ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра інженерії програмного забезпечення

Пояснювальна записка
до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**Розробка Telegram чат-боту з динамічною конфігурацією мовами C# та JavaScript**»

Виконав: студент 4-го курсу, групи ПД-41
спеціальності

121 Інженерія програмного забезпечення

(шифр і назва спеціальності/спеціалізації)

Довгальов М.В.

(прізвище та ініціали)

Керівник Негоденко О.В.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Київ - 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

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

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення

Негоденко О.В.

“ ” 2023 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТА

ДОВГАЛЬОВА МИХАЙЛА ВАЛЕРІЙОВИЧА

(прізвище, ім’я, по батькові)

1. Тема роботи: “Розробка Telegram чат-боту з динамічною конфігурацією мовами C# та JavaScript”

Керівник роботи: Негоденко О.В. к.т.н., доц.

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «24» лютого 2023 року №.26

2. Строк подання студентом роботи “01” червня 2022 року
3. Вхідні дані до роботи:

Методи обробки та зберігання даних;

Інструменти з розробки програмного забезпечення: ASP.NET CORE 7.0, IDE JetBrains Rider 2023.1, JetBrains DataGrip 2023.1, Docker, Git, Linux Server

Науково-технічні документації та література по розробці мікро сервісної архітектури.

Технічна документація АПІ по Telegram ботів

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити).
   - 4.1. Аналіз предметної області
   - 4.2. Технічне завдання
   - 4.3. Функціональні вимоги
   - 4.4. Нефункціональні вимоги
   - 4.5. Асоціативна мапа
   - 4.6. Діаграма класі використання
   - 4.7. Інструменти та засоби розробки програмного забезпечення
   - 4.8. Архітектура системи
   - 4.9. Опис програмного забезпечення
5. Перелік демонстраційного матеріалу
   - 5.1. Титульний слайд.
   - 5.2. Мета, об’єкт, предмет, наукова новизна дослідження.
   - 5.3. Актуальність.
   - 5.4. Аналіз аналогів.
   - 5.5. Технічні завдання.
   - 5.6. Програмні засоби та інструменти реалізації.
   - 5.7. Розробка архітектури.
   - 5.8. Реалізація програми.
   - 5.9. Висновки
   - 5.10. Апробація результатів дослідження.
   - 5.11. Кінцевий слайд.
6. Дата видачі завдання «25» лютого 2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04-14.04	Виконано
2	Аналіз та дослідження існуючих аналогів	15.04-17.04	Виконано
3	Проектування системи	18.04-21.04	Виконано
4	Створення та тестування програмного рішення	21.04-05.05	Виконано
5	Підготовка розділу 1	05.05-07.05	Виконано
6	Підготовка розділу 2	07.05-09.05	Виконано
7	Підготовка розділу 3	08.05-11.05	Виконано
8	Вступ, висновки, реферат	11.05-12.05	Виконано
9	Підготовка розділу 1	12.05-15.05	Виконано
10	Розробка обов‘язкових демонстраційних матеріалів	19.05-25.05	Виконано
11	Попередній захист роботи та перевірка на плагіат	01.06	Виконано
12	Здача роботи	11.04-14.04	Виконано

Студент

_____ _____

(підпис) (прізвище та ініціали)

Керівник роботи

_____ _____

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи 55 с., 32 рис., 3 табл., 2 дод., 13 джерел. МЕСЕНДЖЕР, ЧАТ-БОТ, КОМУНІКАЦІЯ, ЧАТ, МІКРОСЕРВІСИ, C#, .NET CORE, API, REST, VUE JS

Об'єкт дослідження - процес комунікації малого бізнесу з клієнтами.

Предмет дослідження – чат-бот з динамічним конфігуруванням.

Мета роботи - підвищення ефективності комунікації клієнтів за рахунок Telegram чат-боту з динамічною конфігурацією, створеного мовами C# та JavaScript.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати існуючі сучасні аналогічні програмні забезпечення та виявити їх переваги та недоліки. І в результаті побудувати порівняльну таблицю.
2. Розробити вимоги до системи що розробляється базуючись на аналізі переваг та недоліках проаналізованих аналогів.
3. Дослідити програмні засоби та технології, які можуть бути використані при розробці системі відповідно до поставленої задачі.
4. Розробити програмне забезпечення для спрощення створення та конфігурування власного чат-бота згідно з поставленою задачею.
5. Провести тестування розробленої системи.
6. Пройти апробацію на Науково-технічних конференціях;

Практичне значення результатів: розроблений чат-бот оптимізує процес комунікації малого та середнього бізнеса з користувачами.

Галузь використання - малий та середній бізнес.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

DDD	Domain-driven design
TDD	Test-driven design
XML	extensible markup language.
JSON	JavaScript Object Notation
API	Application Programming Interface
REST	Representational State Transfer
ОС	Операційна система
HTML	Hypertext Mark-up Language
SQL	Structured Query Language
DAL	Data access layer
SDK	Software development kit
DLL	Dynamic link library
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
СУБД	Система управління базами даних
SOLID	Набір принципів побудови коду Single responsibility principle, open-closed principle, Liskov substitution principle, interface segregation principle, and dependency inversion principle.
IDE	Integrated development environment
ООП	Об'єктно-орієнтоване програмування
LINQ	Language-Integrated Query
JWT	JSON Web Tokens
UI	User Interface

ВСТУП

Обґрунтування вибору теми та її актуальність: З розвитком інтернету та мобільних пристроїв величезної популярності набули месенджери. У сучасному світі такі додатки як Telegram, What's App, Viber успішно замінюють дзвінки через стільниковий зв'язок та смс.

Більшість популярних месенджерів мають змогу інтегрувати в них свії код у вигляді чат-бота. Бот зможе отримувати повідомлення з чату, відправляти власні згенеровані повідомлення, адмініструвати групи, публікувати статті в канал, тощо

Використання чат-ботів в месенджерах стає все більш популярним як серед великих компаній, так і серед малого та середнього бізнесу. Вони можуть виконувати різноманітні функції - від відповідей на типові питання до проведення складних операцій, таких як бронювання, продажі, підтримка користувачів тощо.

Однак, створення чат-бота може виявитися складним процесом, що вимагає спеціальних знань та навичок, а існуючі платформи для автоматизації процесу розробки боту не надають достатній контроль та гнучкість для конфігурування поведінки бота.

Об'єкт дослідження - процес комунікації малого бізнесу з клієнтами.

Предмет дослідження – чат-бот з динамічним конфігуруванням.

Мета роботи - підвищення ефективності комунікації клієнтів за рахунок Telegram чат-боту з динамічною конфігурацією, створеного мовами C# та JavaScript.

Для реалізації поставленої мети потрібно вирішити наступні завдання:

1. Проаналізувати існуючі сучасні аналогічні програмні забезпечення та виявити їх переваги та недоліки. І в результаті побудувати порівняльну таблицю.
2. Розробити вимоги до системи що розробляється базуючись на аналізі переваг та недоліках проаналізованих аналогів.
3. Дослідити програмні засоби та технології, які можуть бути використані при розробці системі відповідно до поставленої задачі.

4. Розробити програмне забезпечення для спрощення створення та конфігурування власного чат-бота згідно з поставленою задачею.

5. Провести тестування розробленої системи.

6. Пройти апробацію на Науково-технічних конференціях;

Практичне значення результатів: розроблений чат-бот оптимізує процес комунікації малого та середнього бізнеса з користувачами.

Для розробки застосунку використовуються мови програмування C# та JavaScript, з використанням фреймворків asp.net core 7 та Vue.js 3.

Практична значущість результатів: Розроблене програмне забезпечення що спрощує створення та конфігурування власного чат-бота.

Галузь використання - малий та середній бізнес.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Потреба бізнесу в чат-ботах

Розробка додатку у вигляді чат-боту з основним пріорітетом на зручне, просте та гнучке налаштування - це можливість для власників малого бізнеса легко інтегруватися в популярний месенджер з ціллю рекламування та продаж товарів, адміністрування інтернет-ресурсів, технічна підтримка користувачів, розважання користувачів, реалізувати інтеграцію в інші ресурси тощо.

