Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Розробка конструктора телеграм-ботів з ДСА-керованою поведінкою

Текстова частина до курсової роботи
за спеціальністю «Комп'ютерні науки» - 122

Керівник курсової роботи
к-т фіз.-мат. наук, доцент
Гулаєва Н.М.

_____
(Підпис)
"___" _____ 2022 року

Виконав студент МП КН-1
Кобєлєв М. Д.
"___" _____ 2022 року

Київ 2022

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики технологій факультету інформатики

ЗАТВЕРДЖУЮ
Викладач кафедри інформатики,
к-т фіз.-мат. наук, доцент
_____ Гулаєва Н.М.

„____” _____ 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студенту Кобєлєву Михайлу Дмитровичу
факультету інформатики 1 курсу магістерської програми

ТЕМА: **Розробка конструктора телеграм-ботів з ДСА-керованою поведінкою**

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
Розділ 1. Огляд методів класифікації та побудови чат-ботів.
Розділ 2. Проектування та розробка застосунку-конструктора ботів.
Розділ 3. Демонстрація можливостей застосунку.
Висновки.
Список літератури
Додатки (за необхідністю)

Дата видачі „___” _____ 2022 р. Керівник_____
(підпис)
Завдання отримав _____
(підпис)

Тема: **Розробка конструктора телеграм-ботів з ДСА-керованою поведінкою**

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	10.03.2022	
2.	Огляд літератури за темою роботи.	11.03.2022	
3.	Проектування архітектури застосунку.	20.03.2022	
4.	Розробка сервісів.	23.03.2022	
5.	Тестування сервісів.	10.04.2022	
6.	Виправлення помилок і доопрацювання недоліків.	15.04.2022	
7.	Написання документації застосунку.	25.04.2022	
8.	Написання пояснювальної роботи.	10.05.2022	
9.	Здача курсової роботи	10.06.2022	

Студент _____
Керівник _____
“_____” _____

ЗМІСТ

Анотація

Чат-боти стають все більш популярними в месенджер платформах, проте для їх створення потрібні навички розробки та розгортання застосунків. В цій роботі описано застосунок-конструктор чат-ботів у відомому месенджері Телеграм, за допомогою якого можна створювати чат-ботів у веб-інтерфейсі. Поведінка користувача чат-бота описується за допомогою моделі детермінованого скінченого автомата (ДСА). Розроблено та розгорнуто застосунок-конструктор ботів, і додано інструкцію користувача до нього.

Ключові слова:

Чат-бот, месенджер, Телеграм, Телеграм Bot API, скінченний автомат, мікросервісна архітектура, документні БД, in-memory БД, веб-фреймворк nextjs.

Вступ

Чат-боти широко використовуються як помічники в рекламі бізнесу, розповсюдженні інформації або просто як окремі сервіси. Останнім часом почали набувати популярності зокрема чат-боти у месенджер платформах, таких як Телеграм, Slack, Viber. Відмінність таких чат-ботів полягає в тому, що месенджер-платформа повністю відповідає за зовнішній вигляд чату, а боти виглядають як звичайні користувачі платформи. Розробникам в таких випадках надається доступ до API, за допомогою якого можна управляти ботом та отримувати повідомлення від користувачів.

Управління ботом в таких випадках відбувається програмно, а отже для створення свого бота треба мати навички програмування та знання, як розгорнути застосунок. Щоб уникнути цих труднощів, в цій роботі розроблено застосунок-конструктор чат-ботів на прикладі одного з найвідоміших месенджерів – Телеграму. В роботі детально описано проектування та створення конструктора, а також представлено модель скінченого автомата, як спосіб керування чат-ботом.

Метою роботи є розробка ПЗ - конструктора ботів, що дозволяє з використанням графічного інтерфейсу легко створювати чат-ботів з детермінованою поведінкою.

Для досягнення мети були виділені задачі:

- Оглянути різні способи класифікації та побудови чат ботів.
- Розробити спосіб представлення поведінки чат-бота як моделі ДСА.
- Спроектувати, реалізувати та впровадити застосунок-конструктор.

Об’єкт дослідження:

Управління чат-ботом.

Предмет дослідження:

Методи управління чат-ботом.

Методи дослідження:

Використання формальної теорії автоматів для опису керування чат-ботом.

Використання шаблонів проектування для створення застосунку.

Структура роботи:

Робота має три розділи.

Розділ 1 – коротко описані основні способи класифікації чат-ботів та методи їх розробки. Описано математичну модель скінченого автомата для керування чат-ботом.

Розділ 2 – спроектовано застосунок-конструктор ботів, запропоновано графічну інтерпретацію моделі скінченого автомата та детально описано кожний сервіс застосунку. Описано процес розробки та впровадження застосунку.

Розділ 3 – надано інструкцію користувача та розглянуто практичний приклад використання.

Розділ 1. Огляд методів класифікації та побудови чат-ботів

1.1 Поняття та способи класифікації чат-ботів

Чат-бот – це комп'ютерна програма, з якою користувачі можуть обмінюватись повідомленнями. Програма може бути розроблена на основі нейронних мереж та алгоритмів штучного інтелекту, проте в цій роботі розглядаються саме чат-боти, поведінка яких чітко визначається розробником.

Такі чат-боти широко використовуються для ряду задач, наприклад для поширення довідкової інформації, як персональні утиліти для покращення продуктивності, або як «проксі»-зв'язок з командою підтримки продукту.

Першим чат-ботом в історії вважається ELIZA, розроблена 1966 Йозефом Вайценбаумом [1]. Програма імітувала розмову з терапевтом та задавала відкриті питання. Незвичним було те, що бот навіть продовжував розмову більш глибокими питаннями. Це досягалось розпізнаванням ключових слів у відповідях користувачів.

Існує багато способів класифікації чат-ботів. Shafquat Hussain et al, підсумовує різні способи класифікації чат-ботів [2]. Серед основних можна виділити:

- За способом комунікації:
    - Текстові – боти, що спілкуються із співрозмовником через текстові повідомлення.
    - Голосові – боти, що можуть розпізнавати голос та генерувати голосові відповіді.
- За використаними технологіями:
    - З використанням штучного інтелекту – на приклад, з використанням нейронних мереж або навчання без вчителя (reinforcement learning).

    - Побудовані на правилах – правила, що керують ботом задаються заздалегідь під час розробки бота.
- За цілями:
    - Направлені на вирішення конкретної задачі – на приклад, довідник або бот-асистент при подачі заявки.
    - Направлені на симуляцію вільної розмови – на приклад, бот терапевт або бот-асистент (Siri, Google Assistant).
- За платформою:
    - Веб-сторінка – боти розміщаються на веб-сторінці та пересилають питання, що виникають у відвідувачів до менеджерів сайту.
    - Месенджер – боти є користувачами месенджерів з обмеженими можливостями.
    - Окремі прилади – наприклад, Amazon Alexa.

Надалі ми будемо розглядати тільки текстові чат-боти побудовані на правилах та направлені на вирішення конкретної задачі довідкового характеру. Розглядаємо побудову чат-ботів тільки в месенджері Телеграм, так як він надає найбільш зручний та зрозумілий API для цього.

Shafquat Hussain також згадує про три основні методи побудови чат-ботів [2]:

- Rule-based – боти будуються на заданих правилах.
- Retriebal-based – за допомогою виявлення ключових слів або машинного навчання обирається найкраща відповідь на повідомлення з множини можливих.
- Generative-based – за допомогою методів машинного навчання без вчителя розробляється алгоритм, який може самостійно генерувати відповідь на повідомлення.

1.2 Переваги та недоліки самостійного створення чат-ботів

Багато платформ, такі як Telegram, Viber, Slack, Facebook Messenger, пропонують API, що можна використовувати програмно, при цьому розробляючи та розгортаючи бота окремо. Для цього треба мати знання та навички програмування, а також, в залежності від масштабу бота, знаходити хмарні ресурси для його розгортання.

Деякі платформи, наприклад WhatsApp, роблять наголос на «розумних» ботах, що використовують машинне навчання. WhatsApp пропонує WhatsApp Business API [3], проте ця опція доступна лише для бізнес-клієнтів.

Зазвичай, месенджери для управлінням ботом надають URL–адресу, на яку треба робити HTTP-запит. На приклад, така адреса для Телеграм є:

`https://api.telegram.org/bot<token>/<method>`

<token> - треба замінити на виданий платформою токен доступу.

<method> - треба замінити на необхідний метод. Список можливих методів та параметрів до них можна знайти за посиланням https://core.telegram.org/bots/api .

Виділимо переваги створення ботів самостійно за допомогою API, що надають месенджер-платформи:

- Гнучкість – не має обмежень щодо технологій розробки, типу бота чи його масштабу.
- Up-to-date – використовуючи API напряму, можливо використовувати найновіші можливості, додані до останніх версій.
- Безпека – немає потреби передавити токени доступу до API третім сторонам.

Розглянемо тепер недоліки такого підходу:

- Необхідність розробки програмного коду.

- Власне розгортання боту та налаштування інфраструктури.

Незважаючи на те, що недоліків мало, вони є досить суттєвими і стають перешкодами перед створенням своїх ботів для людей, що не є програмістами. Тому і виникла ідея розробити графічний застосунок-конструктор ботів, щоб усунути вищезгадані недоліки.

1.3 Детермінований скінченний автомат (ДСА) як метод моделювання поведінки користувача

Скінченний автомат – абстракція, що використовується для опису змін скінченої кількості станів об'єкта в залежності від інформації отриманої ззовні [4].

