій частині діаграми на рисунок 3.2 і пов'язані з описом процесу. У правій частині діаграми показані сутності, пов’язані з виконанням процесу. Вони пов’язані за допомогою функцій (команд), описаних у вузлах, і застосовуються до завдань у екземплярах вузлів.

2.3 Запуск процесу

Запуск процесу — це фактична реалізація бізнес-процесу. Будь-який процес має початкову точку і може мати кінцеву, але можуть бути й циклічні процеси. Будь-який процес може проходити будь-яку кількість запусків, у тому числі ініційовані паралельно. Запуск процесу виконується екземплярами вузла, який зазвичай називають task, тобто задача. Хід процесу можна відстежувати за допомогою індикаторів (або інформаційних панелей).

Індикатор – це засіб для спостереження та моніторингу ходу процесу. Для відображення статистики за період та/або миттєвого значення в реальному часі надаються такі індикатори:

— кількість завдань, що надходять до вузла;

— кількість завдань, що виходять з вузла;

— загальна кількість спожитих тактів;

— кількість спожитих тактів, обраних процесом.

Примірник вузла або екземпляр вузла пов’язаний з чергою об’єктів , що підлягають обробці. Екземпляри вузлів працюють незалежно й асинхронно, оскільки над ними немає «контролера». Синхронізація та взаємодія між екземплярами вузлів відбуваються завдяки обміну об'єктами. Кожен екземпляр вузла вибирає наступний об’єкт зі своєї черги, виконує задані функції, які можуть мати додатковий компонент, і згодом відправляє об’єкт на обробку в чергу іншого екземпляра, а потім негайно переходить до обробки наступного об’єкта. Таким чином, вузли утворюють ланцюг, який обробляє задачу з максимально можливим коефіцієнтом продуктивності.

Задача (task) – це програмний об'єкт, який рухається по процесу із момента запуску до того, як він потрапить у фінальну ноду. Кожна окрема задача проходить певну чергу вузлів і рухається послідовно від одного вузла до наступного у черзі. В одному процесі одночасно можуть оброблятись безліч задач. Цей об’єкт являє собою пасивну структуру даних, яка використовується для зберігання та передачі інформації від одного екземпляра вузла обробки до іншого під час виконання. Платформа Corezoid є

революційною в обробці об’єктів, адже їх може бути справді багато: тисячі й мільйони.

Параметри в задачі повинні мати ім’я та значення, а значення може бути як простим типом, так і об`єктом чи списком з власними параметрами та значеннями. Отже, task це ієрархічна асоціативна пам’ять типу «ключ-значення».

2.4 Висновки до розділу

У другому розділі було розглянуто особливості роботи із обраною у першому розділі платформою Corezoid. Описано функціональні можливості та конкурентні переваги хмарної платформи, її архітектурні особливості та наведено приклади застосування. Була описана та проілюстрована модель використання, модель предметної області, модель домену. Було визначено поняття запуску процесу та терміни що потрібні для розуміння цього поняття: індикатор, екземпляр вузла, задача, фінальний вузол та інші.

У результаті встановлено, кожна система, що буде створюватись на основі платформи складатиметься із основних складових: зовнішнє обладнання, програмне забезпечення та користувачі, які взаємодіють із системою. У платформі наявно три види користувачів, та вісім варіантів використання, два з яких мають центральну роль.

У результаті встановлено, що Corezoid чудово підходить для розробки системи керування чат-ботом, як і іншого комплексного ПЗ оскільки багатоцільова хмарна платформа для розробки, обслуговування, управління, виконання, моніторингу та оптимізації програмного забезпечення для автоматичних або автоматизованих бізнес-процесів, хоч у неї є свої особливості та недоліки.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Налаштування перед початком роботи

На даному етапі усе готово для того, щоб перейти до реалізації інформаційної системи обраними нами технологіями.

3.1.1 Розгортання Bot Platform

Bot Platform — це набір готових процесів Corezoid для створення та керування ботами в різних месенджерах. Corezoid дозволяє створювати універсальні процеси для управління бізнес-логікою в різних месенджерах замість того, щоб розробляти індивідуальну бізнес-логіку для кожного месенджера. Підтримувані месенджери:

— Facebook Messenger;

— Viber;

— Telegram;

— Apple Business Chat (iMessage);

— Whatsapp.

Bot Platform отримує доступ до ботів у месенджерах за допомогою HTTP API з ідентифікацією токена.

Для створення чат-бота було обрано два найбільш популярних в Україні мессенджера: Telegram та Viber. Завдяки широкому функціоналу та великому набору API-методів вони дозволяють створювати звучний та зрозумілий користувачу інтерфейс для взаємодії із створеною інформаційною системою.

Процес створення чат-боту в Telegram відбувається за використання BotFather. Його можна знайти за посиланням: https://t.me/BotFather. Щоб почати роботу потрібно ввести команду /start . Після цього створюємо нового бота ввівши команду /newbot та вказуємо назву, унікальний юзернейм, опис, аватар і т д. (рисунок 3.1). Якщо все було зроблено правильно - BotFather надішле у відповідь токен, який буде нам потрібний пізніше.

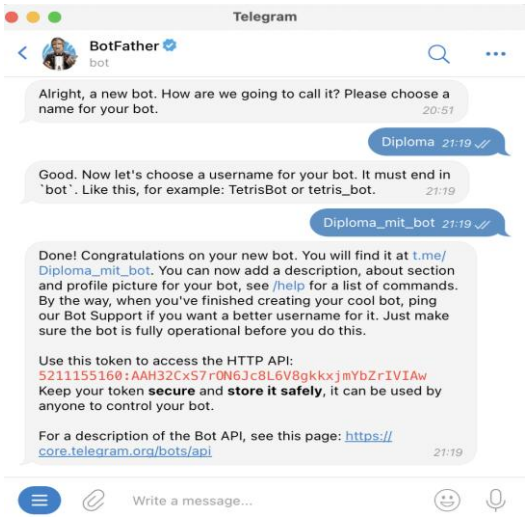

Рисунок 3.1 – Отримуємо токен

Для створення чат-ботів у Viber потрібна адмін панель. Для того, щоб авторизуватись достатньо ввести номер телефону та код підтвердження, що буде надіслано у смс. Аналогічно до реєстрації Telegram-бота вводимо інформацію про чат-бот і отримуємо токен (рисунок 3.2)

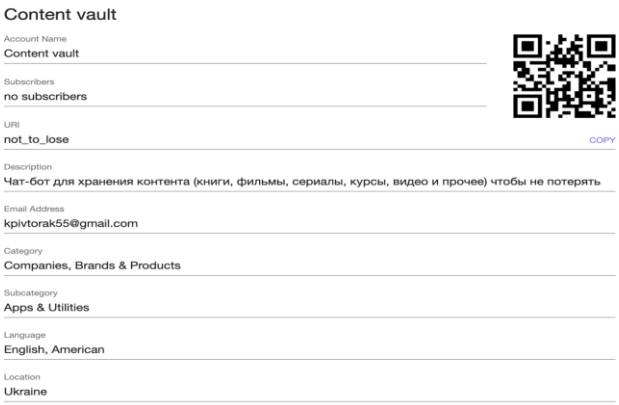

Рисунок 3.2 – Створений чат-бот у viber admin panel

Для розгортання Bot Platform потрібно обрати бажану компанію та натиснути Create > Bot Platform та у відповідному вікні вказати токени створених чат-ботів.

Створена Bot Platform складається із компонентів, що описані нижче. Для зручності вони будуть розташовані у тому ж порядку, який буде проходити задача з моменту отримання повідомлення від користувача до відправки відповіді ботом.

Процеси Viber/Telegram Receiver починаються із стартової ноди, що являє собою вебхук. Взаємодія користувача з чат-ботом (відправлені повідомлення та файли, натискання кнопок, переходи за посиланнями тощо) буде потрапляти сюди у вигляді окремого завдання для кожної події. Після чого перетворює отримані дані в уніфіковану стандартну структуру даних яка містить наступні параметри:

— channel — канал повідомлень. Доступні варіанти: telegram, facebook, viber, abc;

— chat_id — ідентифікатор чату, користувача тощо;

— event — тип події;

— message — об'єкт повідомлення.

Основний процес (Main) отримує перетворені дані від месенджерів, перевіряє користувача на наявність у діаграмі User Profile, і запускає початкову бізнес-логіку бота перед маршрутизацією завдань у підпроцеси. Якщо це новий користувач, то у вище згаданій діаграмі буде створений його профіль із референсом channel_chat_id, де channel та chat_id це відповідні параметри даного користувача.

Процес маршрутизатора (Router) виконує основну логіку обробки подій Bot Platform. В залежності від переданих даних із основного процесу та отриманих із відповідної діаграми даних про активність користувача він направляє завдання у потрібний підпроцес, створений розробником (рисунок 3.3).

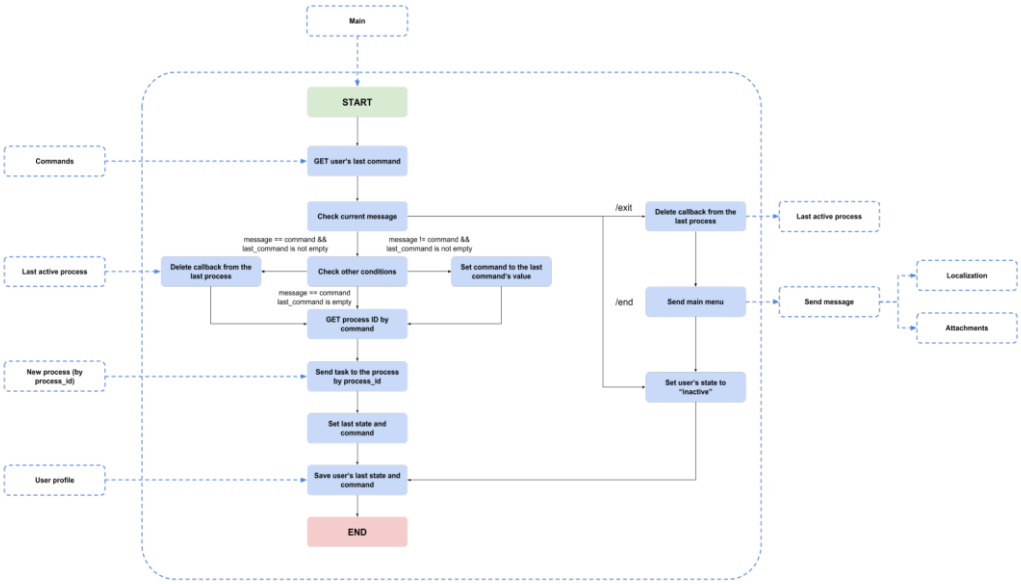

Рисунок 3.3 – Схема роботи процесу Router

Процес надсилання повідомлення (Send Message) отримує текстові шаблони, вкладення та значення аргументів, щоб динамічно об’єднати їх у єдине готове до надсилання повідомлення (рисунок 3.4). Крім того, він виконує локалізацію повідомлень.