Однією з основних переваг чат-боту є можливість зниження витрат на розробку та впровадження продукту, оскільки він не вимагає створення окремого інтерфейсу та забезпечує зберігання історії чатів, медіа-файлів та іншої інформації на серверах месенджера. Також не менш важливою перевагою є непотреба в окремому інтерфейсі, що знижує вартість розробки та підтримку продукту.

Даний чат-бот відрізняється від інших продуктів цієї галузі своєю гнучкістю. Клієнт отримує готовий продукт, який модна легко гнучко налаштувати та допрацьовувати під свої потреби власноруч. Також, дана платформа дозволяє здійснювати взаємодію з клієнтами через месенджер, що зменшує навантаження на співробітників та дозволяє зосередитися на важливих бізнес-операціях.

Однією з особливостей цієї технічної задачі є необхідність надання клієнтам гнучкості та можливості налаштування функціоналу під їхні потреби. Це дозволяє власникам бізнесу забезпечити індивідуальний підхід до своїх клієнтів та використовувати продукт в різних сферах бізнесу[1]. Для досягнення цієї мети потрібно розробити гнучку архітектуру, яка дозволить додавати та видаляти функції в залежності від потреб бізнесу.

Однією з відмінних особливостей цього продукту порівняно з іншими продуктами цієї галузі є гучкість і при цьомуможливість налаштування функціоналу не потребує особливих технічних знань. Таким чином, клієнти можуть самостійно

змінювати налаштування продукту та додавати чи видаляти функції в залежності від своїх потреб, не поглиблюючись в архітектурні проблеми, залишаючи акцент уваги на бізнес вимогах.

Чат-бот має безліч інших варіантів використання, серед яких можна виділити наступні[2]:

- Розсилання рекламних повідомлень. Чат-бот може бути використаний для розсилання рекламних повідомлень клієнтам. Він може бути налаштований для відправки інформації про новинки, акції, знижки та інші пропозиції.
- Автоматизація продажів. Бот може бути налаштований для автоматичної обробки замовлень, збору замовлень та відправки їх на склад для обробки. Це допоможе зменшити час, потрібний для обробки замовлень та підвищити ефективність роботи.
- Організація опитувань та опитувальних листів. Бот може бути використаний для проведення опитувань та опитувальних листів, які можуть бути корисними для збору даних про клієнтів та їхні потреби.
- Відстеження замовлень. Бот може бути налаштований для відстеження замовлень та надання клієнтам інформації про статус їхнього замовлення.
- Розваги для користувачів. Бот може бути налаштований для надання розваг для користувачів, таких як квести, ігри та інші форми розваг.
- Реалізація інтеграції з іншими сервісами. Бот може бути налаштований для інтеграції з іншими сервісами, такими як CRM системи, сервіси розсилки повідомлень, або інші системи управління бізнесом. Це допоможе автоматизувати роботу та підвищити ефективність бізнесу.
- Моніторинг стану системи. Бот може бути використаний для моніторингу стану системи, що дозволить оперативно виявляти проблеми та вживати необхідні заходи для їх вирішення.

- Організація технічної підтримки. Бот може бути налаштований для надання технічної підтримки користувачам, що дозволить швидко та ефективно вирішувати технічні питання.
- Підвищення лояльності клієнтів. Бот може бути використаний для підвищення лояльності клієнтів, надаючи їм персоналізовані пропозиції та допомагаючи вирішувати проблеми.
- Оптимізація роботи з соціальними мережами. Бот може бути налаштований для оптимізації роботи з соціальними мережами, надаючи користувачам можливість здійснювати публікації, розміщувати рекламу, відстежувати статистику та багато іншого.
- Підвищення продуктивності роботи. Бот може бути використаний для підвищення продуктивності роботи, допомагаючи клієнтам здійснювати рутинні завдання та автоматизуючи процеси.

Цільовою аудиторією цього продукту є власники малого бізнесу, менеджери, технічні спеціалісти та смм-фахівці, які хочуть полегшити взаємодію з клієнтами та збільшити ефективність бізнес-операцій через платформу месенджера. Цей продукт дозволяє спростити процеси взаємодії з клієнтами та зменшити навантаження на співробітників, що виконують ці завдання. Окрім того, даний продукт може бути корисним для малого бізнесу з обмеженим бюджетом, оскільки він є дешевшим в обслуговувані порівняно з іншими продуктами цієї галузі.

Користувачами продукту можуть бути:

- Власники онлайн-магазинів, які хочуть підвищити ефективність продажів та підтримувати взаємодію з клієнтами.
- Компанії з сфери послуг, які шукають інструмент для покращення комунікації з клієнтами та підвищення рівня обслуговування.
- Компанії, які хочуть забезпечити швидку та якісну технічну підтримку своїм клієнтам.

- Сервіси з онлайн-навчання, які шукають інструмент для покращення комунікації зі студентами та швидкого надання відповідей на запитання.
- Компанії, які займаються туризмом, які шукають інструмент для взаємодії з клієнтами та підтримки під час подорожей.
- Фінансові установи, які шукають інструмент для взаємодії з клієнтами та швидкого відповіді на запитання.
- Організації, які займаються благодійністю та шукають інструмент для комунікації з донорами та підписниками.
- Власники ресторанів та кафе, які хочуть забезпечити ефективну комунікацію зі своїми клієнтами та приймати замовлення через чат-бота.
- Фрілансери, які хочуть покращити комунікацію з клієнтами та забезпечити швидкий відгук на запитання.
- Стартапи, які шукають інструмент для підвищення ефективності та швидкості внутрішніх процесів.

1.2. Dialogflow

Dialogflow - це програмний засіб для розробки чат-ботів, що базується на штучному інтелекті та нейронних мережах Google. Він дозволяє легко створювати та налаштовувати інтерактивних розмовних агентів для різних платформ, включаючи Facebook Messenger, Telegram, Viber та інші[3].