Будь-кого task-oriented чат-бота зручно представити у вигляді скінченого автомата:

- Вхідна абетка Σ - набір символів, що позначають повідомлення, які може розпізнавати бот. Складається з команд c та текстових повідомлень w. Наприклад, $\Sigma = \{c_{start}, w_{world\$}, w_{any}\}$, де c_{start} – команда «`/start`», $w_{world\$}$ – всі текстові повідомлення, що закінчуються на «`world`», w_{any} – будь-які текстові повідомлення.
- Маємо набір станів S – визначені розробником стани, в яких перебуває екземпляр бота у чаті із співрозмовником.
- Стан s_{start} – єдиний початковий стан, в якому знаходяться всі боти (екземпляри) до контакту з користувачем. Анти-спам політика Телеграму забороняє ботам першими писати користувачам.

- Множина кінцевих станів F – порожня, оскільки ми не отримуємо повідомлення від Телеграму, коли користувач заблокував бота. Оскільки користувач перестає діяти на бота, то нічого не відбувається до того моменту, коли користувач самостійно не продовжить переписку.
- Функція переходу δ: $S \times \Sigma \to S$ визначає, в який стан перейде бот у відповідь на повідомлення або команду користувача.
- Вихідна абетка Γ – множина можливих дій бота, наприклад $\Gamma = \{a_{do\ nothing}, a_{send\ msg}, \dots\}$.
- Функція виходів (дій бота) ω: $S \times \Sigma \to \Gamma^n$.

Розглянемо простий приклад Exho-боту. Суть чат-боту полягає в тому, що коли користувач надсилає повідомлення, то бот його відсилає назад. У такому разі

$$\Sigma = \{c_{start}, w_{any}\}$$

$$S = \{s_{start}, s_{echo}\}$$

Символ c_{start} – позначає команду користувача «`/start`», а w_{any} – повідомлення, що складається з будь-яких літер та не є командою.

Представимо функцію переходів (чорним) та функцію виходів (зеленим) графічно:

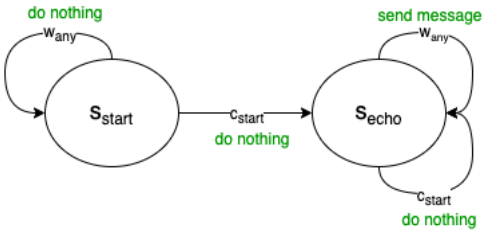

Рисунок 1.1. Схема exho-бота

Таким чином, бот буде неактивний до команди користувача /start, а потім буде пересилати назад будь-які повідомлення користувача, що не є командами.

Тепер ми хочемо ускладнити чат-бота і додати можливість відключати ехо-режим. Для цього введемо стан $s_{no\ echo}$ та додаємо ще дві команди c_{turn_on}- активувати ехо-режим, c_{turn_off} – відімкнути ехо-режим.

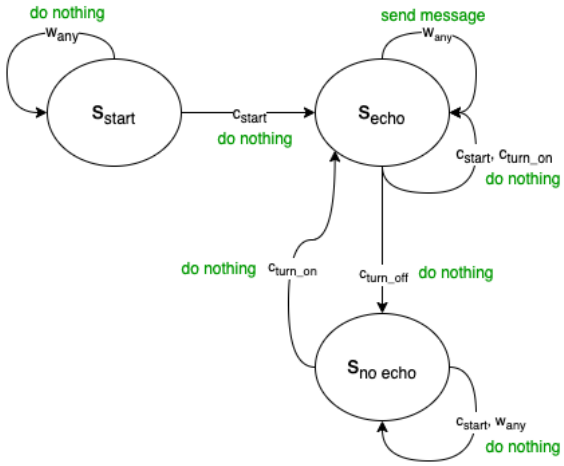

Рисунок 1.2. Схема ускладненого ехо-бота

Таким чином можна проектувати досить складні взаємодії між чат-ботом та користувачем.

Розділ 2. Проектування та розробка застосунку-конструктора ботів

2.1 Вимоги до застосунку

Розглянемо функціональні вимоги висунуті до застосунку. Для цього виділимо ролі:

- Власник бота (адміністратор) – створює ботів у веб-інтерфейсі. Активна роль, що взаємодіє із сервісом.
- Користувач бота – користувач Телеграму, спілкується з чат-ботом через месенджер.

Вимоги можна згрупувати за наступними категоріями:

- *Авторизація*. Адміністратор може:
  - Створити кабінет за допомогою пошти, вказавши пароль.
  - Створити кабінет за допомогою google.
  - Зайти в кабінет за допомогою пошти та пароля.
  - Зайти в кабінет за допомогою google.
- *Управління ботами*. Адміністратор може
  - Створити бота.
  - Видалити бота, при цьому автоматично видаляються всі дані про бота та його користувачі.
  - Призупинити / відновити роботу бота, не видаляючи його.
- *Конфігурація бота*. Адміністратор може
  - Редагувати ім’я та опис бота (тільки для відображення у застосунку).
  - Редагувати Telegram bot token.
  - Налаштовувати поведінку бота за допомогою діаграми.

    - Переглядати користувачів, що контактували бота, переглядати їхню історію переписки.
    - Відправляти повідомлення користувачам від імені бота.
- *Діаграма бота*. Адміністратор може
    - Додавати / видаляти стани.
    - До кожного стану додавати тригери (детальніше в п. 2.2.1):
        - Тригер на команду.
        - Тригер на текстове повідомлення.
    - До кожного тригеру можна додавати будь-яку кількість дій:
        - Відправити повідомлення.
        - Змінити стан.
        - Зберегти дані про користувача.
        - Надіслати API запит на сторонній сервіс.

Тепер розглянемо нефункціональні вимоги.

Оскільки за попереднім оцінками обсяг користувачів системи не має бути великий, то основні вимоги до системи:

- Мінімальні початкові інфраструктурні витрати.
- Легка масштабованість.
- Час відповіді бота на повідомлення має бути мінімальним.

2.2 Графічне представлення бота як ДСА

2.2.1 Тригери

Модель ДСА для моделювання поведінки користувача, що описана в п. 1.3, можна досить просто зобразити графічно. Для цього використаємо поняття *тригеру* [5] – це подія, повідомлення або команда від користувача,

для якої можна задати множину дій, що будуть виконані на цю подію. Тобто таким чином адміністратор графічно визначає вхідний алфавіт та задає функцію переходів і виходів.

Адміністратор також самостійно визначає стани. Для кожного стану можна додавати два типи тригерів: команди та повідомлення. Для тригеру-команди треба точно вказати команду, для тригеру-повідомлення – вказати регулярний вираз. Для кожного тригеру можна обрати 0 або більше дій, що виконає бот, коли відповідна подія надійде від користувача. Це є визначенням функцій переходів та виходів. Причому коли маємо 0 дій – стан не змінюється.

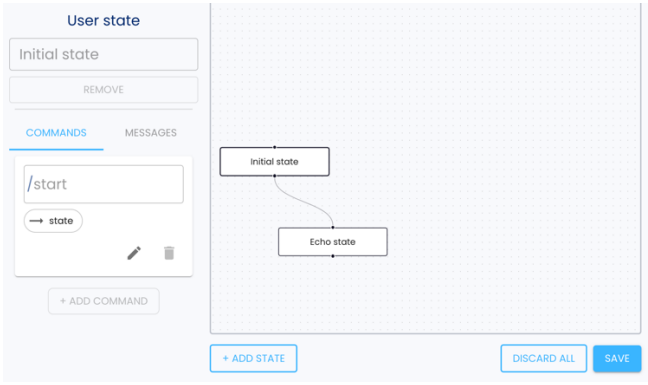

Рисунок 2.1. Приклад тригеру-команди

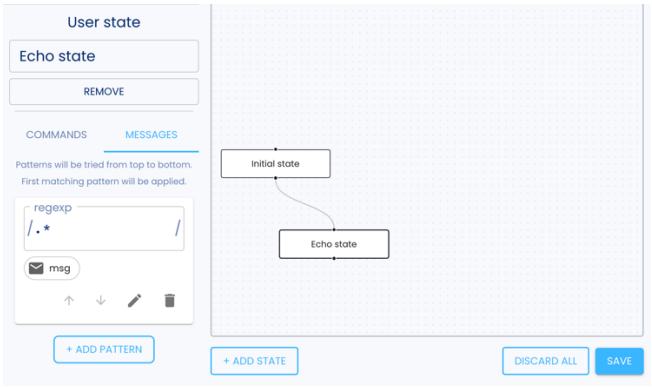

Рисунок 2.2. Приклад тригеру-повідомлення

2.2.2 Дії та шаблонні змінні

Для кожного тригера користувач може задавати наступні дії (вихідний алфавіт):

- Send message – надіслати користувачу повідомлення і залишитись в поточному стані.
- Change state – перейти в новий стан, що заданий адміністратором.
- Make API request – зробити запит на сторонній сервіс і залишитись в поточному стані.
- Save user data – зберегти якусь інформацію про користувача в БД і залишитись в поточному стані. Цю інформацію зможе бачити адміністратор в панелі користувачі, а також її можна буде використовувати в шаблонних змінних (див. нижче).