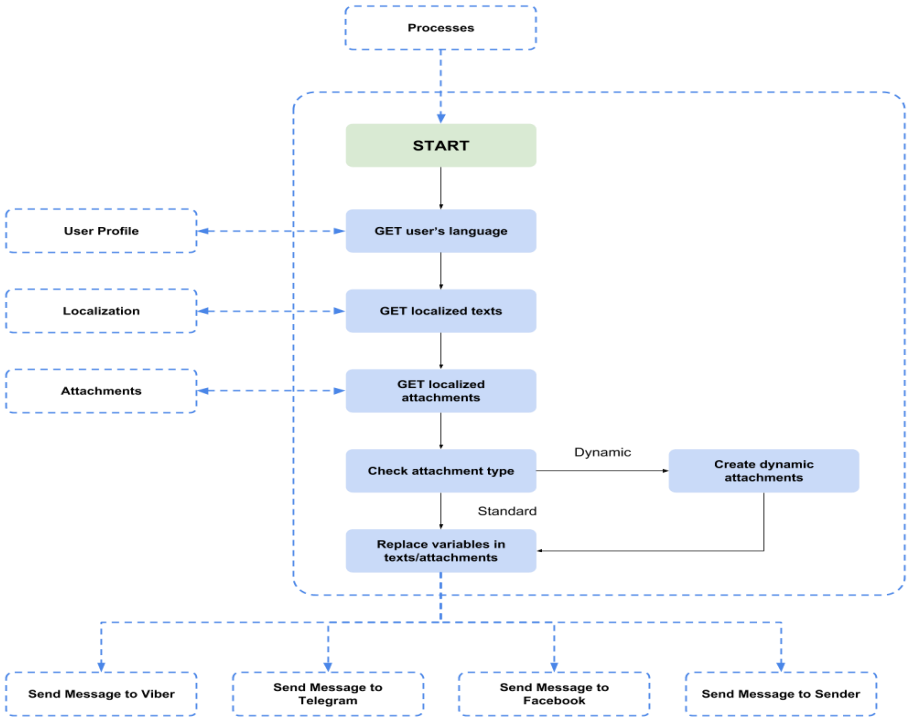

Рисунок 3.4 – Схема роботи процесу Send Message

Крім текстових повідомлень, месенджери підтримують різні вкладення, як-от кнопки, клавіатури, каруселі тощо. Для того, щоб їх використовувати

портібно створити відповідний тип вкладень у діаграмі вкладень (Attachments).

3.1.2 Створення та підключення бази даних

У попередньому розділі було обрано СУБД для роботи. MongoDB пропонує можливість хмарного хостингу. Для того, щоб розгорнути та налаштувати базу даних потрібно обрати регіон та кластерний рівень.

Для створення дипломного проекту було обрано безкоштовну версію оскільки її характеристик достатньо для повноцінної роботи. Якщо проект буде масштабуватись та потребуватиме більшого об'єму сховища, його завжди можна буде розширити. Наступними кроками є налаштування імені кластера, створення адміністратора бази даних та конфігурація правил доступу для безпечного використання. Після цього кластер буде розгорнутий та готовий до роботи (рисунок 3.5).

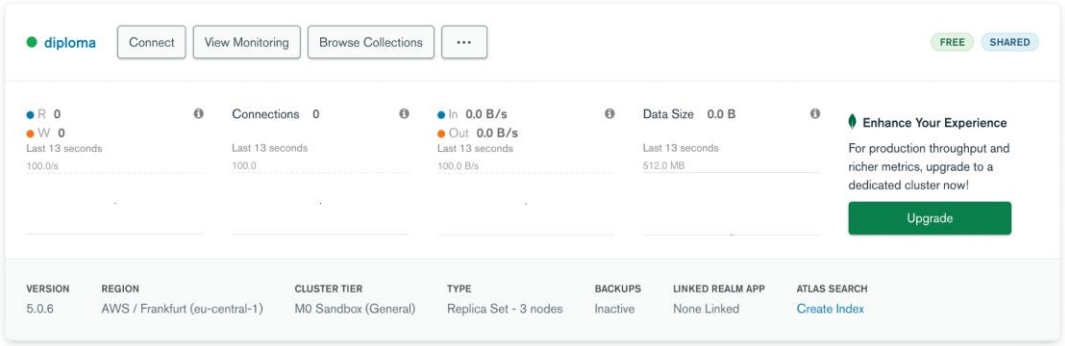

Рисунок 3.5 – Створений кластер

Інтеграція із системою Corezoid буде відбуватись завдяки API, що підтримує CRUD операції. Для розгортання Data API достатньо перейти у відповідний розділ адмін-панелі конкретного кластеру (рисунок 3.6)

Рисунок 3.6 – Data API для інтеграції із Corezoid

Для безпечної роботи із даною API потрібно створити ключ. Для цього треба натиснути Create API Key, ввести назву та отримати ключ. Його важливо зберегти, щоб у подальшому використовувати для формування запитів.

Існують наступні концепції для побудови схеми даних:

- комбінувати всі об’єкти що будуть використовуватись разом;
- можливе дублювання даних оскільки обчислення коштують дорожче за місце на диску;
- варто робити join-и під-час запису, а не під час читання;
- допустиме використання складних агрегацій в схемі;
- оптимізація схеми відбувається під найбільш часті випадки.

На відміну від реляційних баз даних, Mongodb не потребує нормалізації, а інколи, вона може навпаки здорощувати ціну на кластер та зменшувати швидкість виконання запитів.

Використовуючи базовий функціонал Corezoid створюємо API для роботи із Mongodb (по одному під кожну із основних операцій: Create, Read, Update, Delete). Логіка кожного із них буде складатись із базових кроків:

1. Валідація параметрів, що передаються;
2. API-запит на відповідний endpoint;
3. Перевірка відповіді від API;
4. Відповідь на виклик даного процесу.

Нижче представлений приклад реалізації методу Update (рисунок 3.7):

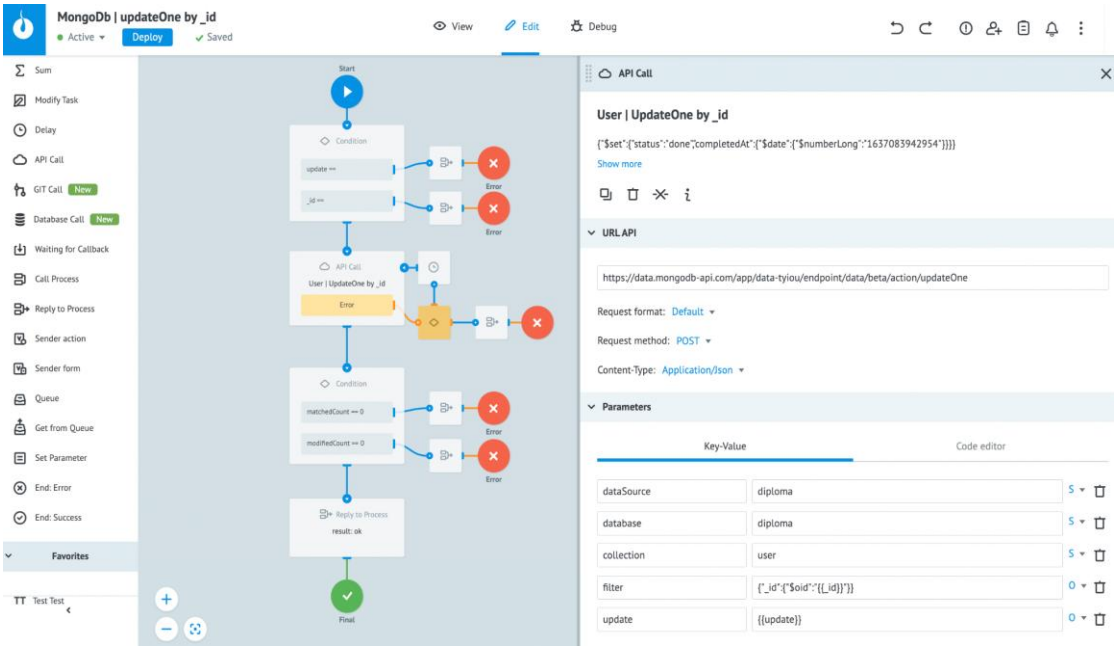

Рисунок 3.7 – Update метод для роботи із БД

Для виклику даного API використовується конструкція Call Process куди передаються дані для оновлення, фільтр та потрібну колекцію (рисунок 4.8).

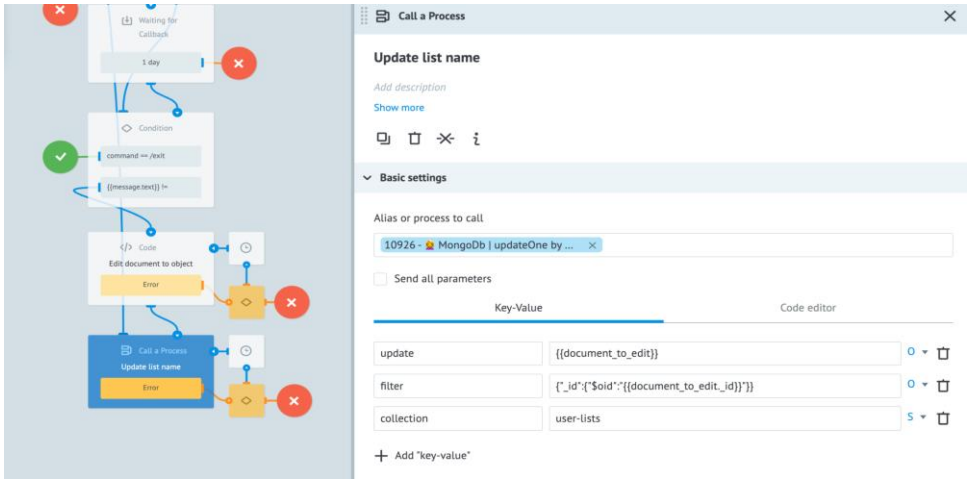

Рисунок 3.8– Використання Update методу для роботи із БД

3.2 Розробка базової логіки чат-боту

Базова логіка чат-боту включає у себе реєстрацію, логіку збереження та керування списками даних та роботу з пошуком.

Для початку налаштуємо маршрутизацію нових користувачів. Для цього, при невдалій спробі знайти даного користувача, створюємо його профіль і перенаправляємо на створений нами процес реєстрації (рисунок 3.9).

Рисунок 3.9 – Маршрутизація нового користувача на реєстрацію

У процесі реєстрації описуємо для користувача функціональні властивості чат-бота та просимо вказати ім'я, що буде використовуватись для взаємодії із користувачем та номер телефону, що буде використовуватись для зв`язки акаунтів користувача у різних месенджерах та забезпечення омніканальності (рисунок 3.10). Обидва питання можна пропустити, але тоді для юзера втрачається описана вище функціональність. Після кожного кроку, де користувач вводить дані, оновлюємо його профіль у Corezoid за допомогою блока Modify Task, що застосовується до процесу User Profile із параметром action, що має значення update. Цей крок важливий, оскільки він забезпечує цілісність даних навіть якщо користувач відволікся під час проходження реєстрації та не закінчив її.

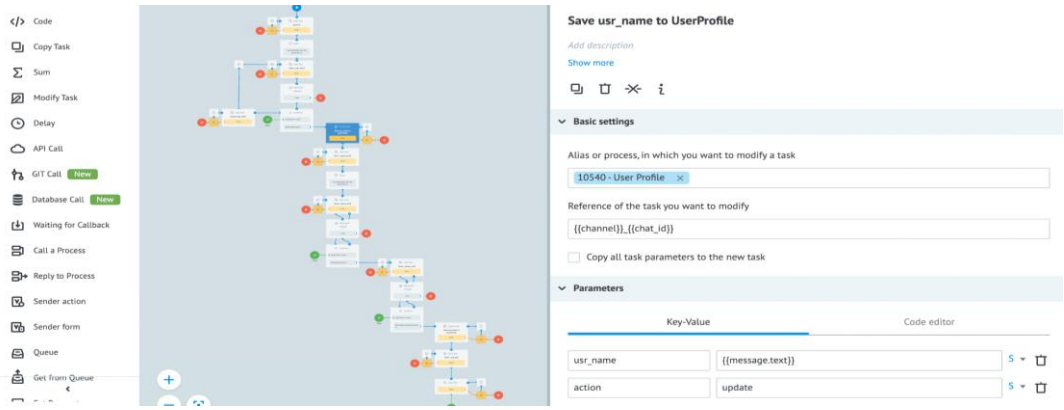

Рисунок 3.10 – Процес реєстрації користувача

Варто звернути увагу на діаграму User Profile в яку також були внесені зміни (рисунок 3.11).