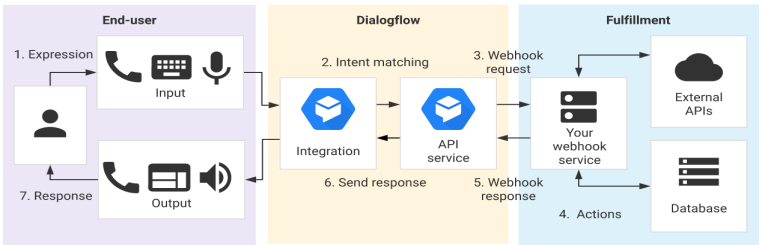


Рисунок 1.1 – Приклад схеми послідовності подій в Dialogflow

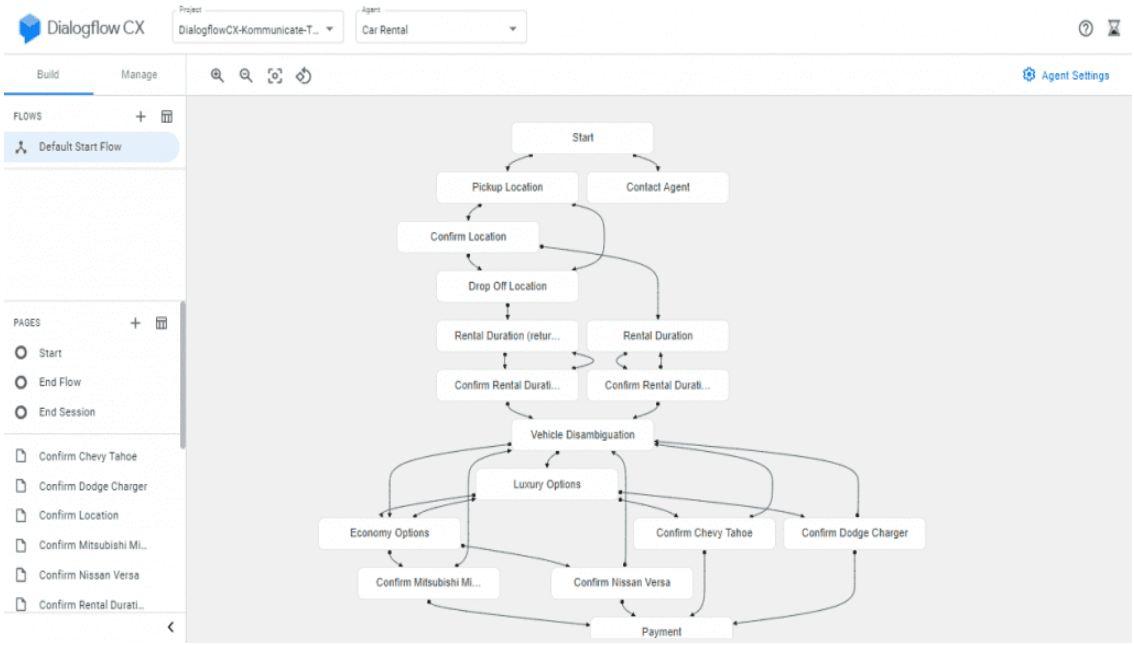


Рисунок 1.2 – Веб-інтерфейс Dialogflow для побудови схеми послідовності подій для користувача

Переваги:

- Легкий та зручний інтерфейс для розробки та налаштування чат-ботів.
- Можливості інтеграції з різними месенджерами.
- Високий рівень штучного інтелекту для побудови шаблонів за рахунок використання нейронних мереж.
- Велика кількість доступних шаблонів та підтримка додаткових інтеграцій та API.
- Інтеграція з іншими сервісами Google, такими як Google Assistant та Google Cloud Platform

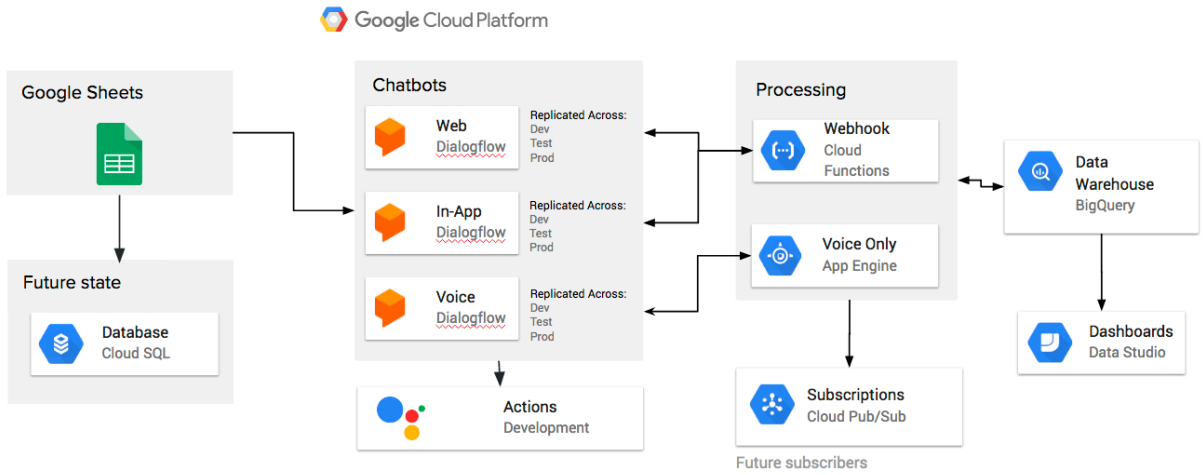


Рисунок 1.3 – Схема інтеграції Dialogflow з іншими сервісами компанії Google

Недоліки:

- Обмежена функціональність без підписки на платну версію.
- Потреба в технічному розумінні при налаштуванні.
- Неможливість розгортання на власних серверах
- Висока вартість в порівнянні з іншими засобами для створення чат-ботів.

Модель монетизації:

Dialogflow пропонує різні плани підписки, що відрізняються за кількістю доступних запитів на місяць та доступною функціональністю.

1.3. ManyChat

ManyChat - це інтерактивна платформа для створення чат-ботів з можливістю автоматизованої взаємодії з клієнтами на основі месенджерів[4].

Рисунок 1.4 – Веб-інтерфейс ManyChat для побудови схеми послідовності подій для користувача

Переваги:

- Простота використання.
- Можливість інтеграції з популярними месенджерами.
- Широкий набір вбудованих інструментів та функціонал для створення різноманітних чат-ботів.
- Можливість побудови детальної аналітики використання чат-бота.

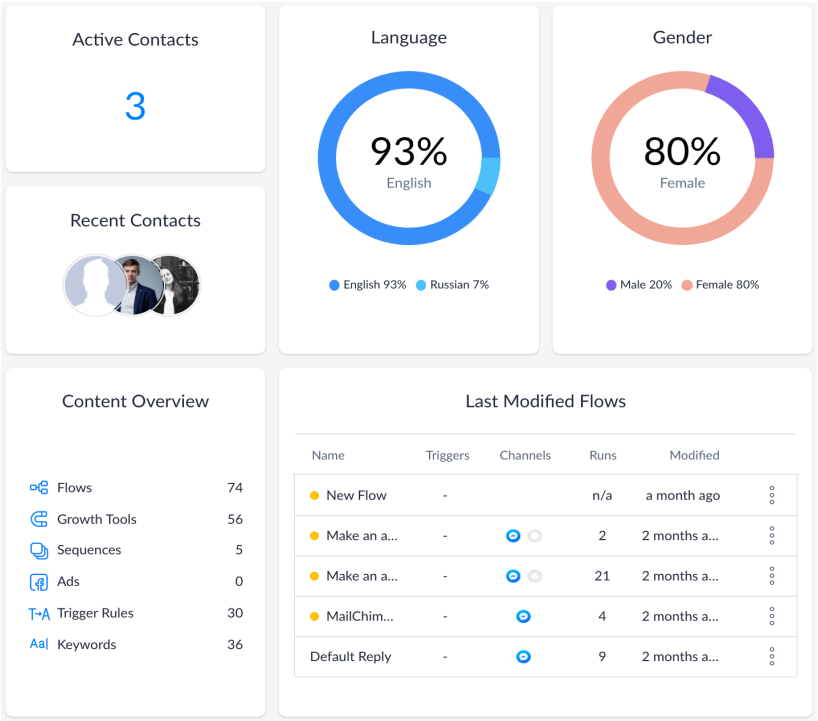


Рисунок 1.5 – Приклад аналітики використання чат-бота створенного за допомогою ManyChat

Недоліки:

- Обмежена кількість інтеграцій з іншими сервісами та платформами.
- Відсутність можливості розгортання на власних серверах.

Модель монетизації:

Безкоштовний тариф з обмеженнями функціоналу та платна підписка від $10 на місяць з більш розширеним функціоналом.

1.4. Chatfuel

Chatfuel - платформа для створення чат-ботів, яка дозволяє швидко і легко створювати ботів без програмування. Вона має інтуїтивний інтерфейс та багатофункціональні інструменти для налаштування та взаємодії з користувачами[5].