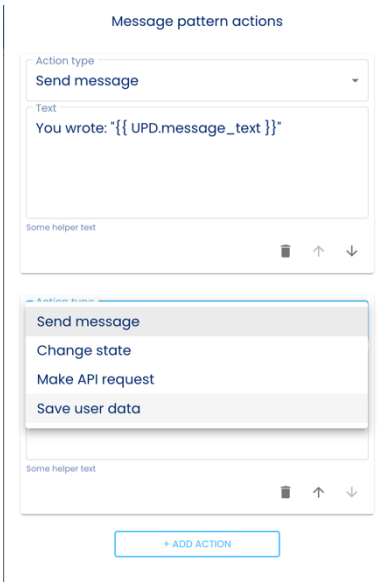

Рисунок 2.3. Приклад дій бота на тригер

Шаблонні змінні – це спеціальний набір символів у текстовому полі, що буде замінений на інше значення при обробці цього поля. Через них надається доступ до інформації про користувача бота. В нашому застосунку змінні завжди мають наступну форму:

$$\{\{\ DOMAIN.variable\ \}\}$$

Domain – це місце, звідки беремо змінну. На разі це лише `UPD` – з update, що надійшов від Телеграму, та `DB` – БД користувача, в яку була збережена змінна дією «Save user data».

Variable – назва змінної.

В прикладі на рис 2.3 `{{ UPD.message_text }}` буде замінено на текст повідомлення, що користувач надіслав боту. Більш детально про можливі змінні йдеться в додатку А.

2.3 Загальна архітектура системи

Враховуючи нефункціональні вимоги системи було вирішено розробити мікросервісну архітектуру застосунку. При незначному навантаженні всі сервіси можна запустити на одному хості, а при збільшенні навантаження використати оркестратор для автоматичного масштабування. При цьому з'являється можливість окремо масштабувати компоненти системи в залежності від їх навантаження.

Отже, на наступній діаграмі зображено основні компоненти системи та їх взаємодія:

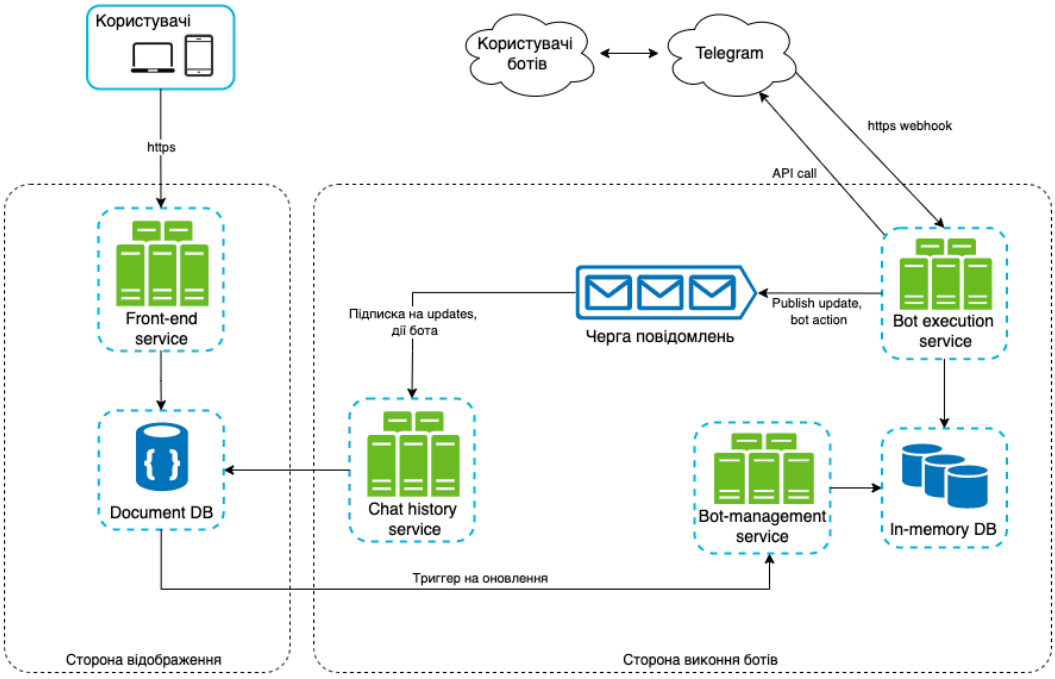

Рисунок 2.4. Схема архітектури застосунку

Інтерфейс користувача (UI) для адміністраторів ботів та логіка відображення, збереження ботів міститься у сервісі Front-end, хоча цей сервіс відповідає не тільки за сторону користувача, а також має простий CRUD API для ботів.

Оскільки наш застосунок не передбачає складних транзакцій, запитів зі сторони користувачів, а лише потребує простого збереження даних про ботів та користувачів, які будуть діставатись за ключем та деякими простими фільтрами (наприклад, вибрати всіх ботів, що належать конкретному користувачу), то доцільно використати документну базу даних.

Сервіс bot-execution відповідає за обробку updates, що приходять від Телеграму, і виконання дій, що мають відбутися на цей update.

Bot-execution має інші вимоги до сховища даних ніж front-end сервіс, а саме: підвищена швидкодія та запити до ботів завжди лише за ключем. Для таких потреб було використано in-memory key-value базу даних.

Окрім цього, нам також необхідно конвертувати документ-об’єкт бота з документної БД у зручний формат для сервісу виконання. Цим займається bot-management сервіс, який слухає на події оновлення інформації у документній базі даних (детальніше у п. 2.4.1 та п. 2.4.3).

Як ми бачимо, два вищенаведені середовища досить слабко зв’язані між собою. Але ми також хочемо, щоб адміністраторам ботів була доступна історія чатів з усіма користувачами ботів. Для цього на кожний update та кожну дію бота ми будемо публікувати повідомлення в чергу повідомлень. Chat-history сервіс буде підписаний на повідомлення з цієї черги та зберігати необхідну інформацію в документну БД для подальшого відображення для адміністраторів.

2.4 Опис основних компонент архітектури

Розглянемо кожен сервіс та компонент архітектури більш детально.

2.4.1 Документна база даних

Для документного сховища даних, в якому має зберігатись інформація про акаунти власників ботів та про самих ботів, обрано MongoDB через її популярність та широкий free tier для managed cloud cluster, можливостей якого цілком вистачає для цього застосунку.

Побудуємо схему даних. Маємо такі сутності:

- **User**

  Користувач сервісу.

  Приклад документу:

```
_id: ObjectId("602fc5f4841d7ec919af3682")
uid: "example_uid"
profile: Object
   email: "user@example.com"
   email_verified: false
   provider: "password"
__v: 0
```

  Поле profile – надається auth провайдером – firebase.

- **Bot**

  Бот, створений користувачем

  Приклад документу:

```
_id: ObjectId("624c138d74fe81a3c936e348")
owner: "example_uid"
name: "Echo bot"
token:
status: false
src: Object
created: 2022-04-05T10:01:49.059+00:00
__v: 0
description: "Test"
```

  Поле src – безпосередньо перелік станів, тригерів та службової інформації, необхідної для відображення діаграми бота. Для прикладу Exo-бота з п. 1.3 поле src виглядатиме так:

```
src: Object
  init: Object
  1650007766991: Object
    id: "1650007766991"
    type: "default"
    position: Object
    data: Object
      label: "Echo state (enabled)"
      commands: Array
      messages: Array
        0: Object
          regexp: ".*"
          actions: Array
            0: Object
              type: "send_message"
              options: Object
                text: "{{ UPD.message_text }}"
              id: "0"
            id: "0"
  1650007909595: Object
```


- **BotUser**

  Користувач Телеграм, який контактував з ботом

  Приклад документу:


```
_id: ObjectId("6231afaa3055e1c8b42a9952")
botId: ObjectId("624c138d74fe81a3c936e348")
botOwner: "example_uid"
id: 425956289
firstName: "Michael"
lastName: "Kobelev"
username: "mike_mars"
created: 2022-03-16T09:36:42.931+00:00
__v: 0
db: Object
state: "1649858917878"
```


- **Message**

  Подія-повідомлення в чаті. Це може бути повідомлення користувача, відповідь бота або інша дія бота.

  Приклад документу:


```
_id: ObjectId("6256e50b2c47a8aac8ecb1a0")
botId: ObjectId("624c138d74fe81a3c936e348")
botOwner: "example_uid"
chatId: 425956289
isBot: false
type: "message"
ts: 1649861462
msg: Object
created: 2022-04-13T14:58:19.751+00:00
__v: 0
```


2.4.2 Сервіс front-end

Front-end сервіс це nextjs сервер, який містить як клієнтьский React код так і API для роботи з MongoDB.

Для полегшення роботи з front-end було використано open-source бібліотеку з готовими графічними елементами Material UI.

Окремою проблемою була візуалізація бота, оскільки існує не так багато сумісних з React бібліотек для відображення діаграм. Було обрано бібліотеку react-flow-renderer через простоту її використання та налаштовування.

2.4.3 In-memory сховище даних

Як вже було наведено, ми хочемо зменшити час відповіді бота на повідомлення користувача. Для цього доречно використати in-memory базу даних, де дані про бота дублюються з MongoDB, але у наступному, більш зручному, форматі:

```
{
“token”: bot access token
“states”: {
	“state_1”: {
		“cmd_triggers”: {
			“/start”: [ actions ]
			“/other_cmd”: [ actions ]
		},
		“msg_triggers”: {
			“world$”: [ actions ]
			“.*”: [ actions ]
		},
	“state_2”: …
}
```

Для цього було використано Redis. Ключ – ID бота, значення – серіалізований JSON зі структурою, наведеною вище.

У окремій структурі Hash [6] ми зберігаємо стани всіх користувачів за допомогою команд:

```
hset <bot id>_states <user id> <state>
hget <bot_id>_states <user id>
```

Ми також не хочемо, щоб при перебоях в роботі Redis сервісу ми втратили інформацію про ботів, тому увімкнули налаштування persistence для Redis [7].