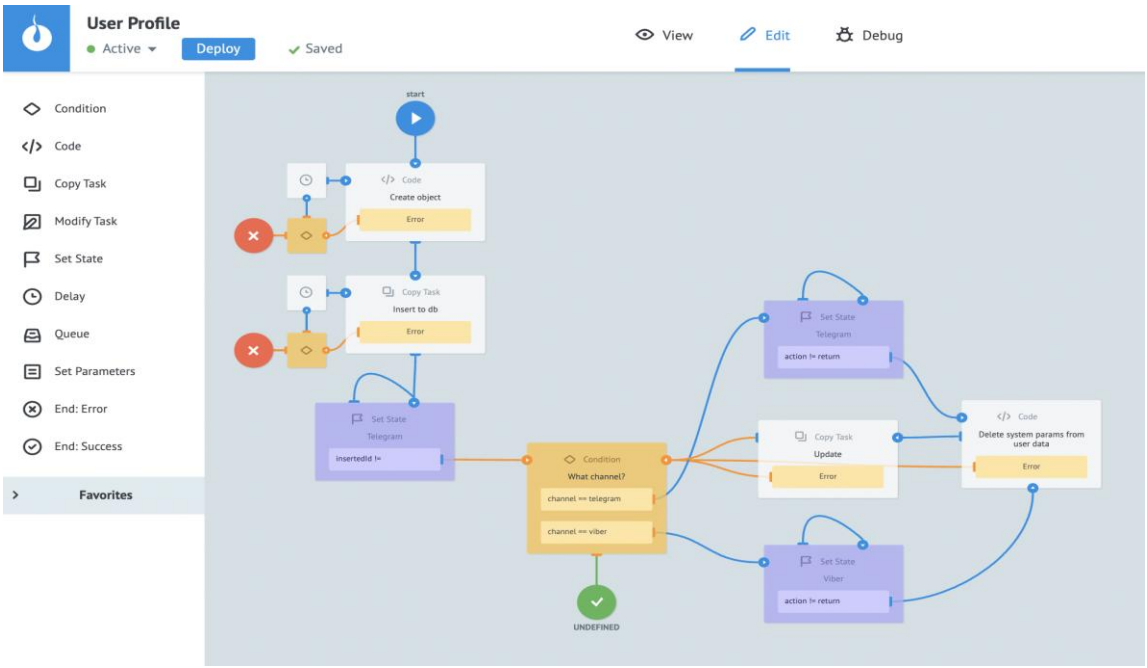

Рисунок 3.11 – Діаграма користувачів

На відміну від системної діаграми, що створюється під час розгортання бот платформи, було додано синхронізацію із базою даних, що відбувається за двох умов: створення нового користувача та коли ми передаємо action зі значенням update, про яку було згадано при описі реєстрації.

Крім реєстрації нових користувачів, кастомна маршрутизація потрібна і для більш зручної взаємодії користувача під час пошуку контенту. Для цього було створено новий процес під назвою /search, що буде використовуватись при виклику відповідної команди користувачем у чат-боті. Також, дана команда була заведена у діаграмі команд бот платформи.

Стандартна бот-платформа, коли отримує від користувача текст, відмінний від описаних команд, перенаправляє користувача на головне меню із текстом, що команда невідома. Для того, щоб користувач міг одразу писати пошуковий запит, без необхідності вводу команд, потрібно внести зміни у маршутизатор бот-платформи. Тому, якщо текст чи команда невідомі, ми перенаправляємо користувача на процес пошуку (рисунок 3.12).

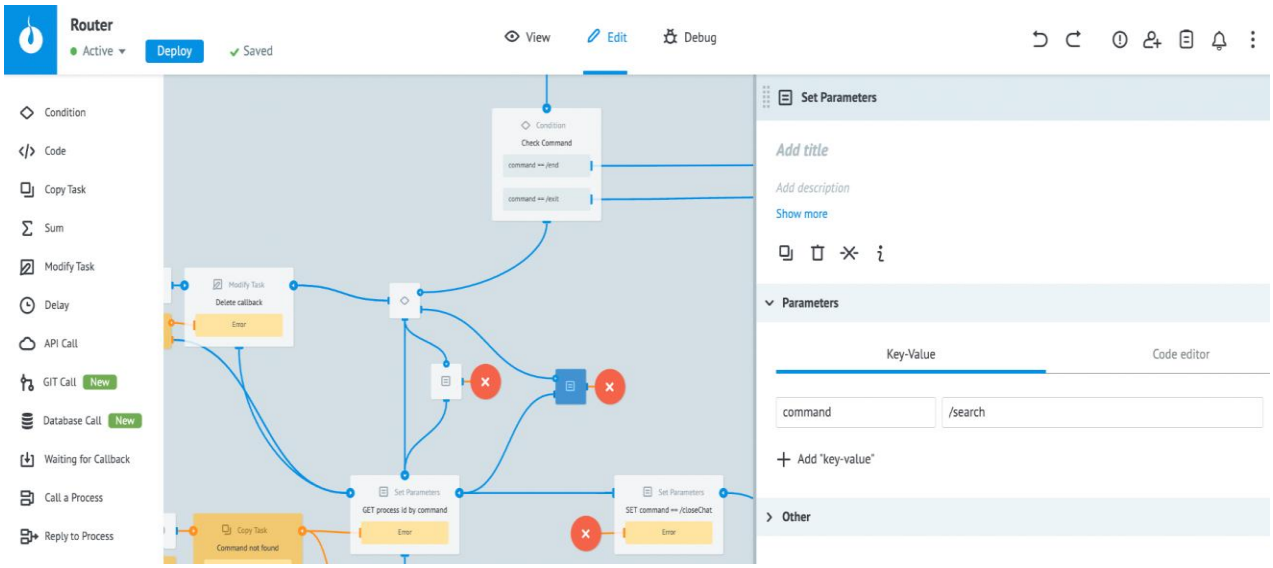

Рисунок 3.12 – Внесені зміни у процес-маршутизатор

Також, у користувача буде можливість перейти у процес пошуку (або будь-який інший) стандартним способом: ввівши команду чи натиснувши відповідну кнопку у головному меню.

3.3 Створення процесів для пошуку та обробки джерел інформації

Створення логіки пошуку можна розділити на три частини:

- Створення API-методів для пошуку контенту;
- Створення логіки для роботи із динамічним пошуком на основі API-методів відповідних месенджерів;
- Створення сценарію взаємодії користувача з чат-ботом;
- Обробка і збереження контенту в список.

Для пошуку контенту у даній роботі будуть використовуватись готові API-методи: YouTube API, Google Books API, The Movie Database (TMDB) API та Coursera API, оскільки на даному етапі їх більш ніж достатньо, але у майбутньому можна буде розширити власними базами даних контенту чи парсерами, що в може значно покращити пошукові здібності чат-бота.

YouTube API, Google Books API належать компанії Google, для роботи з ними потрібно розгорнути відповіді екземпляри із API Library на Google Cloud Platform (рисунок 3.13). Після створення потрібно зберегти API ключі, оскільки вони будуть використовуватись для запитів.

Рисунок 3.13 – Розгорнуті API на Google Cloud Platform

The Movie Database (TMDB) API та Coursera API постачаються відповідними платформами, для отримання API url та ключа достатньо пройти реєстрацію на платформі.

Після того, як ми отримали всі необхідні дані для запитів переходимо до створення процесів. Кожен, описаний вище запит, створюється в окремому процесі у вигляді функції, що буде викликатись під час пошуку за допомогою конструкції Call Process і буде складатись з наступних частин (рисунок 3.14), де:

1. Валідація даних які отримуватиме процес під час його виклику;
2. Отримання API-key із діаграми;
3. Підготовка запиту у відповідному до документації форматі;
4. Сам запит;
5. Обробка отриманої відповіді та парсинг у стандартизований для інформаційної системи формат;
6. Повернення блоку Call Process отриманих даних.

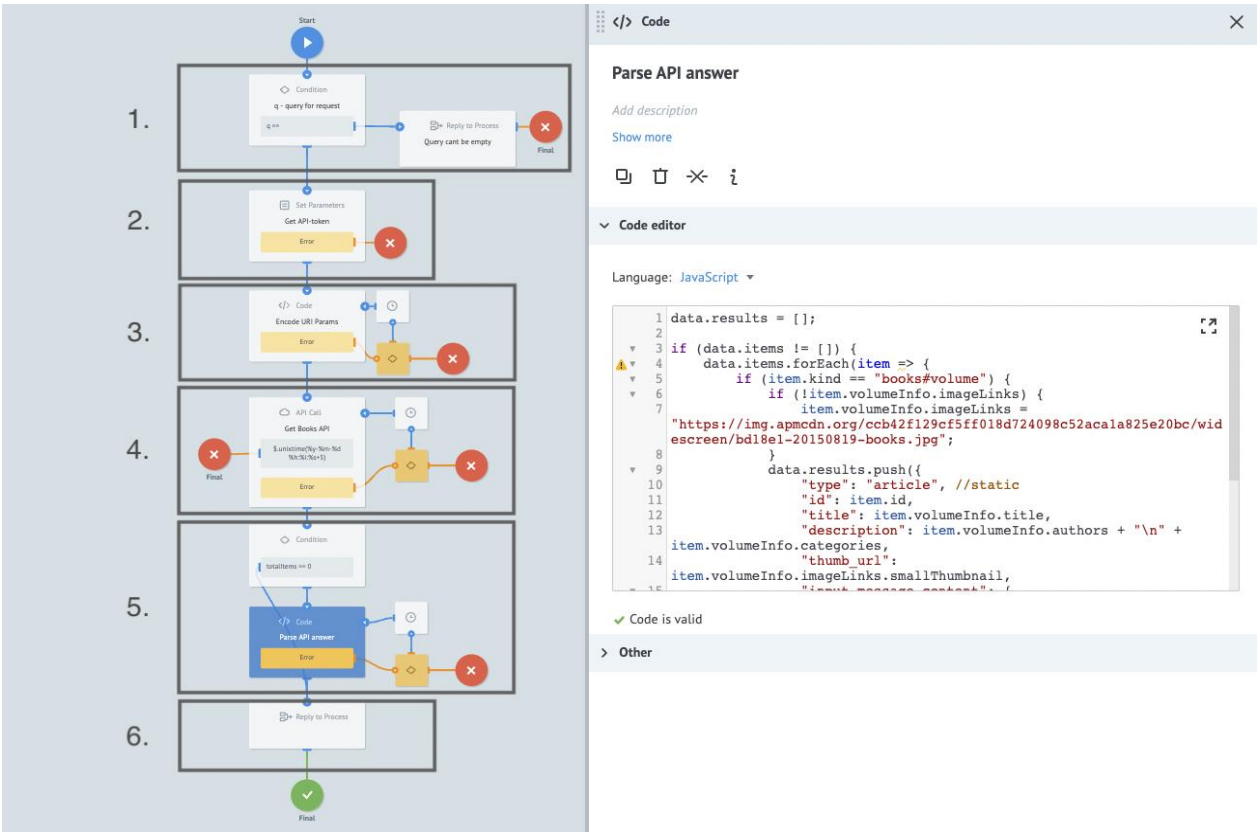

Рисунок 3.14 – Розгорнуті API на Google Cloud Platform

Для роботи із динамічними списками у процеси Receiver відповідних месенджерів був доданий новий формат відповіді — inline_query. Він забезпечує формування списку відповідей на повідомлення користувача і динамічно оновлюється коли користувач пише свій запит, без потреби надсилати повідомлення. Працює він наступним чином: як тільки користувач зупиняється під час писати повідомлення, месенджер відправляє на вебхук написане. У відповідь на це, за допомогою відповідного API методу месенджера, ми можемо надіслати відповідь у вигляді динамічного списку. Якщо користувач продовжить писати повідомлення, описаний вище алгоритм спрацює знову.