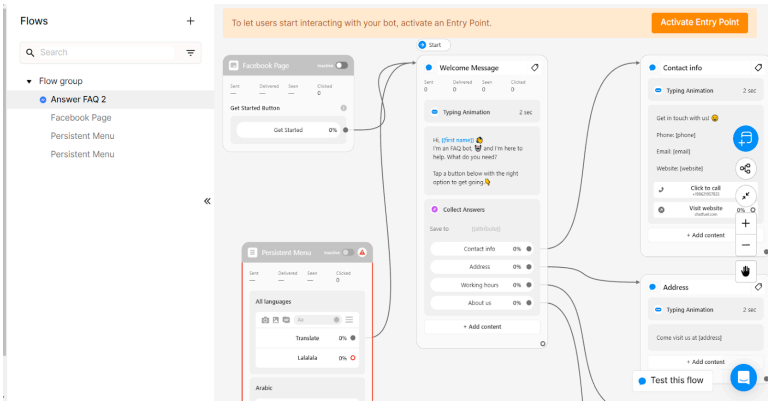


Рисунок 1.6 – Веб-інтерфейс Chatfuel для побудови схеми послідовності подій для користувача

Переваги:

- Простий та інтуїтивний інтерфейс;
- Багатофункціональні інструменти для створення чат-ботів;
- Підтримка більшості месенджерів;
- Велика бібліотека готових шаблонів.
- Можливість побудови аналітики використання чат-бота

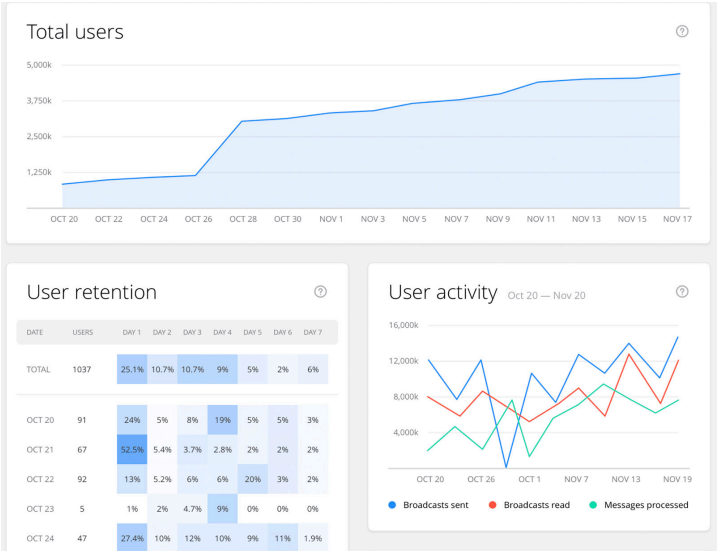


Рисунок 1.7 – Приклад аналітики використання чат-бота створенного за допомогою Chatfuel

Недоліки:

- Обмежені можливості аналітики;
- Обмежена можливість кастомізації
- Обмежена підтримка мов: на даний момент платформа підтримує всього 13 мов, що може бути недостатньо для деяких бізнесів з глобальною аудиторією.
- Відсутність підтримки власних API: Chatfuel не дозволяє користувачам розробляти власні API-інтеграції, що може бути необхідно для деяких специфічних випадків використання.
- Неможливість розгортання на власних серверах.

Модель монетизації:

Chatfuel пропонує безкоштовний та платний плани. Платний план дозволяє користувачам отримати додаткові функції та підтримку.

1.5. Microsoft Bot Framework

Microsoft Bot Framework - це відкрита платформа для створення, розгортання та управління чат-ботами. Цей фреймворк має великий набір інструментів, включаючи SDK для .NET та Node.js, візуальний редактор та конструктор діалогів. Microsoft Bot Framework також має готові шаблони для створення чат-ботів, що спрощує процес розробки[6].

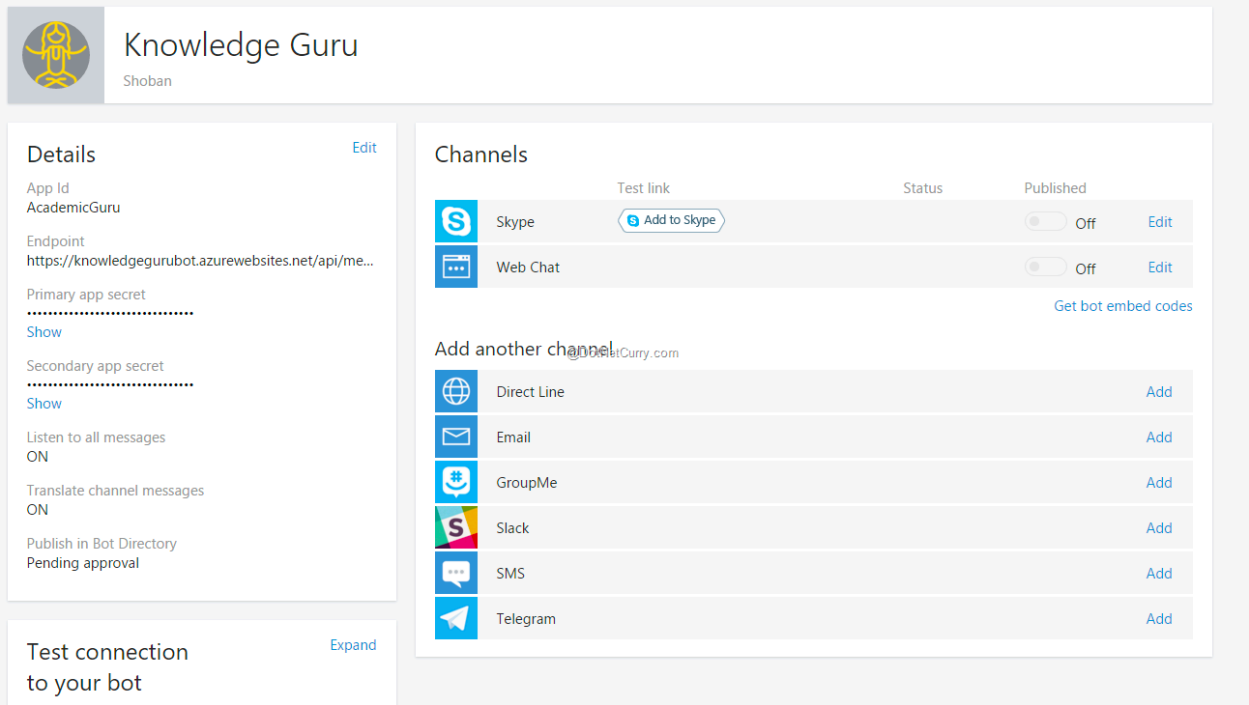


Рисунок 1.8 – Налаштування сервісів для інтегрування Microsoft Bot Framework

Переваги:

- Можливість розгортання використовуючи оркестрацію Kubernetes.
- Інтеграція з іншими популярними сервісами Microsoft, такими як Office 365 та Skype.
- Можливість використовувати штучний інтелект та машинне навчання для покращення роботи чат-ботів.
- Підтримка багатьох мов, включаючи Python, Java та PHP.
- Можливість інтегруватися з системами інтернет-речей.