2.4.4 Сервіс bot-management

Це порівняно невеликий сервіс. Його основна задача – слухати на Change Stream [8] від MongoDB, та коли оновлюється будь який документ бота в базі даних, ми конвертуємо його у зручний формат для збереження в Redis, наведений у п. 2.4.3.

Окрім цього, цей сервіс робить запит на API Телеграму і назначає webhook адресу, куди месенджер буде надсилати update. Webhook адреса формується у вигляді:

`https://<machine-ip-address>/update/<bot-id>`

Таким чином, коли нам надходить update, ми знаємо id бота, якого він стосується.

2.4.5 Сервіс bot-execution.

Сервіс bot-execution – єдиний мікросервіс системи написаний не на JS. Через вимоги до швидкодії було обрано компільовану мову програмування – Go.

Сервіс по суті виступає сервером, що має один ендпоінт - `/update/<bot-id>`. Коли на цей ендпоінт приходить update від Телеграму, ми його парсимо, дістаємо із Redis конфігурацію необхідного бота та стан поточного користувача. В залежності від типу повідомлення ми робимо пошук необхідного тригера (див. п. 2.2.1) та виконуємо дії відповідного тригера.

2.4.6 Черга повідомлень.

Чергу повідомлень, як засіб обміну даними між стороною виконання та стороною відображення (рис 2.4), було обрано, щоб залишити зв'язаність між цими середовищами слабкою. Для цього було обрано RabbitMQ через його популярність, open-source походження та зручний веб-інтерфейс для спрощеного відлагодження.

Розглянемо схему публікації повідомлень.

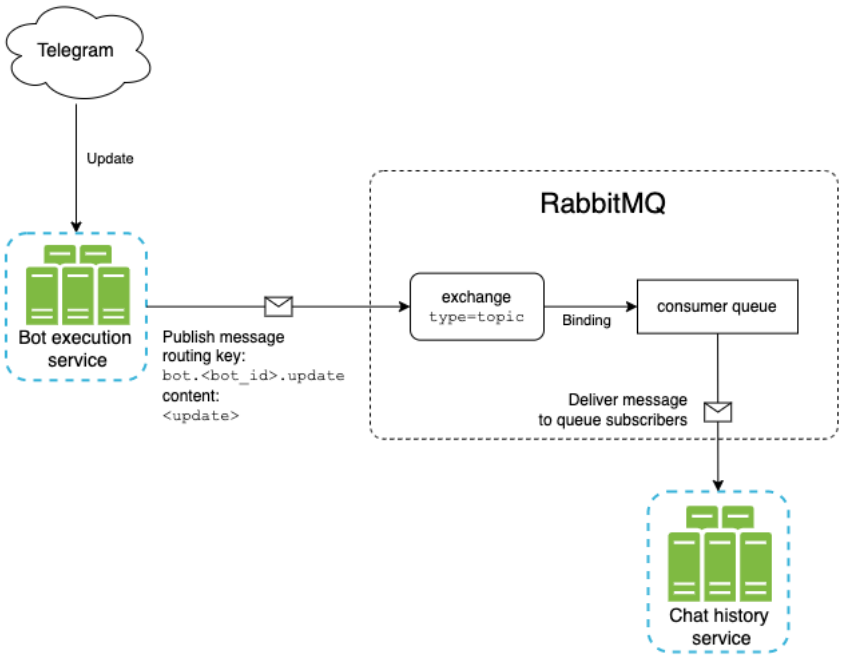

Рисунок 2.5. Схема публікації повідомлень

При надходженні update від Телеграму, ми публікуємо повідомлення з ключем `bot.<bot_id>.update` на topic exchange. Exchanges в RabbitMQ потрібні для того, щоб направляти повідомлення в потрібні черги. Ми прив'язуємо Exchange до черги за допомогою binding. У черзі повідомлення можуть зберігатись довгий час, поки вони не будуть оброблені сервісом, що підписаний на цю чергу. У нашому випадку це сервіс історії чатів, який збереже нове повідомлення від користувача в MongoDB для подальшого перегляду користувачем.

2.4.7 Сервіс історії чатів.

Тепер, коли ми маємо чергу з повідомлень про дії користувачів та ботів, нам потрібен окремий сервіс, що буде зберігати ці дані в документну базу даних. Для цього розроблено невеликий nodejs застосунок, що підписується на чергу з повідомленнями та зберігає їх як Message документи в MongoDB (приклад такого Message в п 2.4.1).

2.5 Розробка мікросервісів

Код для всіх мікросервісів зберігається в одному git-репозиторії. На початку кожний мікросервіс розроблявся та тестувався окремо, оскільки сервіси є незалежними. Запити зі сторонніх служб та сервісів можна було легко зімітувати (mock).

Окремою складністю було інтеграційне тестування та відлагодження декількох сервісів разом.

Для сторони відображення було відносно легко зробити середовище розробки та інтеграційного тестування, оскільки більшу частину роботи бере на себе фреймворк nextjs. Він надає локальний hot-reload static сервер для фронтенду і для функцій API. Необхідно було лише налаштувати окремий інстанс MongoDB для локальної роботи.

Сторону виконання ботів було дещо складніше відлагоджувати, оскільки локально було потрібно запустити сервіси bot-management, bot-execution та сервіс історії чатів. Проте залишалась проблема отримання updates від Телеграму. В месенджері доступні два способи отримання updates:

- Polling – періодичні запити на Телеграм. Не рекомендовано для використання.

- Webhooks – Телеграм сам робить запит на вказану нами адресу з update.

Оскільки для production середовища має використовуватись webhook, то і локально мало сенс тестувати з цим методом. Для налаштування адреси для webhook використовувалась утиліти ngrok, яка перенаправляє запити з тимчасової адреси на вказаний порт локальної машини:

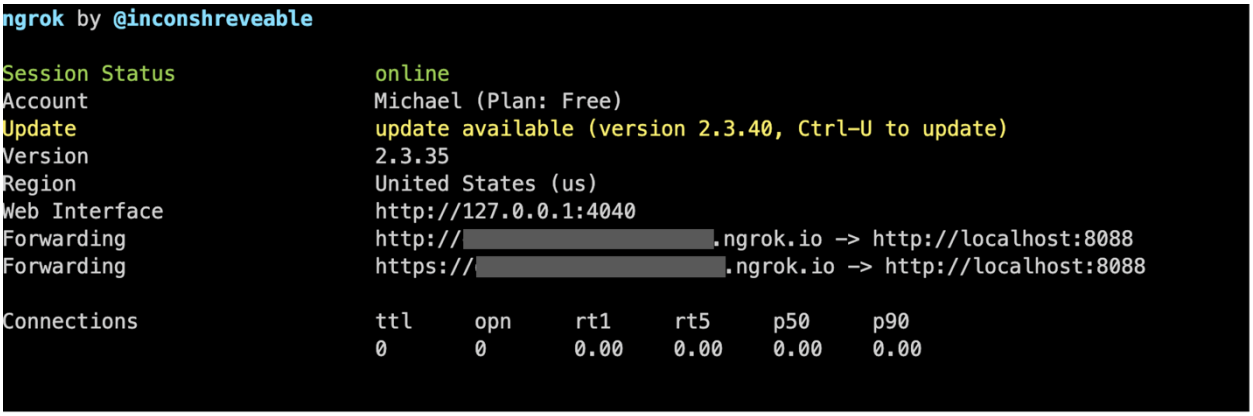

Рисунок 2.6. Використання ngrok

2.6 Deployment

Deployment також можна умовно поділити на дві частини:

Сторона відображення користувача – платформа vercel пропонує безкоштовний хостинг nextjs проектів.

Сторона виконання ботів – це окрема віртуальна машина (використовувся Digital Ocean cloud-provider), на якій запущено всі сервіси окрім front-end. Вони спілкуються через приватну мережу машини, лише до серверу ngnix є відкритий доступ для отримання запитів від Телеграм. Конфігурацію docker-compose можна переглянути в додатку Б.

Розділ 3. Демонстрація можливостей застосунку

3.1 Users guide

Описаний сервіс було розроблено та розгорнуто. Він доступний за наступною адресою: https://tgbot-builder.vercel.app/ . Для роботи з сервісом необхідно створити акаунт.

Для полегшення використання сервісу було розроблено документацію для користувачів та опубліковано на сайті сервісу англійською мовою. Перейти на неї можна за посиланням: tgbot-builder.vercel.app/docs.