Для відповіді на такий тип повідомлення було створено процес, що отримує запит на вхід та надсилає динамічну відповідь. Виклик даного процесу відбуватиметься щоразу, як користувач припинив писати повідомлення. Його можна розділити на наступні частини (рисунок 3.15):

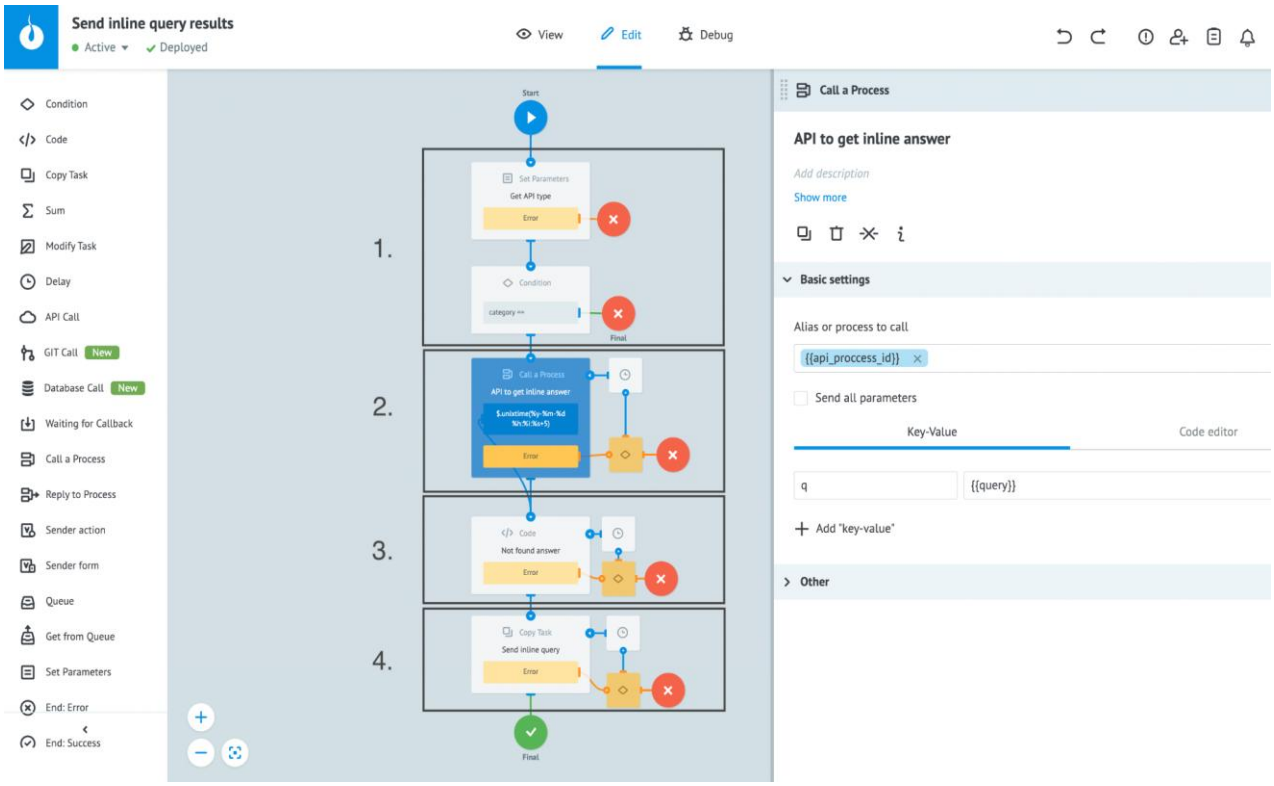

Рисунок 3.15 – Процес формування динамічної відповіді

де:

1. Отримуємо тип запиту, який потрібно зробити;
2. Викликаємо відповідний процес та передаємо туди текст запиту;
    2.1. Викликаний процес формує список відповідей у потрібному форматі;
3. До отриманого списку додаємо стандарту відповідь на випадок, якщо ні одна із знайдених не підходить користувачу;
4. Відправляємо запит на відповідний API-метод месенджера.

Основні корові елементи пошуку були описані вище, тому на даному етапі були залишилось об'єднати їх у процесі /search, куди була налаштована маршрутизація та прописати діалоги і кнопки для інтерфейса користувача. Готовий процес має наступний вигляд (ДОДАТОК А)

На рисунку 3.16 зображено першу частину даної логіки. Тут ми перевіряємо яким чином потрапив користувач у процес: використовуючи відповідну

команду чи ми перенаправили його з головного меню коли він ввів пошуковий запит. Якщо це перший варіант, то ідемо в ліву гілку логіки, де відправляємо повідомлення користувачу з проханням надіслати пошуковий запит. Після чого, для обох варіантів записуємо його в окрему змінну, щоб можна було використовувати його пізніше.

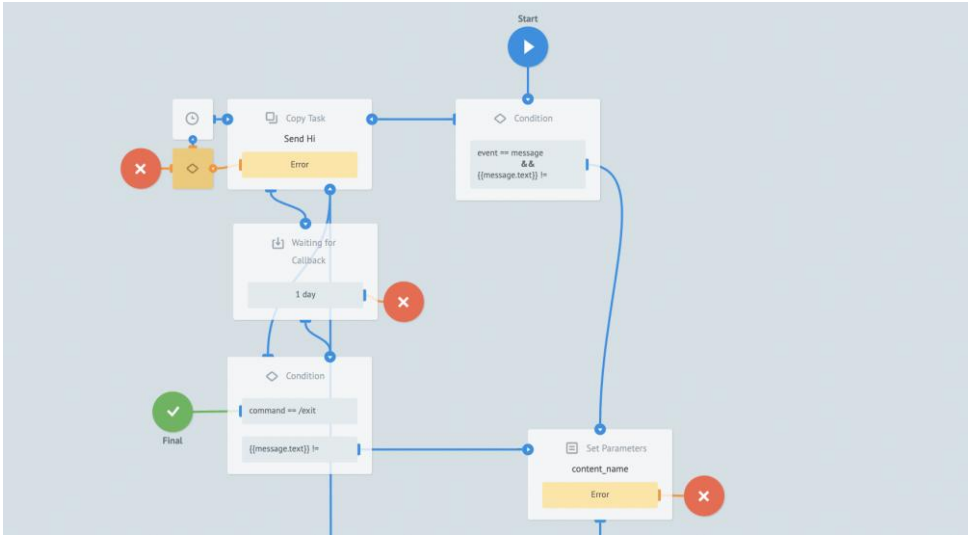

Рисунок 3.16 – Перша частина логіки пошуку

У другій частині просимо обрати категорію, щоб знати яку апі використовувати та налаштовуємо параметри для пошуку. Після цього надсилаємо кнопку із динамічним пошуком (рисунок 3.17)

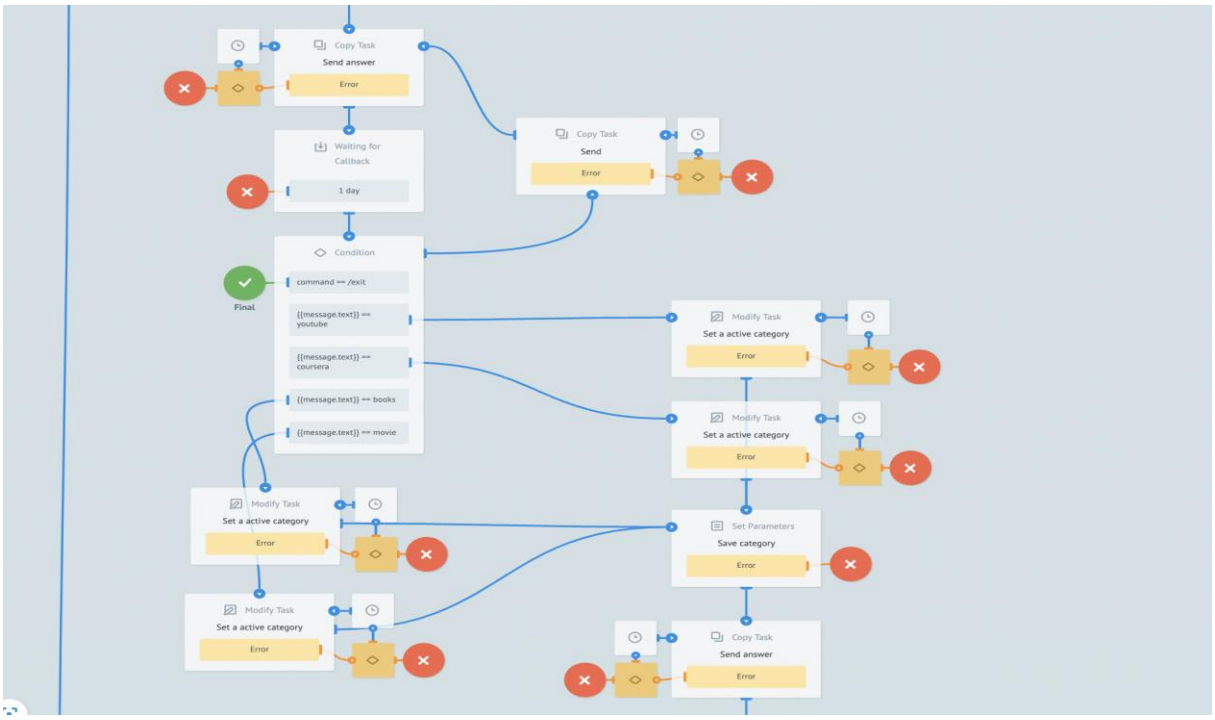

Рисунок 3.17 – Друга частина логіки пошуку

На рисунку 3.18 можна побачити інтерфейс динамічного пошуку у телеграм, де відображаються результати пошуку запиту “learn python” в категорії YouTube.

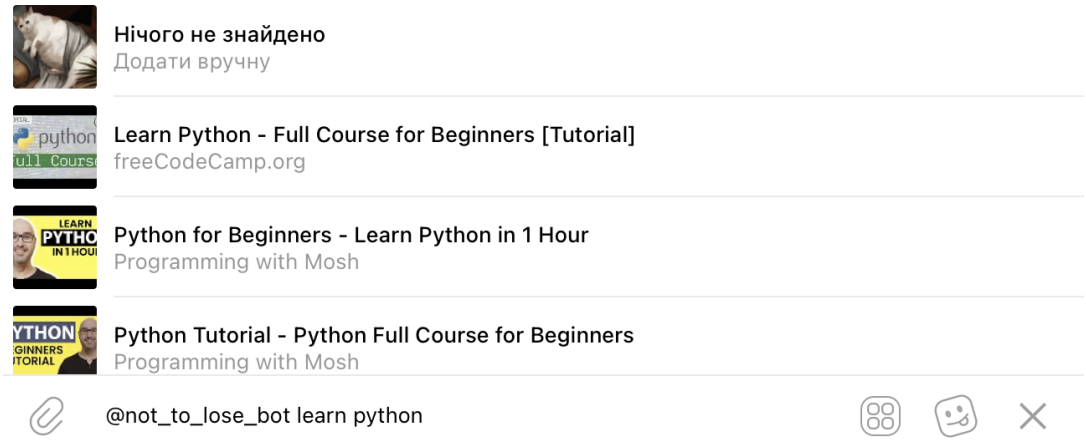

Рисунок 3.18 – Динамічний пошук в Телеграм

Окрім відповідей від API можна побачити стандартну відповідь “Нічого не знайдено, додати вручну”, що дає можливість ввести дані самостійно. У третій частині, після того, як користувач обрав потрібний йому об`єкт із списку, приводимо його у зручний формат та даємо доступ до функціоналу обробки цих даних для формулювання фінального списку (рисунок 3.19)