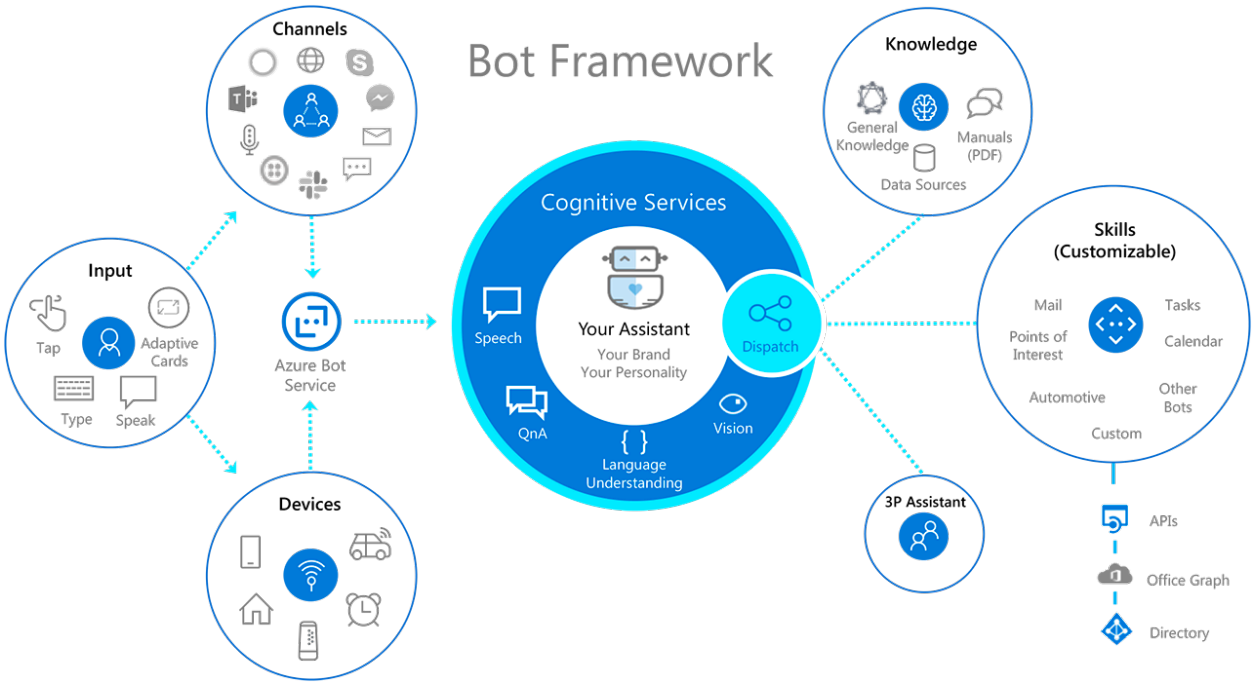


Рисунок 1.9 – Схема можливостей обробки даних платформою Microsoft Bot Framework

Недоліки:

- Потребує серйозних технічних знань для налаштування.
- Переважна більшість інструментів та сервісів платні.
- Неможливість розгортання на власних серверах.

Модель монетизації:

Microsoft Bot Framework пропонує різні тарифні плани, серед яких є безкоштовний варіант для розробників та платні плани для підприємств та організацій. Вартість платних планів залежить від обсягу та типу використовуваних сервісів і починається від $0.50 за 1000 запитів.

1.6. Amazon Lex

Amazon Lex є інструментом створення чат-ботів від Amazon Web Services[7]. Цей інструмент має декілька важливих переваг:

Переваги:

- Підтримується інтеграція з іншими Amazon Web Services, такими як Amazon S3 і Amazon DynamoDB, що дозволяє розширити функціональність боту.
- Amazon Lex використовує глибоке навчання, щоб автоматично навчатися розпізнавати мову клієнтів і аналізувати надходящі запити.
- Інтерфейс розробника Amazon Lex дуже зручний та інтуїтивно зрозумілий.

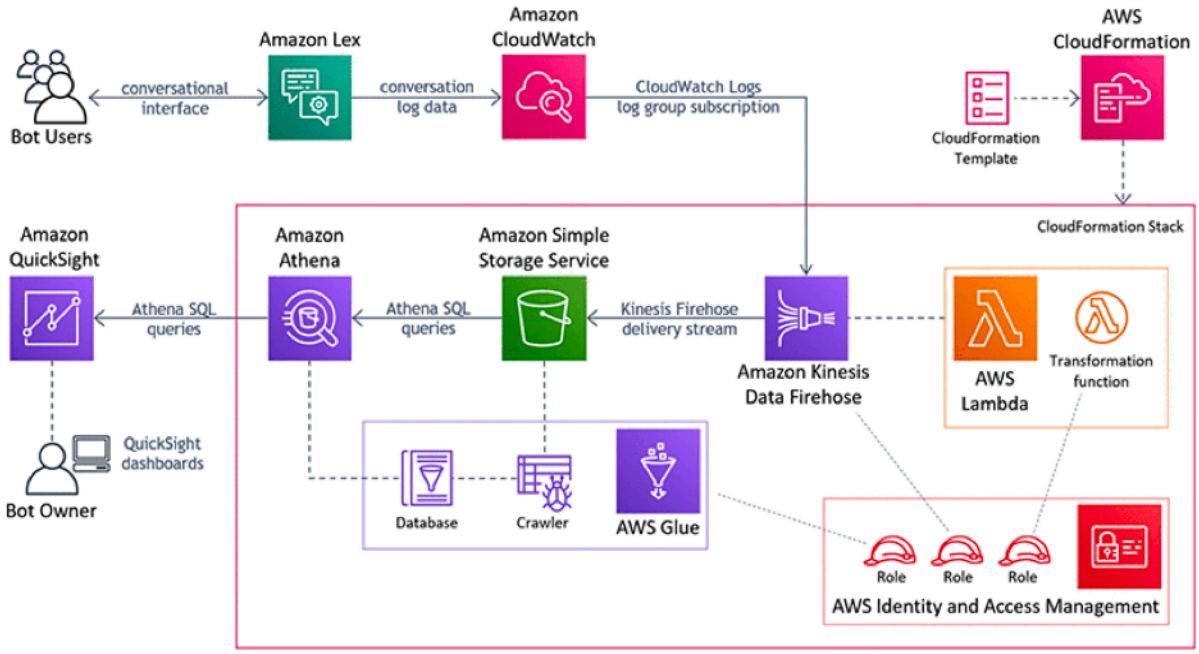


Рисунок 1.10 – Схема послідовності та можливостей інтегрування Amazon Lex з іншими сервісами компанії Amazon

Недоліки:

- Вимагає спеціальних знань з AWS, що може виявитися непростим для новачків у сфері розробки.
- Обмежена можливість розширення функціональності за межі інтеграції з Amazon Web Services.
- Неможливість розгортання на власних серверах

Модель монетизації:

Amazon Lex пропонує платити за кількість запитів до бота. Це означає, що вартість підписки залежить від кількості використання бота. Підписка на платформу може коштувати від $0,004 за запит.

Таблиця 1.1 - Характеристики сервісів для створення чат-ботів

Характеристика	**Dialog flow**	**Many Chat**	**Chatfuel**	**Microsoft Bot Framework**	**Amazon Lex**	**Boobot**
Можливість інтегрування власного коду	+	-	-	+	+	+
Можливість розвертання на своїх серверах	- (Google Cloud)	-	-	- (Azure)	- (AWS)	+
Налаштування без знання мов програмування	-	+	+	-	-	-
Інтеграція з месенджером telegram	+	+	+	+	+	+
Наявність штучного інтелекту для обробки запитів	+	+	+	+	+	-

Продовження таблиці 1.1 - Характеристики сервісів для створення чат-ботів

Аналітика користування чат-ботом	-	+	+	-	-	+
Інтегрування з будь-якими інфраструктурними сервісами (типу S3, Google Cloud Storage) інших провайдерів	-	-	-	-	-	+

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Технічне завдання

Розробити бекенд сервіс мовою C# з використанням платформи ASP.NET Core який буде комунікувати з базою даних, обробляти HTTP запити клієнта-адміністратора та вебхук запити Telegram Bot API для обробки повідомлень користувача в чаті, розробити механізм динамічної компіляції C# коду команд і фронтенд сторінки для авторизації та адміністрування конфігурації бота.
Розробити Docker підтримку для бекенд-сервісу для можливості упакування чат-бота в контейнер і подальшого його розгортання на серверах за допомогою технологій Docker, Kubernetes або Docker-compose.

Для спрощення подальшої підтримки коду та забезпечення більш гнучкої розробки, сервіс повинен бути розроблений згідно архітектурної парадигми DDD (Domain-Driven Design) та дотримуватися SOLID принципів побудови коду. Розробка повина виконуватися згідно комбінування DDD та TDD парадигм для забезпечення самодокунтації коду та ретельності покриття юніт-тестами проекту.
В ролі основної реляційної бази даних використовується сучасна СУБД PostgreSQL, для облегчення взаємодії з базою даних на рівні репозиторіїв використовуються технології Microsoft Entity Framework та LinqToSql.

Кеш-сховищем даних було обрано сучасну нереляційну базу даних з форматом даних “ключ-значення” Redis для досягнення максимальної швидкості доступу до даних необхідних для роботи чату.

Для логування системи використовується технологія GrayLog для менеджменту логів за допомогою бази даних ElasticSearch. Транспортування логів до ElasticSearch відбувається за допомогою технології брокеру повідомлень Apache Kafka, що забезепечує максимальну продуктивніть навіть на значних об’ємах даних.