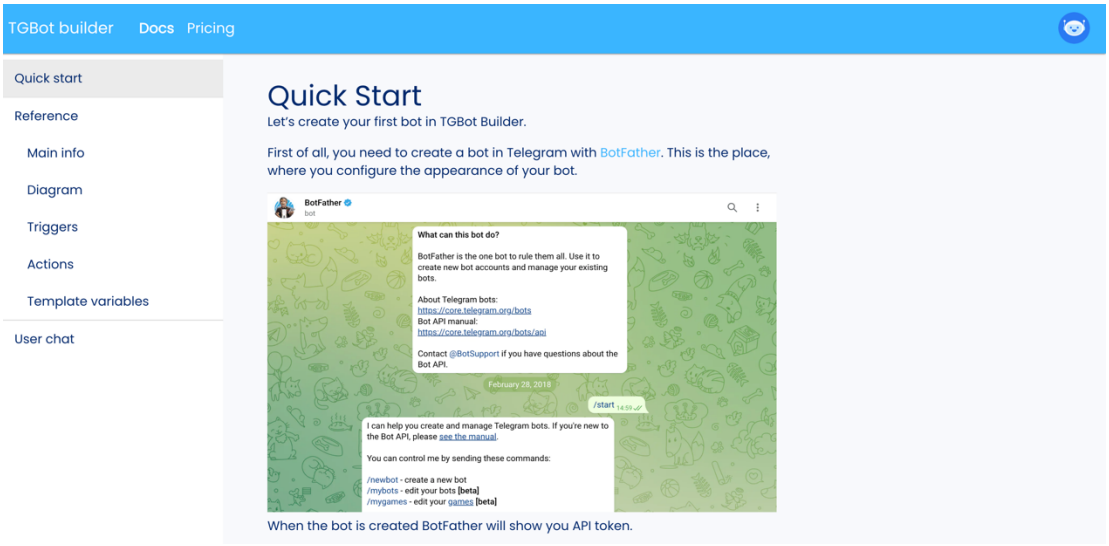

Рисунок 3.1. Скріншот сторінки документації

Документацію можна поділити на такі частини:

- Quick start – приклад створення простого чат-бота від початку і до кінця.
- Reference – детальний опис основних компонент проектування та розробки чат-боту. Серед них:
    - Main info - головна інформація про бота: токен доступу, ім'я та короткий опис.
    - Diagram - додавання нових станів.

- Triggers – визначення вхідного алфавіту, функції переходів.
- Actions – визначення функції виходів для конкретного тригера.
- Template variable - робота із шаблонними змінними при визначенні повідомлень-відповідей бота.

- User chat – перегляд історії листування користувачів з ботом.

3.2 Створення боту на прикладі боту-довідника у надзвичайних ситуаціях

Розробимо за допомогою сервісу прототип боту, що буде надавати інформацію про те, як треба діяти в надзвичайних ситуаціях.

Для цього, як описано у Quick Start секції документації, треба спочатку отримати токен доступу від служби Bot Father Телеграму.

Коли токен отримано, відкриваємо сторінку My Bots та клікаємо кнопку Create bot та заповнюємо поля.

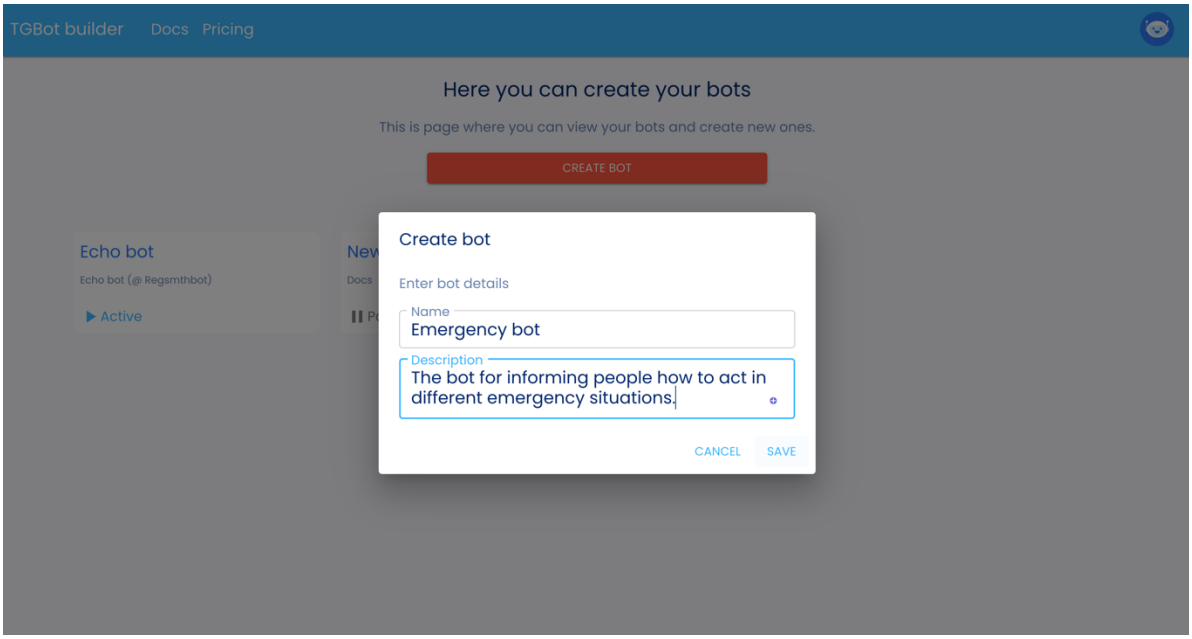

Рисунок 3.2. Скріншот створення бота

Варто пам’ятати, що ім’я та опис бота стосуються лише цього застосунку. Щоб налаштувати їх для Телеграму треба користуватись службою Bot Father, оскільки Телеграм не дає програмного доступу до конфігурації ботів.

Оскільки ми робимо прототип бота, то текстові повідомлення будуть прикладами, не оригінальним текстом. Ми не співпрацювали з ДСНС, тому інформація може бути не повною та не є достовірною, і ми не маємо права публікувати цей чат-бот.

Отже, поділимо наш довідник наприклад на такі розділи:

- Війна
    - Сирена
    - Чути постріли
    - Чути свист вгорі
- Здоров’я
    - Виклик швидкої
    - Запаморочення
    - Крововилив
- Небезпечні ситуації вдома
    - Пожежа
    - Витік газу
    - Розбився термометр

Також є сенс зробити один основний розділ, який буде перелічувати всі доступні розділи та коротко їх описувати.

Тоді переходимо до розділу Diagram для конфігурації бота та додаємо наступні стани:

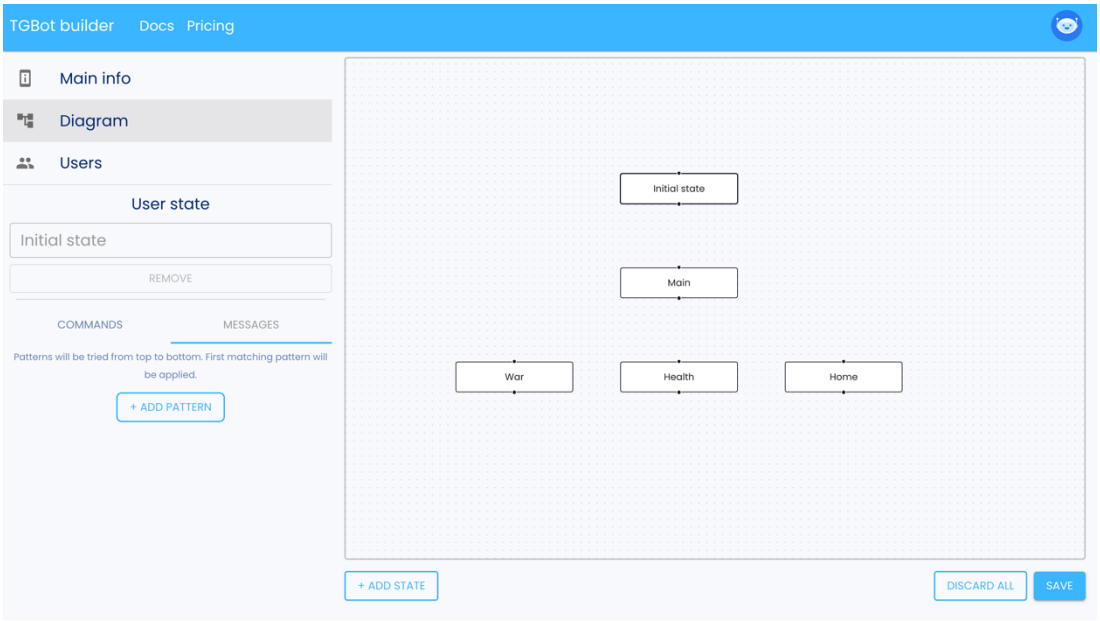

Рисунок 3.3. Скріншот діаграми бота зі станами

Стан initial – існує за замовчуванням. В цей стан на команду «/start» робимо відправку привітального повідомлення і одразу перехід на стан Main.

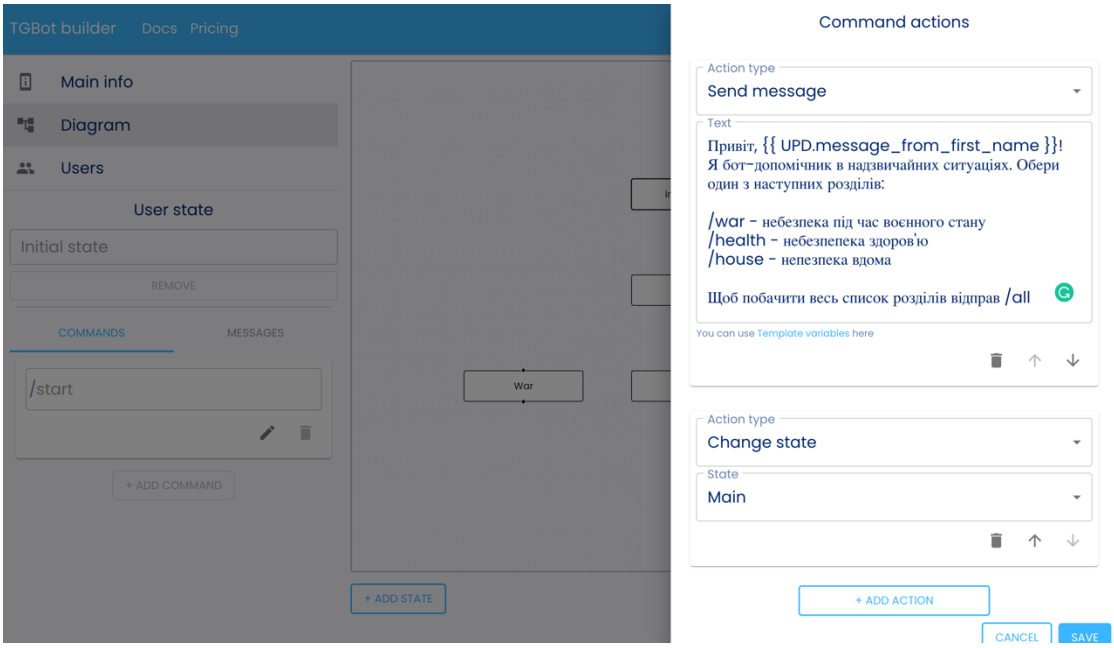

Рисунок 3.6. Скріншот повідомлення на команду /start початкового стану

Тепер налаштуємо команди для переходу на тематичні стани зі стану Main:

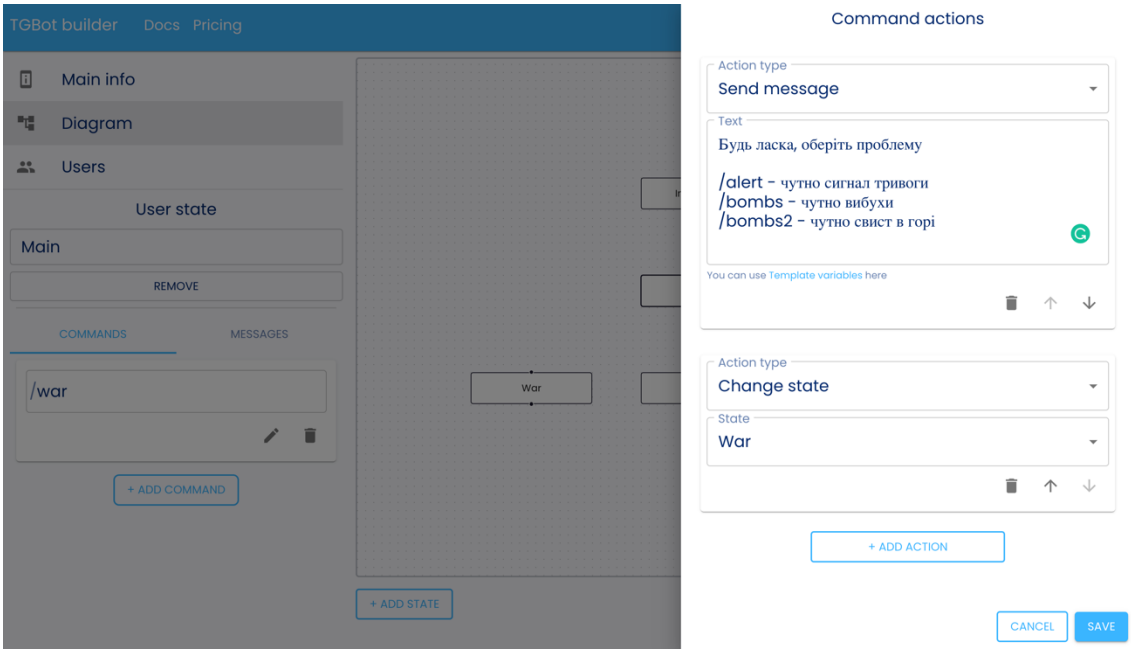

Рисунок 3.4.. Скріншот дії на в стані Main

Аналогічно для всіх 3-х команд

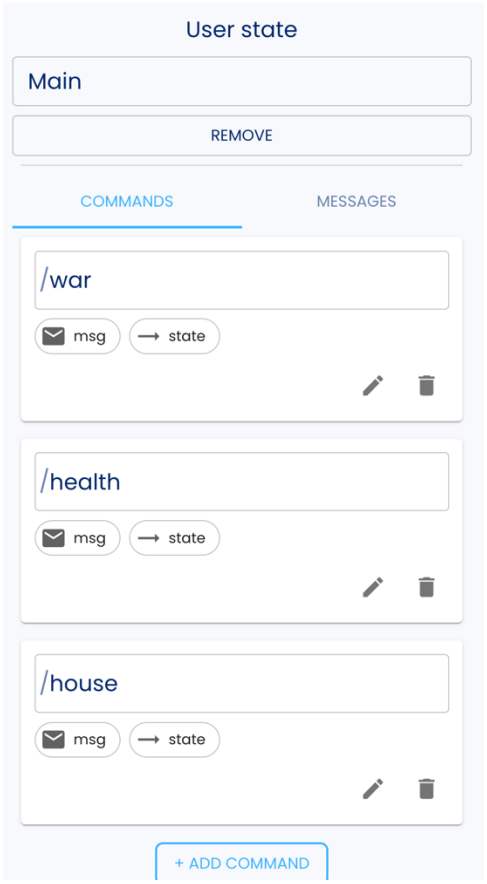

Рисунок 3.5. Скріншот команд бота в стані Main

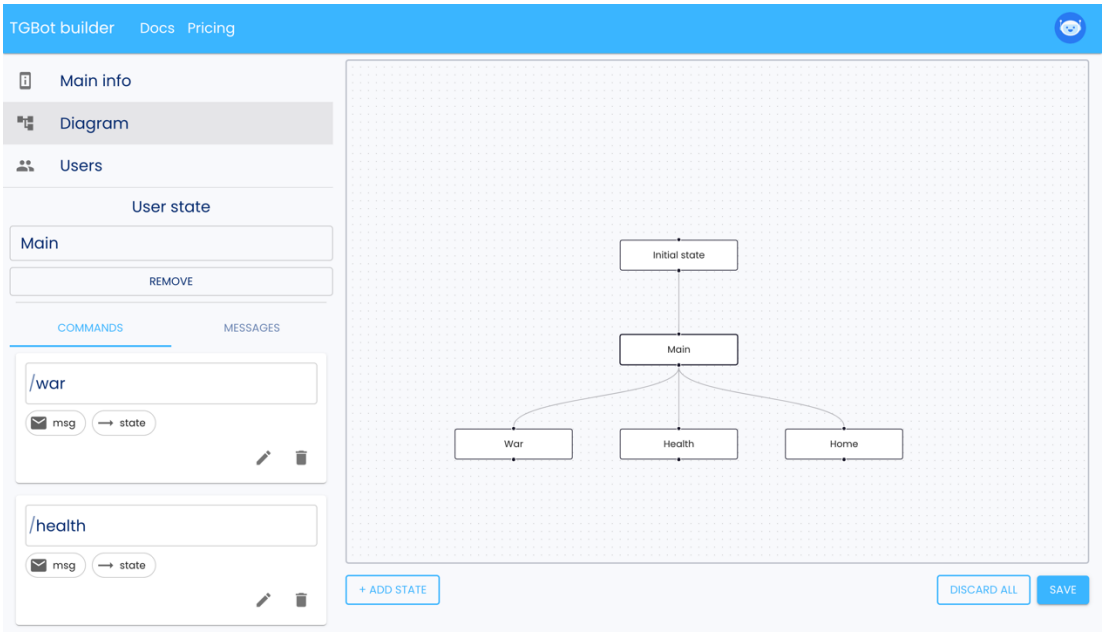

Рисунок 3.6. Скріншот діаграми бота

Тепер окремо для кожного стану пропишемо підкоманди та команду повернення до головного меню:

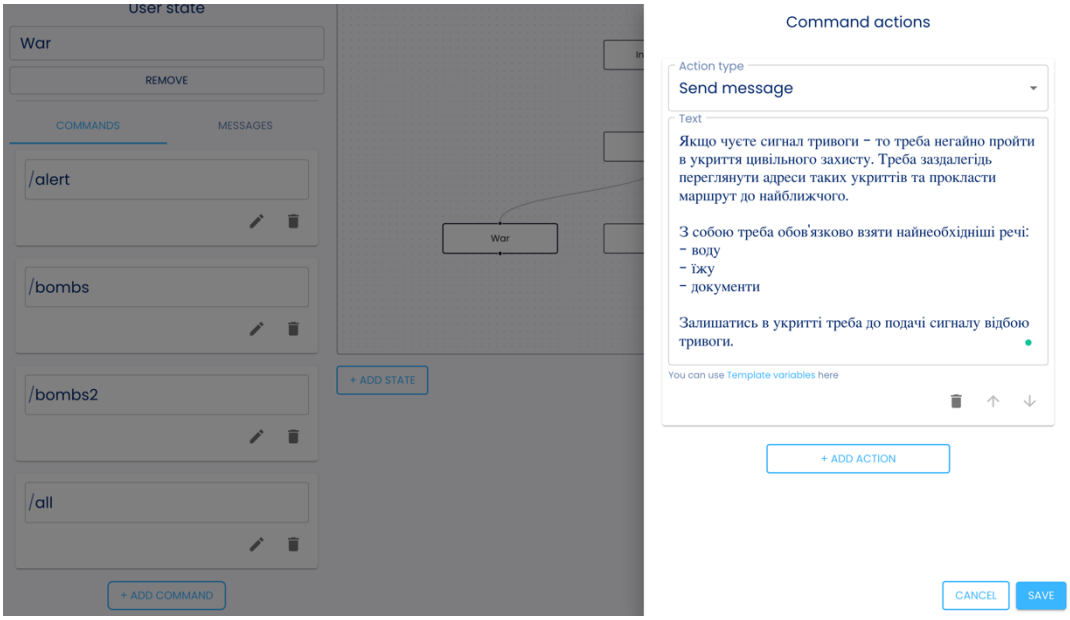

Рисунок 3.7. Скріншот повідомлення

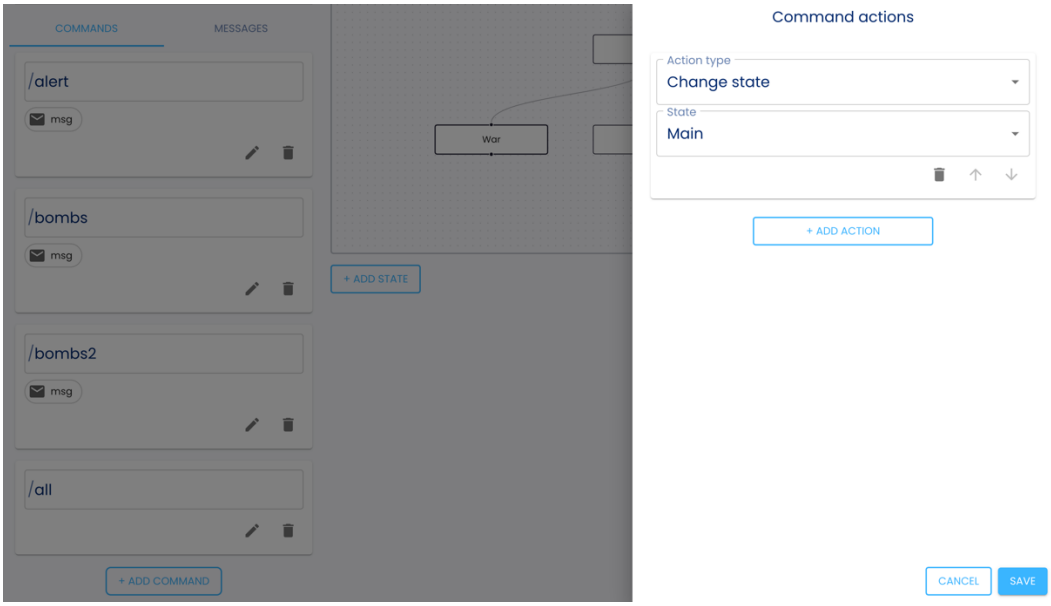

Рисунок 3.8. Скріншот дії «Зміна стану»

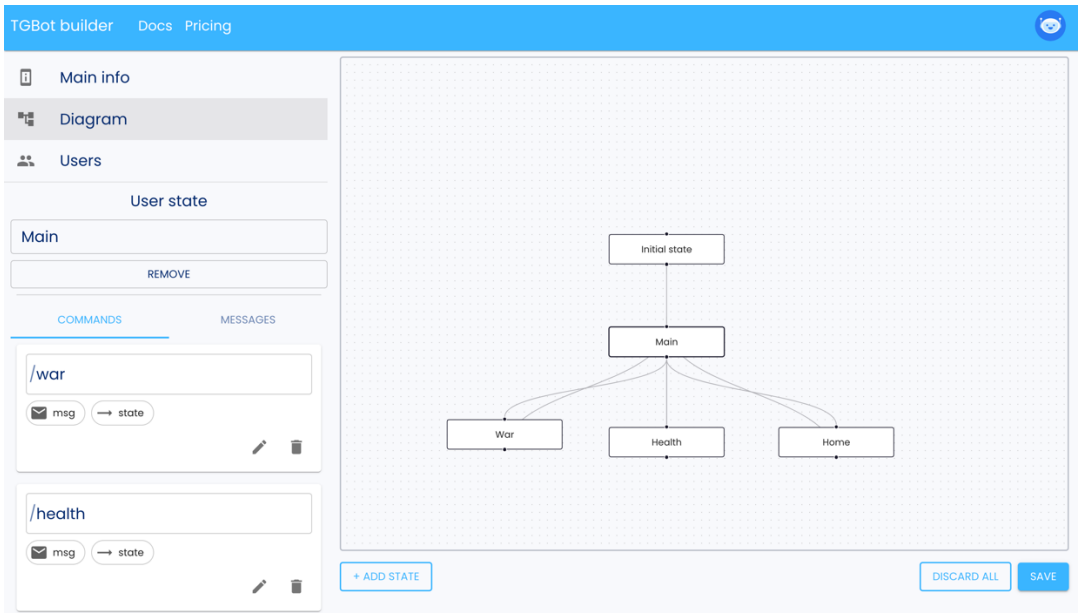

Рисунок 3.9. Скріншот фінальної діаграми бота

Тепер клікаємо на кнопку Save та переходимо в Main info. Додаємо токен доступу та активуємо бота.

Рисунок 3.10. Скріншот налаштувань бота

Ботом вже можна користуватись:

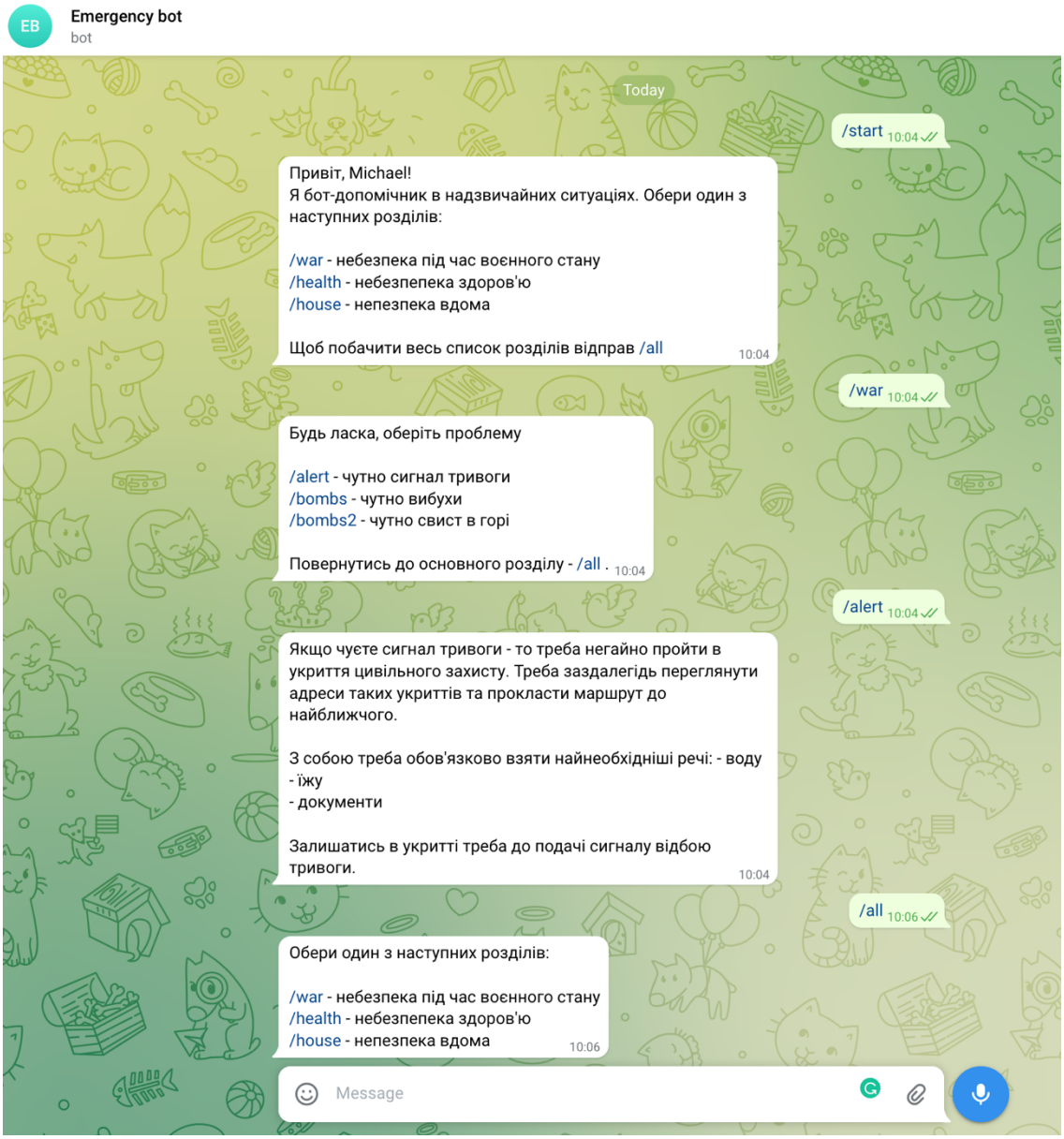

Рисунок 3.11. Скріншот переписки з ботом

В розділі користувачів тепер можна побачити, хто контактував бота та переглянути переписку:

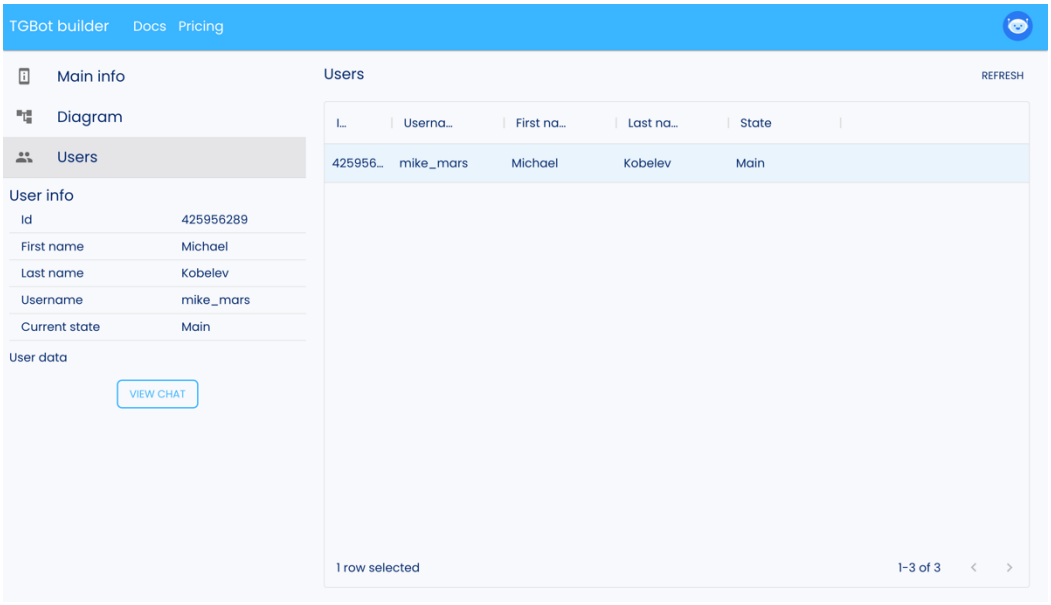

Рисунок 3.12. Скріншот спику користувачів

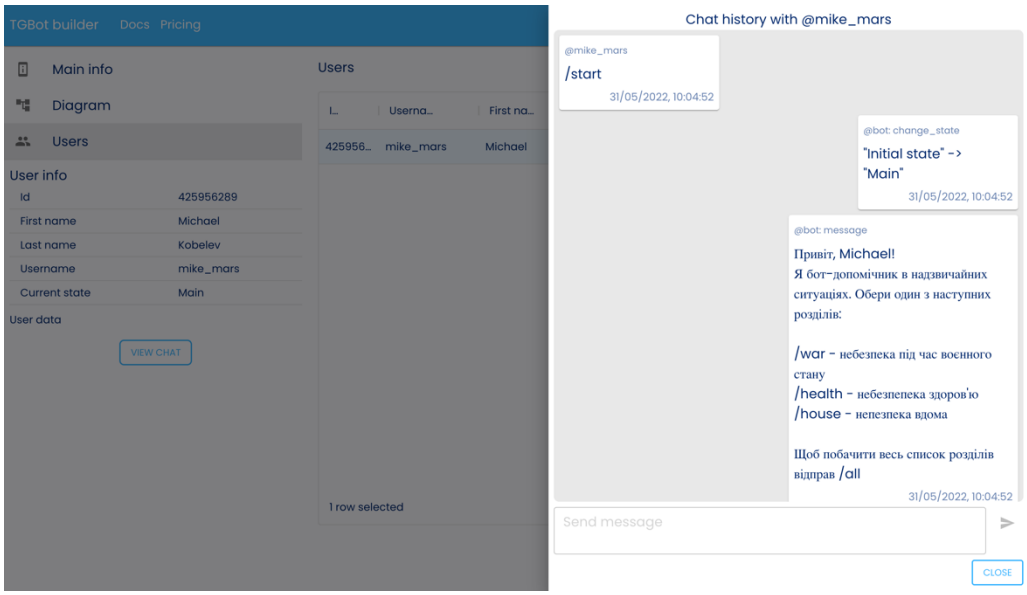

Рисунок 3.13.. Скріншот переписки з користувачем

Висновки

В ході роботи було проведено огляд основних підходів до класифікації чат-ботів та їх побудови. В практичній частини роботи розглядаються rule-based текстові чат-боти в месенджер-платформі Телеграм. Основною задачею роботи було створення UI конструктора для таких чат-ботів. З цією метою, було використано модель скінченного автомата, що стала доречною для графічного модулювання взаємодії користувача з чат-ботом. Цю модель було покладено в основу UI конструктора.

Під час виконання роботи було уточнено вимоги для застосунку та розроблено архітектуру, що складається з кількох невеликих та порівняно незалежних сервісів. Застосунок протестовано та розгорнуто в хмарному середовищі. Функціонал застосунку дозволяє моделювати різні за складністю варіанти взаємодії користувача з чат-ботом, проте його може бракувати для виконання більш складних дій, як наприклад обмін файлами або використання інтерактивних кнопок.

Отже, з цього можна зробити висновок, що розроблений застосунок є досить зручним для швидкої побудови чат-ботів і може суттєво спростити етапи їх розробки та розгортання. Наприклад, створення ботів-довідників, ботів-помічників із подання заявок або ботів-помічників для організації подій. Проте для більш кастомних взаємодій, таких як складна логіка обробки повідомлень, поширення файлів та використання платежів, він не підійде, і в таких випадках необхідно розробляти чат-бота власноруч.

Надалі планується розширити описану в роботі модель іншими концептами. На приклад, додати для зручності ще два тригери:

- Initial – дії виконуються одразу, як користувач переходить в цей стан.

- Schedule – користувач визначаю розклад за допомогою crontab, коли треба виконати ці дії.

З їх допомогою можна буде створювати, на приклад, ботів для нагадування та розсилок. Окрім тригерів, планується також додати підтримку Телеграм-клавіатури для більш зручного інтерфейсу бота на стороні месенджеру.

Список використаної літератури

1. Story of ELIZA, the first chatbot developed in 1966 [Електронний ресурс] – Режим доступу до ресурса: https://analyticsindiamag.com/story-eliza-first-chatbot-developed-1966/
2. Hussain, Shafquat & Sianaki, Omid & Ababneh, Nedal. (2019). A Survey on Conversational Agents/Chatbots Classification and Design Techniques. 10.1007/978-3-030-15035-8_93. (https://www.researchgate.net/publication/331746678_A_Survey_on_Conversational_AgentsChatbots_Classification_and_Design_Techniques)
3. WhatsApp Business | Transform Your Business [Електронний ресурс] – Режим доступу до ресурса: https://business.whatsapp.com/
4. Скінченний автомат — Вікіпедія [Електронний ресурс] – Режим доступу до ресурса: https://uk.wikipedia.org/wiki/%D0%A1%D0%BA%D1%96%D0%BD%D1%87%D0%B5%D0%BD%D0%BD%D0%B8%D0%B9_%D0%B0%D0%B2%D1%82%D0%BE%D0%BC%D0%B0%D1%82
5. Triggers in Bot Framework Composer | Microsoft Docs [Електронний ресурс] – Режим доступу до ресурса: https://docs.microsoft.com/en-us/composer/concept-events-and-triggers?tabs=v2x
6. HSET | Redis [Електронний ресурс] – Режим доступу до ресурса: https://redis.io/commands/hset/
7. Redis persistence | Redis [Електронний ресурс] – Режим доступу до ресурса: https://redis.io/docs/manual/persistence/

8. Change Streams — MongoDB Manual [Електронний ресурс] – Режим доступу до ресурса: https://www.mongodb.com/docs/manual/changeStreams/

ДОДАТКИ

Додаток А

(обов’язковий)

Перелік шаблонних змінних та їх опис

- message_message_id – ID повідомлення в чаті
- message_date – timestamp відправки повідомлення
- message_text – текст повідомлення
- message_from_id – ID користувача, що надіслав повідомлення
- message_from_first_name – ім’я користувача бота
- message_from_last_name – прізвище користувача бота
- message_from_username – Телеграм username користувача бота
- message_from_language_code – мова інтерфейсу користувача бота
- message_chat_type – типу чату(private / group / channel)
- message_chat_title – назва чату
- message_chat_username – Телеграм username чату
- message_chat_first_name – ім’я чату (лише для приватного)
- message_chat_last_name – прізвище чату (лише для приватного)
- message_chat_description – опис чату

Додаток Б
(обов'язковий)
Конфігурація Docker compose

```
version: '3.8'

services:
  redis:
    image: redis:6.2-alpine
    container_name: redis
    command: redis-server --appendonly yes --save 300 1 --dir /data
    restart: always
    volumes:
      - redis-data:/data
    ports:
      - "127.0.0.1:6379:6379"

  rabbitmq:
    hostname: 'rabbitmq'
    container_name: rabbitmq
    image: rabbitmq:3.9.13-management
    restart: always
    ports:
        - "127.0.0.1:5672:5672"
        - "127.0.0.1:15672:15672"
    volumes:
        - rabbitmq-data:/var/lib/rabbitmq/mnesia/

  bot-execution:
    build:
      context: ./bot-execution
    command: ['update_http']
    environment:
      - RABBITMQ_HOST=rabbitmq
      - REDIS_HOST=redis
      - SENTRY_DSN=${SENTRY_DSN_EXEC}

  bot-management:
    build:
      context: ./bot-management
```

```
    command: ["npm", "start"]
    restart: always
    environment:
      - REDIS_HOST=redis
      - NODE_ENV=production
      - WEBHOOK_HOST=https://${HOST_IP}
      - SENTRY_DSN=${SENTRY_DSN_MANAGE}
      - CONN_STR=${MONGO_CONN_STR}
    ports:
      - "127.0.0.1:4001:4001"

  history:
    build:
      context: ./history
    command: ["npm", "start"]
    environment:
      - NODE_ENV=production
      - RABBITMQ_HOST=rabbitmq
      - SENTRY_DSN=${SENTRY_DSN_HISTORY}
      - CONN_STR=${MONGO_CONN_STR}

volumes:
  redis-data:
  rabbitmq-data:
```