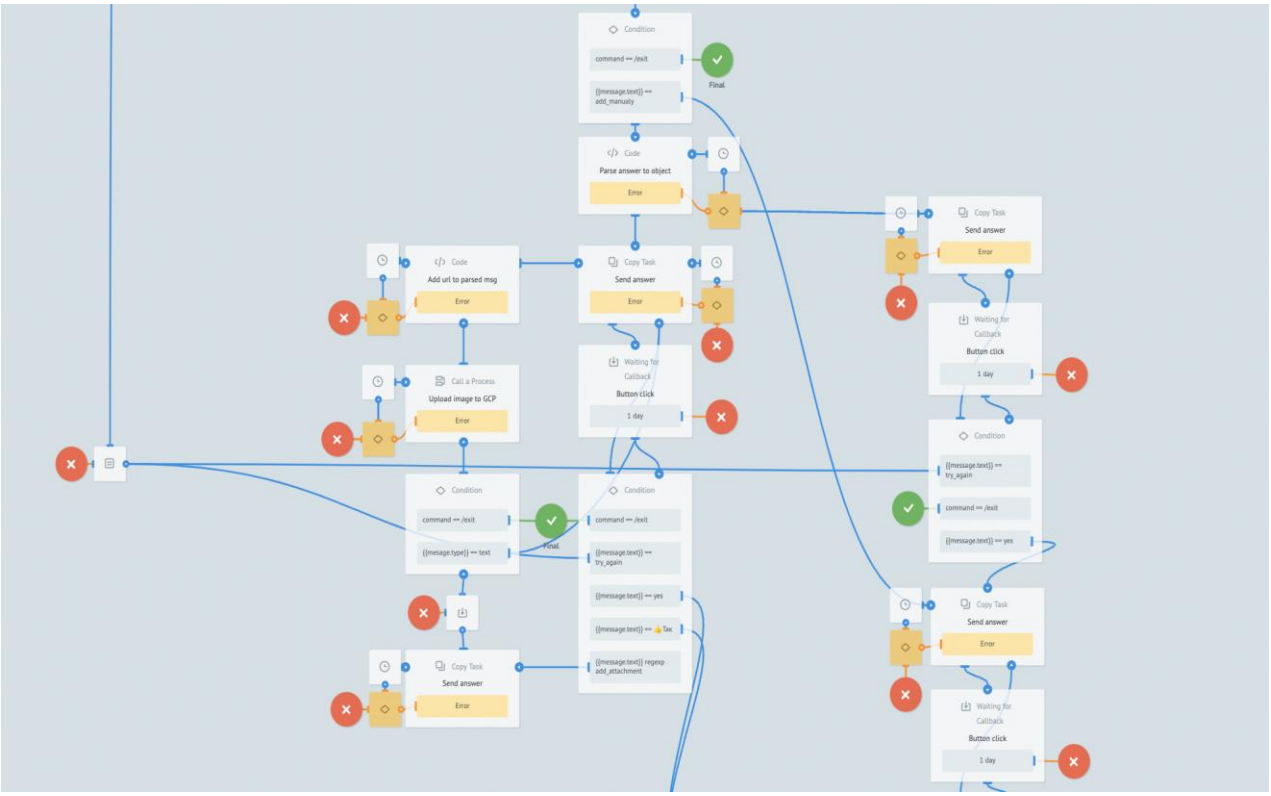

Рисунок 3.19 – Третя частина логіки пошуку

Після цього, надсилаємо клавіатуру для керування об’єктом (рисунок 3.20)

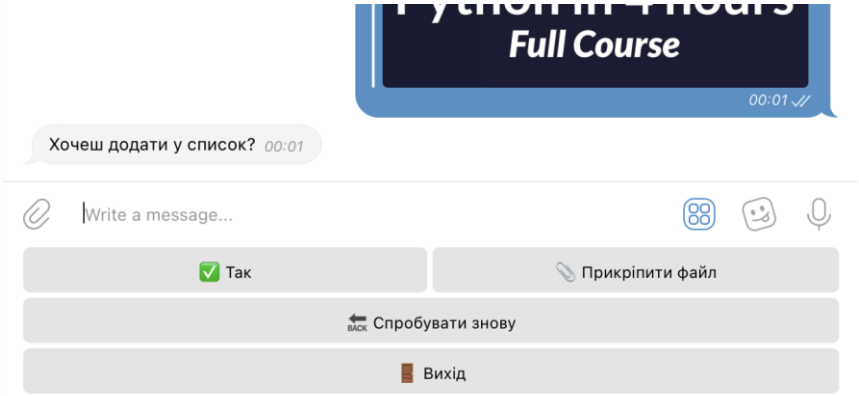

Рисунок 3.20 – Меню керування результатами пошуку

Далі перенаправляємо користувача на процес керування списками, детальна робота якого буде описана у наступному розділі. На рисунку 3.21 продемонстрований інтерфейс для вибору списка у який ми хочемо додати знайдений контент.

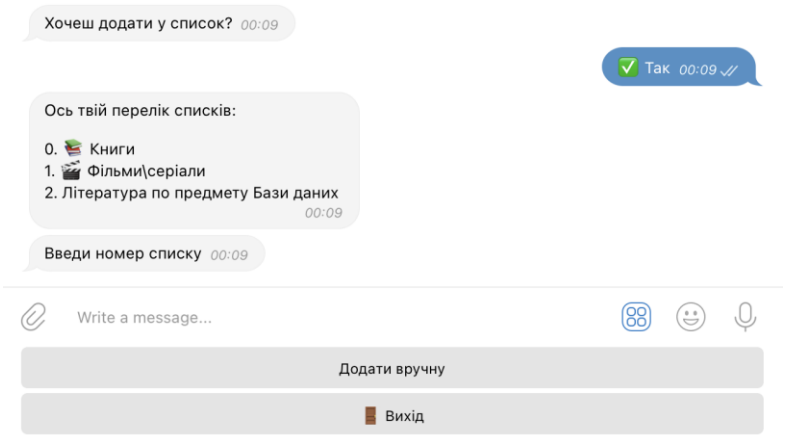

Рисунок 3.21 – Додавання знайденого у список

3.4 Створення API для роботи з файлами на базі Python Flask API

Оскільки більшість месенджерів зберігають файли лише певний період (наприклад Viber) доцільніше буде зберігати їх у власному сховищі, що гарантуватиме їх цілісність та доступність. Найпростішим способом є

створити власний Flask API Post метод (ДОДАТОК Б), що буде отримувати посилання на файл, завантажувати його у наше сховище та віддавати публічний лінк на нього. Для забезпечення безпечної роботи API було додано Bearer аутентифікацію по токену та обмеження у вигляді максимального розміру файлу.

Токен зберігатиметься у змінному середовищі, оскільки воно надає чудовий спосіб налаштувати програму на Python, усуваючи необхідність редагувати вихідний код під час зміни конфігурації.

Елементи конфігурації, які часто передаються до програми через змінні середовища, — це ключі API, мережеві порти, сервери баз даних та будь-які користувацькі параметри. Нижче представлений запит без авторизації (рисунок 3.22).

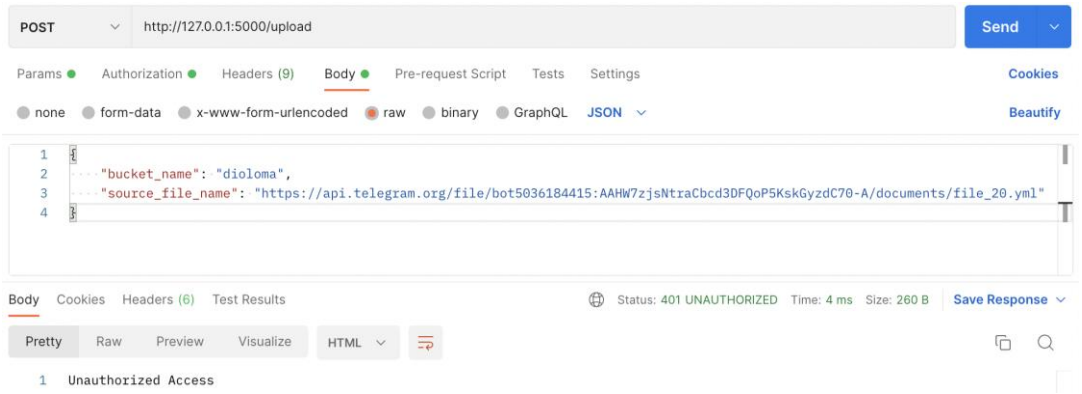

Рисунок 3.22 – Запит без авторизації

Якщо при запиті обрати аутентифікацію та передати правильний токен, то отримаємо відповідь з посиланням на файл (рисунок 3.23).

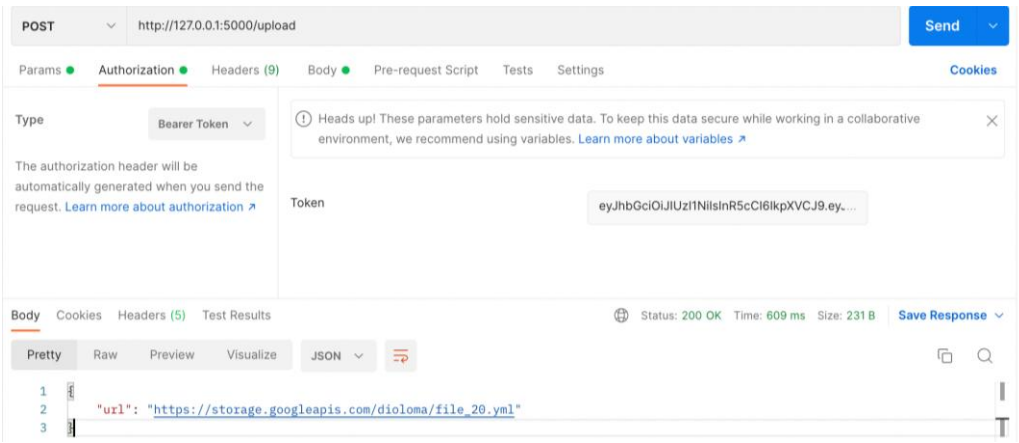

Рисунок 3.23 – Запит з авторизацією

Для того, щоб мати доступ до API із Corezoid, його потрібно розгорнути на хостингу. Для цього був обраний App Engine (рисунок 3.24).

```
Creating App Engine application in project [diplomakari] and region [europe-west]....done.
Services to deploy:

descriptor:                  [/Users/kari/Desktop/images/app.yaml]
source:                      [/Users/kari/Desktop/images]
target project:              [diplomakari]
target service:              [default]
target version:              [20220504t002008]
target url:                  [https://diplomakari.ew.r.appspot.com]
target service account:      [App Engine default service account]


Do you want to continue (Y/n)?  y

Beginning deployment of service [default]...
Created .gcloudignore file. See `gcloud topic gcloudignore` for details.
╔═ Uploading 6 files to Google Cloud Storage ═╗
File upload done.
Updating service [default]...
```

Рисунок 3.24 – Розгортання Flask Api для збереження файлів

Створений процес для роботи з API на Corezoid виглядає наступним чином (рисунок 3.25):