Як середовище для розробки використовувати IDE від JetBrains - Rider, для налаштування та іншої взаємодії з PostgreSQL субд використовувати бд-клієнт DataGrip. Kafka-ui використовувати для налагодження брокеру повідомлень Apache Kafka. Для тестування HTTP адрес бекенд-сервісу використовувати Postman що дозволяє імітувати DTO які буде надсилати фронтенд код чи месенджер telegram застосовуючи вебхук ендпоінти чат-бота.

2.2. Функціональні вимоги

Аутентифікація адміністратора. Адміністратор повинен мати змогу авторизуватися лише за допомогою імені та паролю вказаних в конфігурації (або сконфігурованих за замовчуванням) при розгортанні бота щоб забезпечити надійний рівень безпеки та запобігти несанкціонованого доступу до налаштувань бота стороніми особами. При невдалій спробі авторизації адміністратор повинен бачити індикацію що ввів неправильні дані.

Динамічне створення нових команд. Адміністратор повинен мати змогу створювати нові команди налаштовуючи назву для ідентифікування команди, її аліас (назва для вказання в повідомленні) щоб реагувати на повідомлення користувача в чаті та ідентифікувати команду для подальшого її виконання, і код команди який безпосередньо повинен виконуватися при отриманні повідомлень користувача.

Налаштування існуючих динамічних команд. Повинна бути можливість редагування назви, аліасу та коду існуючих команд.

Валідація коду при збережені команди. Для запобігання збереження коду з помилками на рівні компіляції, код повинен перевірятися при створенні нової або редагуванні існуючих команд, не зберігатися у випадку наявності помилок, а також відображати помилки компіляції користувачу.

Вмикання та вимкнення існуючих команд. Повина бути можливість вимкнути та ввімкнути існуючу команду щоб вона не брала участь в опрацюванні запита користувача.

Видалення існуючих команд. Повина бути можливість видалення існуючих команд.

Виконання динамічно створених команд. Код сконфігурованих команд повинен компілюватися та запускатися щоб реагувати на повідомлення в чаті від користувачів. Основною можливістю динамічно виконуваного коду є генерування та відправка повідомлень від імені бота.

Підрахунок статистики. Повина рахуватися статистика використання команд користувачами для можливості аналітики популярності команди серед користувачів чату.

2.3. Нефункціональні вимоги

Продуктивність. Чат-бот повинен забезпечувати швидку відповідь на запити користувачів. Затримка відповіді повинна бути мінімальною, щоб забезпечити користувачам плавну та гладку взаємодію.

Масштабованість. Система повинна бути спроектована так, щоб вона могла легко масштабуватися з підвищенням кількості користувачів або обсягу запитів. Переважна підтримка горизонтального масштабування для балансування навантаження.

Безпека. Чат-бот повинен гарантувати безпеку даних користувачів. Всі персональні дані, що обробляються чат-ботом, повинні бути захищені від несанкціонованого доступу.

На транспортному мережевому рівні повинні використовуватися захищені протоколи як HTTPS.

Надійність. Система повинна бути надійною і забезпечувати стабільну роботу. Чат-бот повинен бути доступний для використання 24/7.

Простота використання. Інтерфейс чат-бота повинен бути інтуїтивно зрозумілим для користувачів різних рівнів технічної компетенції.

Ремонтопридатність. Система повинна детально та зрозуміло логуватися для оперативного виявлення помилок та максимально ефективного їх усунення для забезпечення максимальної надійності.

Підтримка та оновлення. Система повинна бути підтримувана та оновлюватися регулярно, щоб відповідати змінам технологій та потреб користувачів.

Тестування. Код проекту повинен бути покритий юніт-тестами для запобігання появи нових помилок в старому коді при оновленні проекту для полегшення подальшої розробки проекту та підвищення степені його надійності. Мінімальний прийнятний рівень покриття автоматичними юніт тестами 30%, однак іншими типами тестів, включаючи, ручне тестування покриття повино сягати 100%

Сумісність. Сервіси повинні підтримувати наступні показники сумісності:

- Система повинна бути сумісна з сучасними версіями операційних систем Linux, Windows Server
- Система повинна бути сумісна з архітектурами процесорів arm64, x64, x86
- Формат даних JSON;
- Архітектурний стиль REST.

Системні потреби. Мінімальні потреби для інфраструктури для забезпечення коректного та ефективного функціонування системи:

- **Операційна система:** сервер повинен працювати на Debian 11 або новішою версією. Рекомендовано встановити останні оновлення безпеки.
- **Апаратне забезпечення:** процесор: 64-бітний, 4 ядра, 2.0 ГГц або швидше
- **Оперативна пам'ять.** Мінімум 8 ГБ, рекомендовано 16 ГБ
- **Вільне місце на диску**: Рекомендовано 20 ГБ

- **Мережевий адаптер:** Ethernet (рекомендовано)

Програмне забезпечення

- **Framework для frontend:** Node.js 14 або новіша версія
- **База даних:** PostgreSQL 13 або новіша версія
- **Веб-сервер:** Nginx 1.18 або новіша версія
- **Система керування контейнерами:** Docker 20.10 або новіша версія

Мережеві вимоги

- Потрібне стабільне інтернет-з'єднання зі швидкістю не менше 10 Мбіт/с.
- Потрібний відкритий порт 443 для HTTPS трафіку.