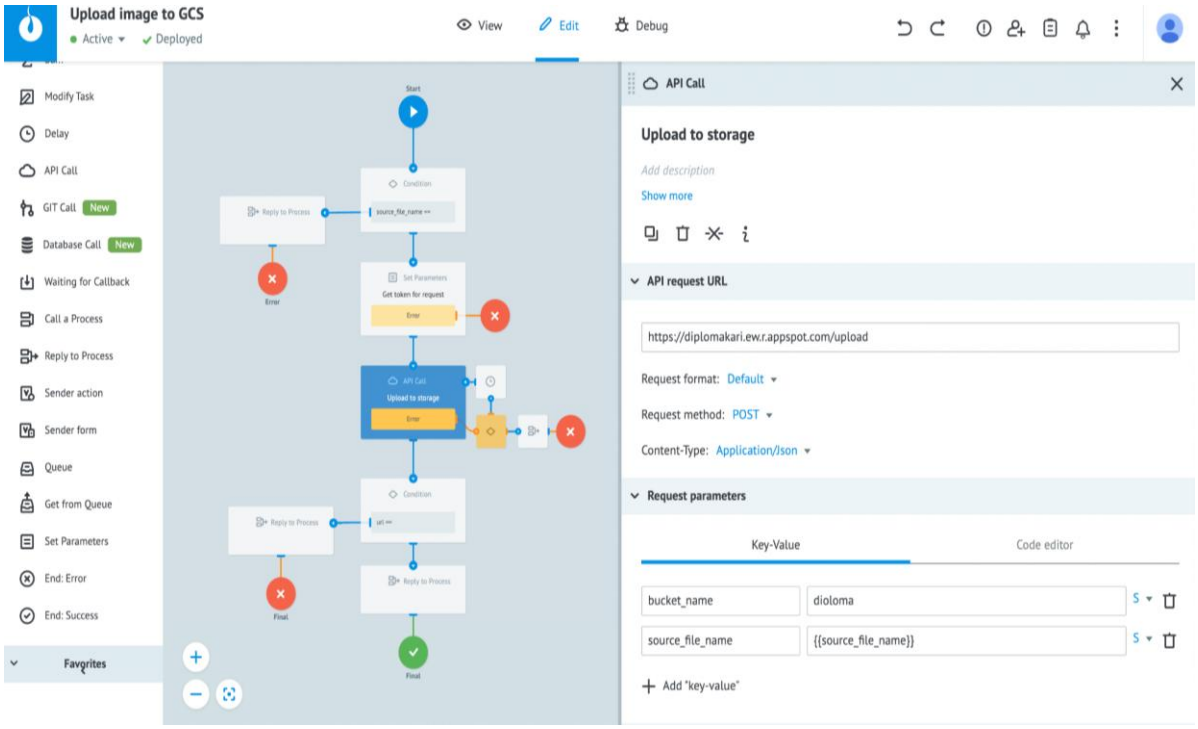

Рисунок 3.25 – Розгортання Flask Api для збереження файлів

Тепер наявний процес можна використовувати для збереження будь-яких файлів розміром до 10 мегабайт. Даний функціонал буде доступний на етапі підтвердження додавання об’єкта в базу у кнопці “Прикріпити файл”.

3.5 Створення процесів для керування списками користувача

Створення процесів для керування списками користувача можна розділити на основний функціонал та додатковий. Основний функціонал складається із базових функцій таких, як створення, редагування, додавання у список, видалення списку і т д та працює наступним чином (рисунок 3.26). Користувач потрапляє у процес керування списками. По user_id знаходимо всі списки користувача та виводимо їх у чат-бот. Якщо у користувача ще немає списків, то пропонуємо додати новий або скористатись заготовленеми.

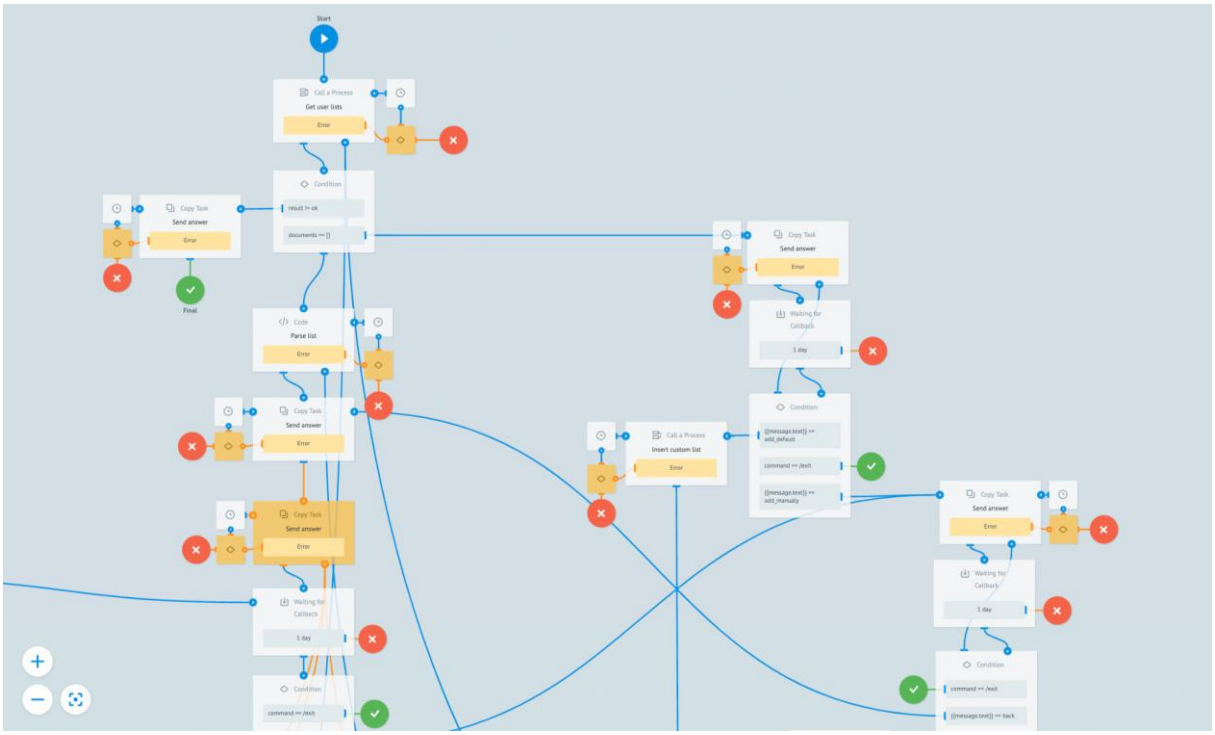

Рисунок 3.26 – Пошук та формування списків користувача

Після цього пропонуємо користувачу обрати список, з яким він буде взаємодіяти, ввівши його номер (рисунок 3.27)

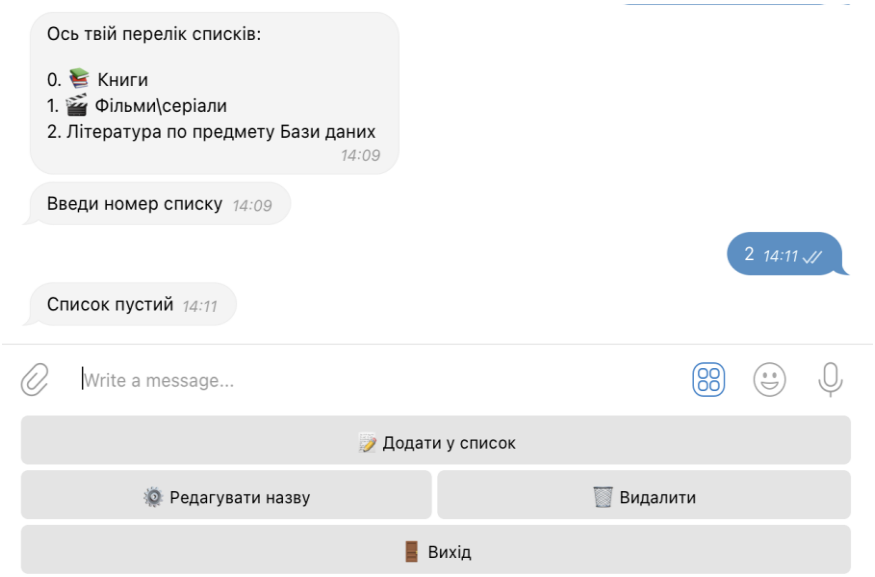

Рисунок 3.27 – Вибір списку користувачем

Якщо при роботі з списками ми отримали параметр parsed_msg, це означає що користувач прийшов із процесу пошуку, отже, після вибору списка користувачем потрібно потрібно зберегти знайдений об’єкт, а вже після цього надавати можливість керувати списком (рисунок 3.28).

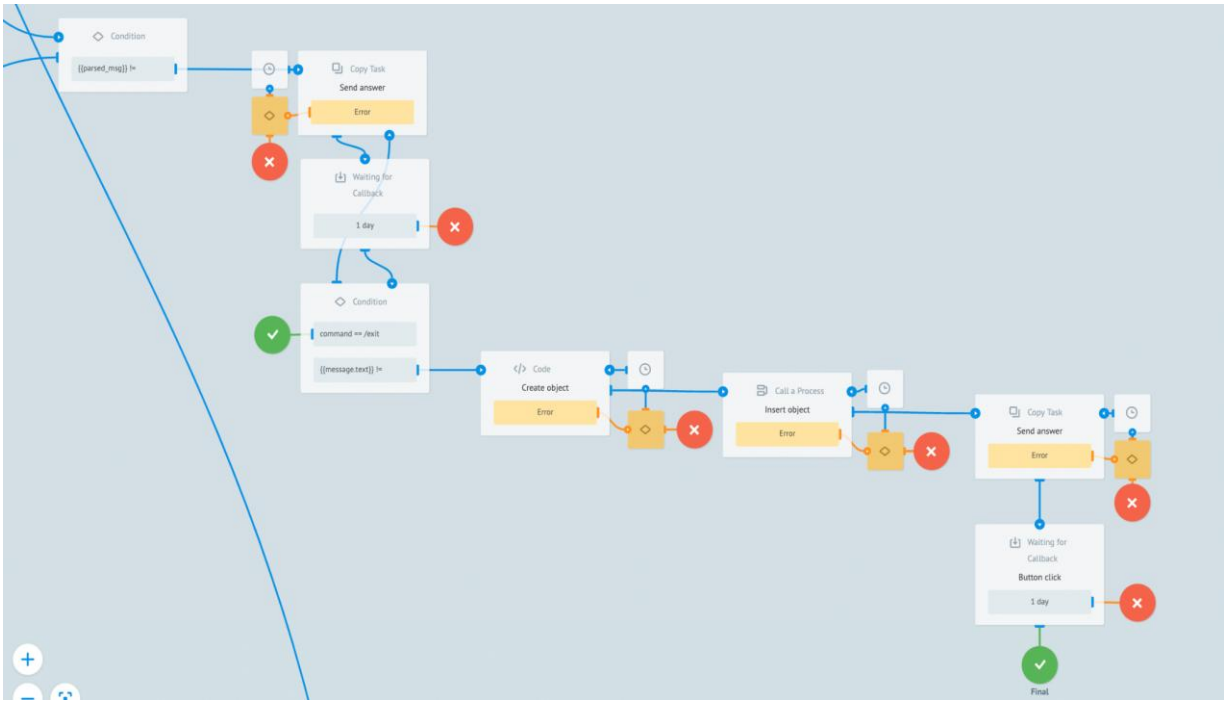

Рисунок 3.28 – Додавання знайденого в обраний список

Після цього стає доступним функціонал для керування доданим у список (рисунок 3.29).

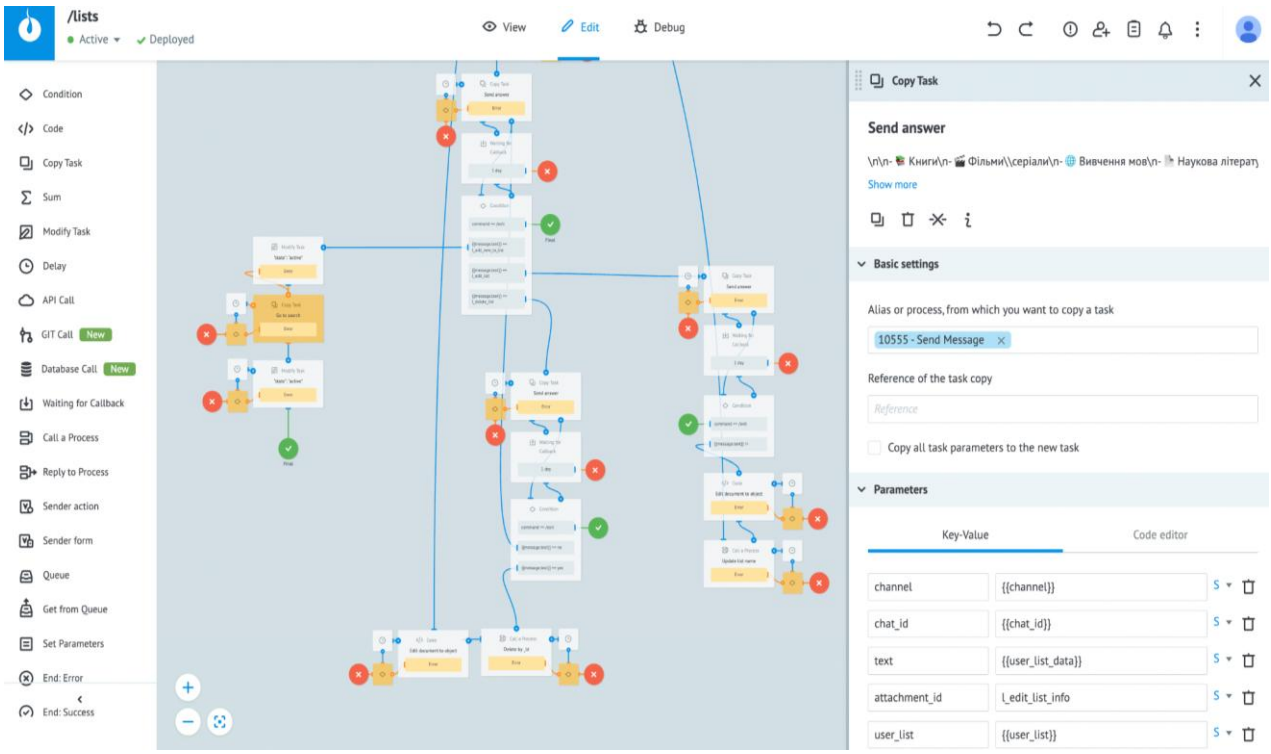

Рисунок 3.29 – Обробка списків відповідно до існуючого функціоналу

Крім базових можливостей доступний такий функціонал, як можливість поділитись списком із іншим зареєстрованим користувачем або колесо фортуни на базі React та Telegram Web App API, що використовується для вибору випадкового об’єкту у обраному списку (рисунок 3.30).

Рисунок 3.30 – Telegram Web App вибору випадкового об’єкту

Для того, аби поділитись потрібно обрати відповідну кнопку та надіслати отримане повідомлення із списком іншому користувачу. Якщо він захоче додати цей список собі, потрібно просто надіслати його чат-боту. По айді списку будуть знайдені усі збережені довідкові джерела та додані до акаунту відповідного користувача. На рисунку 3.31 представлена готова частина даної логіки.

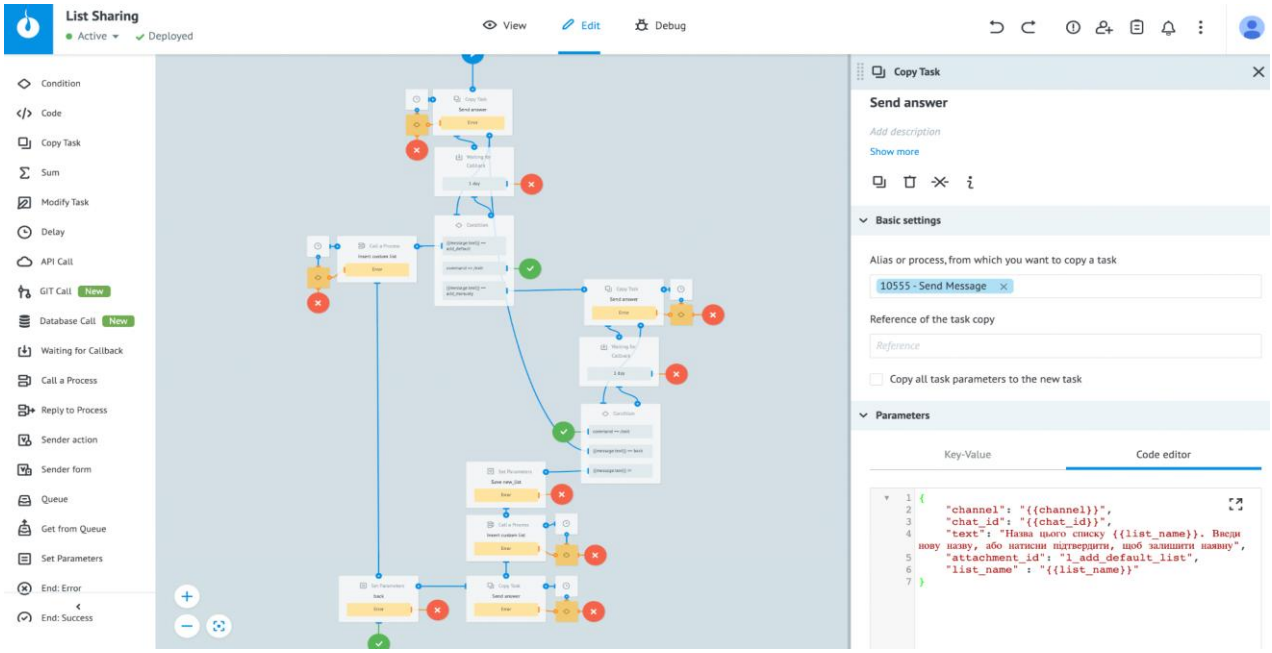

Рисунок 3.31 – Логіка для функціоналу “Поділитися списком”

Готовий процес керування списками представлений у Додатку В.

3.6 Висновки до розділу

У даному розділі було налаштовано середовище для розробки та створено інформаційну систему на його базі. Для налаштування було розгорнуто Corezoid Bot Platform та створено і підключено обрану у першому розділі базу даних.

Для роботи з Messengers API було зареєстровано чат-боти у месенджерах Viber та Telegram для отримання токену, що використовується для API запитів.

Розробка базової логіки чат-боту включає у себе створення реєстрації, логіки збереження даних та роботу з пошуком. Для їх реалізації було модифіковано корові процеси Corezoid Bot Platform та створено нові. Окрім цього платформа була розширена API методами месенджерів що доступні для роботи, але не були представлені у базовій логіці платформи.

Для створення процесів для пошуку та обробки джерел інформації було знайдено та проаналізовано API таких платформ як YouTube API, Google Books API, The Movie Database (TMDB) API та Coursera API. Після цього вони були розгонуті на Google Cloud Platform та налаштовані для подальшої роботи. Це дозволило провести інтеграцію Corezoid та цих сервісів та забезпечити користувачу можливість пошуку відео, книг, фільмів, серіалів та курсів.

Для того, щоб розширити перелік доступного для збереження контенту була додана можливість вводу інформації вручну та прикріплення файлів. Останнє із них забезпечується завдяки API, написаному на Flask, що зберігає файли на Google Cloud Storage. Описаний вище функціонал та додаткові елементи системи забезпечують користувачу зручний спосіб взаємодії із інформаційною системою.

ВИСНОВОК

У результаті виконання дипломної роботи здійснене дослідження та порівняння сучасних методів реалізації додатків та інформаційних систем, особливу увагу було приділено технології Low-Code та платформам що забезпечують можливість роботи із цими технологіями. Був проаналізований та обґрунтований вибір оптимальної технології для реалізації обраної системи, здійснена розробка серверної та клієнтської частини системи на базі Corezoid, Messengers Chat-Bot API та MongoDb.

До створюваної системи були поставлені чіткі функціональні вимоги, які були повністю реалізовані за результатами дипломної роботи:

1. Система дозволяє зберігати та керувати різними видами інформації та контентом, знайденим в інтернеті.
2. Система веде облік та надає доступ до створених списків, а також має додатковий функціонал для взаємодії із ними.
3. Система надає функціонал пошуку та структуризації знайдених даних.
4. Система реалізує окремий файловий сервіс для збереження будь-яких файлів.
5. Система дозволяє ділитись наявними списками та додавати собі уже створені.

У першому розділі розглянуто основні технології для розробки інформаційних систем та методи їх реалізації: найпопулярніші методи це веб-сайт, настільний додаток та чат-бот. Перевагами останнього є зручний доступ до чат-бота із будь-якого пристрою, де встановлений один із месенджерів. Користувач матиме змогу у будь-який момент додати нове джерело чи книгу у список літератури, а також поділитись ним із іншими у тому ж месенджері. Основні переваги:

- діалог – основна форма спілкування;

- уніфікований інтерфейс (поле для введення питання та історія листування);
- можливість відповіді за допомогою кнопок (без набору тексту).

Встановлено, що існують основні методи реалізації інформаційних систем: звичайне кодування (pro-code), low-code, no-code, проаналізовано особливості кожного із них. Проведення дослідження було здійснено за допомогою аналізу кожного типу та особистого досвіду створення чат-ботів кожним із методів. Вибір low-code платформи базувався на аналізі найпопулярніших представників за їх призначеням, особливостями та методологіями розробки. Варто зазначити що у кожної платформи є так звана “філософія” якій варто слідувати щоб створювати оптимальні рішення. Corezoid чудово підходить для створення чат-ботів завдяки готовому фреймворку на його базі під назвою Corezoid Bot Platform та простотою інтеграцій на основі API. База даних була обрана під обрані технології для забезпечення зручності в інтеграції та використанні. Оскільки месенджери працюють на базі API JSON типу та задача у Corezoid має такий же формат, то оптимальним рішенням є вибір Nosql база даних, як ось MongoDb.

У другому розділі була розглянута платформа Corezoid, описана та проаналізовала модель використання, модель предметної області та запуску процесу, що фактичною являється реалізацією бізнес-процесу. У подальшому були зазначені основні функціональні блоки платформи, та терміни, що необхідні для базового розуміння даної платформи. Було визначено, що Corezoid — це відносно нова оригінальна багатоцільова хмарна платформа для розробки для обслуговування, управління, виконання, моніторингу та оптимізації програмного забезпечення для автоматичних або автоматизованих бізнес-процесів яка слідує сучасній концепції Platform as a Service, що забезпечує найвищий ступінь паралельності та масштабованості.

У третьому розділі здійснений процес декомпозиції системи на функціональні складові та виконані підготування для роботи: розгорнуто Corezoid Bot Platform, створено чат-боти в обраних месенджерах, підключено

токени до платформи, створена та підключена база даних, розроблені базові API методи для інгерації із MongoDb. Розробка базової логіки чат-боту включає у себе реєстрацію, логіку збереження та керування списками даних та роботу з пошуком. На цьому етапі були створені: процеси для пошуку та обробки джерел інформації, API для роботи з файлами на базі Python Flask API, процеси для керування списками користувача, проведена інтеграція із такими платформами, як Google Books, YouTube, Coursera, The Movie Database (TMDB). Була розширена Corezoid Bot Platform таким функціоналом, як динамічний пошук, автоматична маршрутизація у процес пошуку та доданий функціонал Telegram Web App, що з'явився під час написання даної дипломної роботи.