2.4. Асоціативна мапа

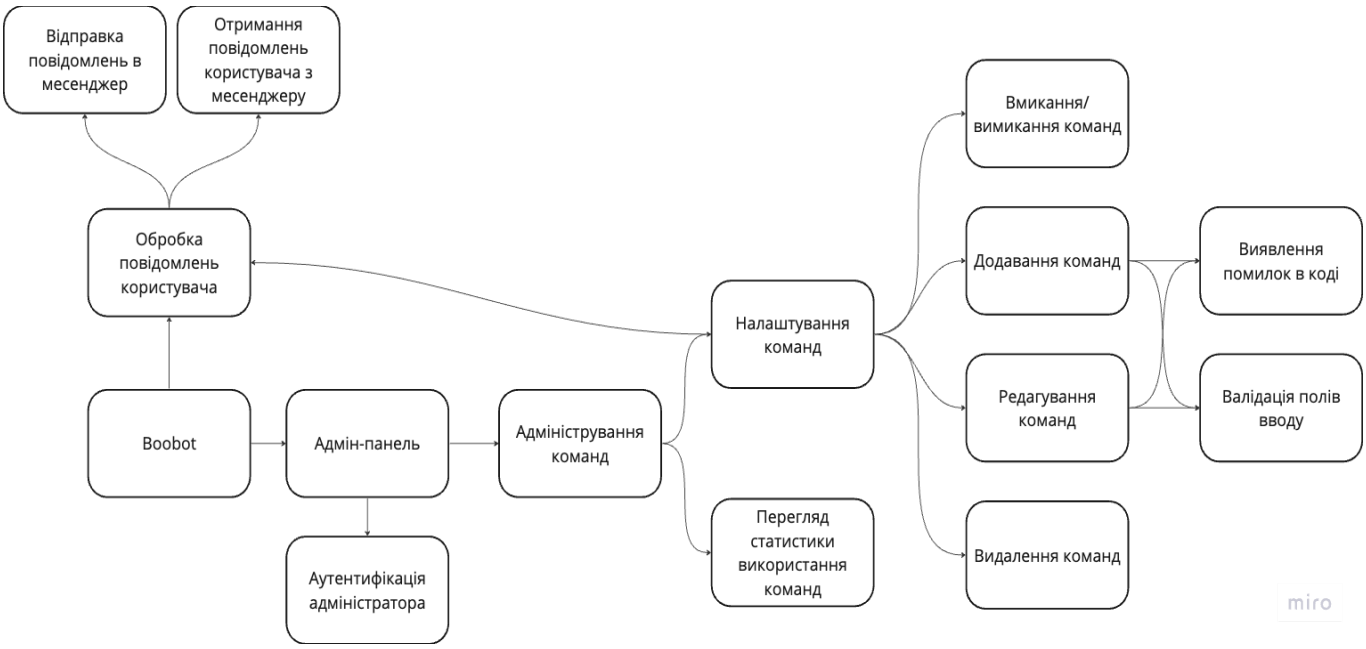


Рисунок 2.1 Асоціативна мапа - функціональне представлення системи

2.5. Діаграма класів використання

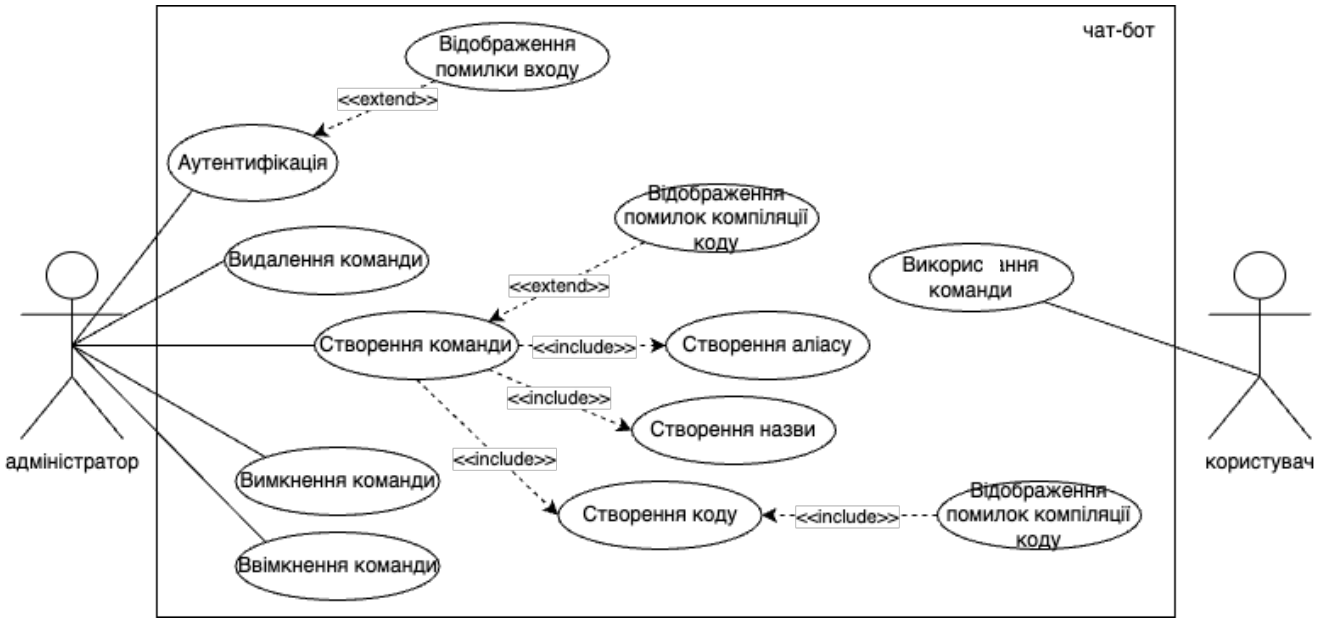


Рисунок 2.2 Діаграма використання

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Інструменти та засоби розробки

Мова програмування C# є сучасною, об'єктно-орієнтованою мовою програмування, яку розробила компанія Microsoft. Вона була створена з метою забезпечити легкість використання та ефективність в розробці програмного забезпечення. Ключові переваги та особливості:

- **Об'єктно-орієнтованість**: C# є повноцінною об'єктно-орієнтованою мовою програмування, що означає, що вона підтримує концепції спадкування, поліморфізму та інкапсуляції. Це спрощує процес розробки складного програмного забезпечення.
- **Безпечність типів**: C# має строгу безпечність типів, що допомагає забезпечити надійність та зменшити кількість помилок під час виконання.
- **Інтеграція з .NET**: C# була спеціально розроблена для роботи з платформою .NET, яка включає велику кількість бібліотек та інструментів для розробки програмного забезпечення. Це дає можливість розробникам швидко створювати потужні та ефективні додатки.
- **Інтероперабельність**: C# може взаємодіяти з багатьма іншими мовами та платформами, що робить її дуже гнучкою для розробки різноманітних видів додатків.
- **Автоматичне керування пам'яттю**: Завдяки використанню сміттєзбірника (Garbage Collector) розробник може уникнути ручного вивільнення пам'яті і управляти ресурсами програми більш ефективно. Сміттєзбірник дозволяє спростити процес роботи з пам'яттю і забезпечує автоматичне усунення непотрібних об'єктів, звільняючи розробника від необхідності прямого контролю над цим процесом.

- **Многопоточність та асинхронність**: C# має вбудовані конструкції для роботи з многопоточністю та асинхронним програмуванням.

ASP.NET Core - це фреймворк для розробки веб-додатків і служб, який має деякі особливості та переваги. Ключові переваги та особливості:

- **Хмарна сумісність**: ASP.NET Core 7.0 підтримує розгортання в хмарних середовищах, таких як Azure, AWS і Google Cloud. Це дозволяє розробникам швидко та зручно розгортати свої додатки в хмарних оточеннях.
- **Кросплатформеність**: Фреймворк є переносимим і може працювати на різних платформах, включаючи Windows, Linux і macOS. Це дає розробникам велику гнучкість вибору платформи для розгортання своїх додатків.
- **Модульність**: Фреймворк має модульну структуру, що дозволяє розробникам вибирати тільки необхідні компоненти для їх проекту. Це сприяє зменшенню розміру додатків та покращує продуктивність.
- **Web API підтримка**: ASP.NET Core 7.0 надає потужну підтримку для розробки веб-служб за допомогою технології Web API. Це дозволяє легко створювати та експонувати RESTful API для взаємодії з іншими додатками і службами.
- **Висока продуктивність**: ASP.NET Core 7.0 має оптимізовану архітектуру, що забезпечує високу продуктивність веб-додатків. Він може працювати з високими навантаженнями і масштабуватися залежно від потреб проекту.

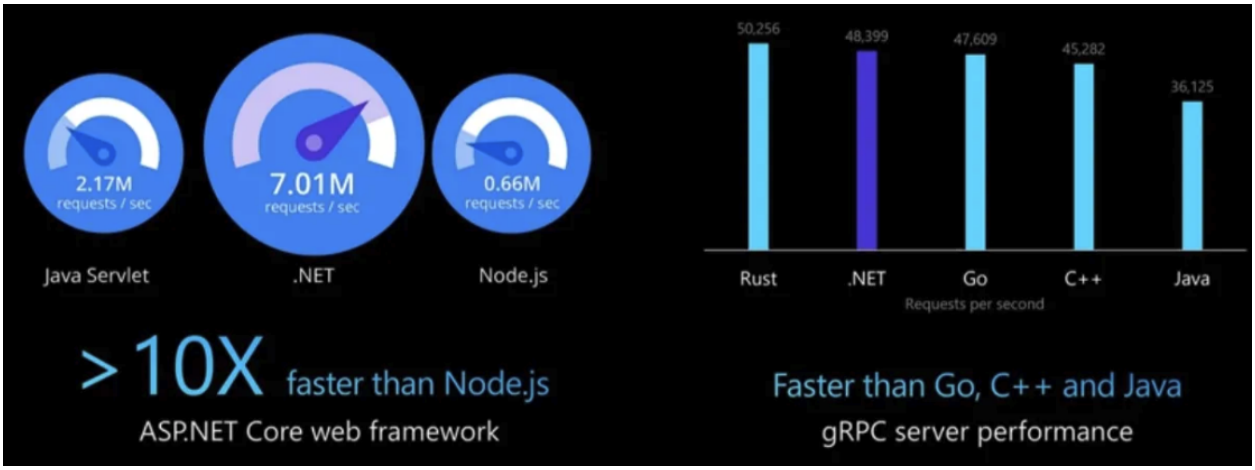


Рисунок 3.1 - Швидкість ASP.NET Core в порівнянні з іншими фреймворками[9]