Остаточним результатом роботи стало проведене дослідження технологій для створення інформаційних систем, а також створення системи керування чат-ботом, що базується на Low-Code платформі Corezoid.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Енциклопедія PCMag, визначення поняття “Настільний додаток” [електронний ресурс], - Режим доступу: https://www.pcmag.com/encyclopedia /term/desktop-application

2. Техопедія, визначення поняття “Мобільний додаток” [електронний ресурс], - Режим доступу: https://www.techopedia.com/definition/2953/mobile-application -mobile-app

3. Вікіпедія, визначення поняття “Чат - бот” [електронний ресурс], - Режим доступу: https://en.wikipedia.org/wiki/Chatbot

4. Класифікація мов програмування [електронний ресурс], - Режим доступу: https://studref.com/509365/informatika/klassifikatsiya_yazykov_programmirova niya

5. Мова програмування високого рівня[електронний ресурс], - Режим доступу: https://www.wiki.uk-ua.nina.az/Високорівнева_мова_програмування.html

6. Платформи розробки з низьким кодом і без коду [електронний ресурс], - Режим доступу: https://searchsoftwarequality.techtarget.com/definition/low-code -no-code-development-platform

7. Різниця між IAAS, PAAS та SAAS [електронний ресурс], - Режим доступу: https://www.geeksforgeeks.org/difference-between-iaas-paas-and-saas/

8. Що таке платформа додатків як послуга [електронний ресурс], - Режим доступу: https://www.outsystems.com/glossary/what-is-application-platform-as-a-service/

9. Історія платформ з низьким кодом [електронний ресурс], - Режим доступу: https://kissflow.com/low-code/history-of-low-code-development-platforms/

10. Знайомство з YAML і JSON [електронний ресурс], - Режим доступу: https://www.educba.com/yaml-vs-json/?source=leftnav

11. Що таке ROI та як цей показник допоможе власникові бізнесу? [електронний ресурс], - Режим доступу: https://bakertilly.ua/news/id49125

12. 7 плюсів і мінусів Low-Code/No-Code [електронний ресурс], - Режим доступу: https://blog.hslu.ch/majorobm/2021/04/05/7-pros-and-cons-of-low-code-no-code-ntsy-2-ua-192667621-1/

13. Визначення поняття Citizen Developer [електронний ресурс], - Режим доступу: https://www.interfacing.com/citizen-developers-meaning-definition-low-code-no-code-development

14. Популярність Low-Code платфлом у 2021 році [електронний ресурс], - Режим доступу: https://www.gartner.com/en/newsroom/press-releases/2021-02-15-gartner-forecasts-worldwide-low-code-development-technologies-market-to-grow-23-percent-in-021

15. Платформи з низьким кодом [електронний ресурс], - Режим доступу: https://www.forrester.com/blogs/category/low-code-platforms/

16. Платформа OutSystems [електронний ресурс], - Режим доступу: https://www.outsystems.com/

17. Платформа Mendix [електронний ресурс], - Режим доступу: https://www.mendix.com/

18. Платформа Microsoft Power Platform [електронний ресурс], - Режим доступу: https://powerplatform.microsoft.com/ru-ru/

19. Платформа Appian [електронний ресурс], - Режим доступу: https://appian.com/

20. Платформа Oracle Apex [електронний ресурс], - Режим доступу: https://apex.oracle.com/en/

21. Платформа Corezoid [електронний ресурс], - Режим доступу: https://corezoid.com/

22. MySQL Documentation [електронний ресурс], - Режим доступу: (https://dev.mysql.com/doc/)

23. Документація к Postgres Pro Standard 13.4.1 [електронний ресурс], - Режим доступу: https://postgrespro.ru/docs/postgrespro/14/

24. А.В. Філіпенков, Е.Л. Кузьмін “Порівняння існуючих систем управління базами даних з метою вибору кращої при реалізації вимог щодо скорочення витрат та імпортозаміщення”, 2018, 6с

25. Функціонал платформи Coreziod [електронний ресурс], - Режим доступу: https://doc.corezoid.com/docs/release-notes-551

26. Огляд Corezoid Process Engine [електронний ресурс], - Режим доступу: https://corezoid.com/blog/corezoid-process-engine-overview/

27. Початок роботи із платформою Corezoid [електронний ресурс], - Режим доступу: https://doc.corezoid.com/docs/quick-start

28. Глосарій платформи Corezoid [електронний ресурс], - Режим доступу: https://doc.corezoid.com/docs/glossary

ДОДАТОК А

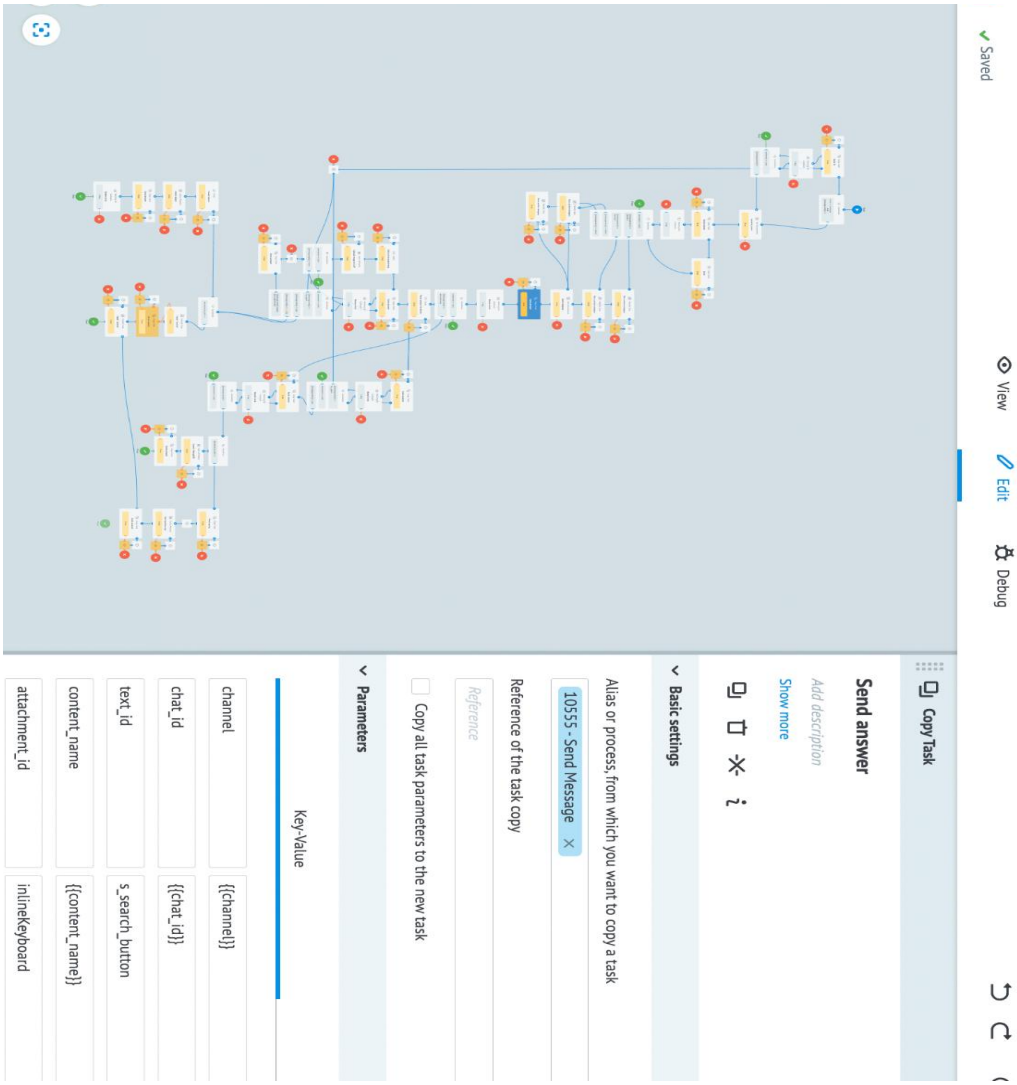

Рисунок А.1 – Готова логіка для пошуку контенту користувачем

ДОДАТОК Б

```
from google.cloud import storage
import os
from flask import Flask, request
from flask_httpauth import HTTPTokenAuth

app = Flask(__name__)
auth = HTTPTokenAuth(scheme='Bearer')

API_TOKEN = os.environ.get('API_TOKEN')

@auth.verify_token
def verify_token(token):
    if token == API_TOKEN:
        return token

@app.route('/upload', methods=['POST'])
@auth.login_required
def index():
    request_data = request.get_json()

    bucket_name = request_data['bucket_name']
    source_file_name = request_data['source_file_name']

    fsouse =  source_file_name.split("/")[-1]
    filename = requests.get(source_file_name)

    open("/tmp/" + fsouse, "wb").write(filename.content)
    stats = os.stat("/tmp/" + fsouse)
    print(stats.st_size)

    if stats.st_size > 10000000:
        os.remove("/tmp/" + fsouse)
        return { "status" : 500, "desctiption" : "max file 10 MB"}

    storage_client = storage.Client.from_service_account_json(
        'creds.json')

    bucket = storage_client.get_bucket(bucket_name)
    blob = bucket.blob(fsouse)
    # blob.upload_from_filename(fsouse)
    os.remove("/tmp/" + fsouse)
    return { "status" : 200, "url" : blob.public_url}

if __name__ == '__main__':
   app.run()
```

Рисунок Б.1 – Flask Api для збереження файлів

ДОДАТОК В

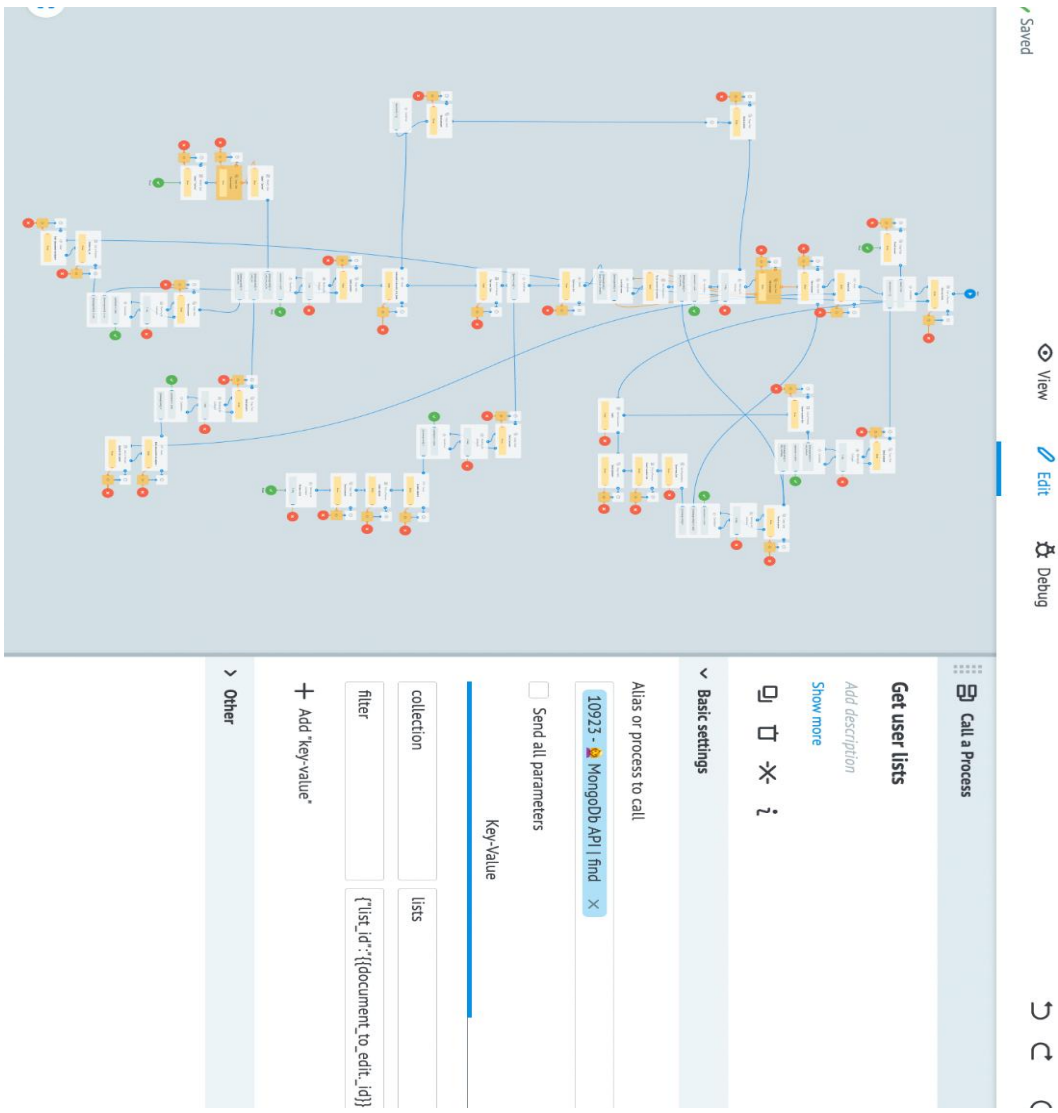

Рисунок В.3 – Готова логіка для пошуку контенту користувачем