- **Вбудована захист від атак**: ASP.NET Core 7.0 має вбудовані заходи безпеки, що допомагають захистити додатки від різних видів атак, таких як перехоплення даних, введення шкідливого коду (injection), міжсайтовий скриптинг (XSS), перекручування параметрів і багато інших.
- **Автентифікація та авторизація**: ASP.NET Core 7.0 надає розширену підтримку для автентифікації та авторизації. Розробники можуть використовувати різні механізми аутентифікації, такі як JWT (JSON Web Tokens), OAuth і інші, для забезпечення безпеки доступу до своїх додатків та ресурсів.
- **Захист даних**: Фреймворк має вбудовану підтримку шифрування даних та безпечного зберігання конфіденційної інформації, такої як паролі або особисті дані користувачів. Це допомагає запобігти несанкціонованому доступу до цінних даних.
- **Захист від перевантажень та атак на доступність**: ASP.NET Core 7.0 надає механізми для захисту від перевантажень (DDoS) та інших атак на доступність. Він має вбудовану підтримку обмеження швидкості, кешування, обробки помилок та інших технік, які допомагають підтримувати стабільну роботу додатків під великим навантаженням та захищати їх від атак.

JavaScript – це мова програмування з динамічною типізацією, що широко використовується для створення інтерактивних веб-сторінок. Мова підтримує різні парадигми програмування, включаючи процедурне, об'єктно-орієнтоване (прототипне) і функціональне програмування.

Основні переваги:

- **Універсальність**. Широко застосовується в розробці веб-додатків, в тому числі для бекенд частини системи завдяки платформі Node js
- **Популярність та розширюваність**. Має велику кількість готових до використання бібліотек та фреймворків, що допомагають розробникам в різних задачах.
- **Асинхронність**. JavaScript має вбудовану підтримку асинхронного програмування.

Основні недоліки:

- **Неоднорідність**. JavaScript може мати різну поведінку в різних браузерах, що може призвести до помилок.
- **Неочевидність**. Мова має кілька незвичних особливостей і "пасток", що може призвести до помилок, якщо розробник не досить добре знайомий з мовою.
- **Важка контрольованість**. Великі JavaScript-проекти можуть бути важкими для управління без використання додаткових інструментів або фреймворків.

Vue.js - це прогресивний JavaScript-фреймворк для створення користувацьких інтерфейсів. Ряд переваг Vue.js в порівнянні з «чистим» JavaScript, які можуть полегшити розробку веб-застосунків:

- **Компонентна структура**: Vue.js використовує систему компонентів, яка дозволяє розбивати застосунок на повторювані, самодостатні частини. Це сприяє більшому перевикористанню коду та полегшує його підтримку.

- **Декларативне рендеринг**: Vue.js дозволяє створювати UI, описуючи, як UI повинен виглядати залежно від стану додатку, не турбуючись про те, як саме ці зміни досягаються.
- **Реактивність**: Vue.js автоматично стежить за змінами у ваших даних і оновлює DOM відповідно, що знижує необхідність прямого керування DOM та написання коду для оновлення інтерфейсу.
- Інтеграція: Vue.js можна легко інтегрувати з існуючим проєктом на мові JavaScript, що не потребує повного переписування коду.
- **Спільнота та екосистема**: Vue.js має сильну спільноту розробників, яка постійно розвиває та підтримує додаткові бібліотеки та інструменти, такі як Vue Router, Vuex та інші.
- **Документація**: Vue.js має відмінну документацію, що полегшує вивчення та використання фреймворку.

Деякі недоліки Vue.js:

- **Кількість вакансій**. Незважаючи на свою популярність, Vue.js використовується меншою кількістю компаній порівняно з React або Angular, що може обмежити можливості знаходження роботи для розробників Vue.js.
- **Гнучкість**. Vue.js може бути менш потужним для деяких складних сценаріїв порівняно з іншими фреймворками, такими як Angular.

JetBrains Rider - це кросплатформене інтегроване середовище розробки (IDE) від компанії JetBrains. Він забезпечує продуктивність розробки коду для платформи .NET на різних мовах, однією з яких є C#.

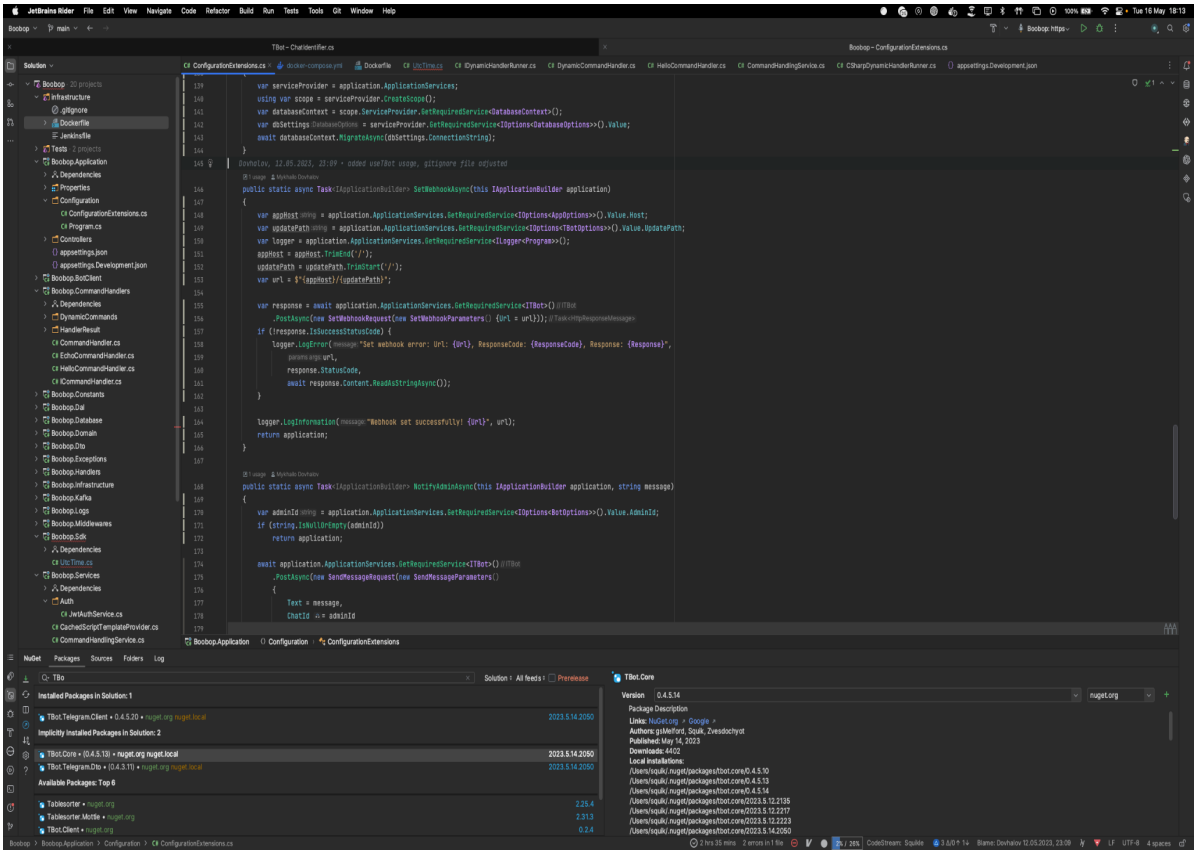


Рисунок 3.2 - UI JetBrains Rider

Основні переваги та особливості:

- **Універсальність**: підтримка мов C#, F#, VB.NET, ASP.NET (Razor View та ASPX), javascript, TypeScript, XAML, xml, sql, і багато інших.
- **Шаблони**: швидке створення проектів за допомогою шаблонів.
- **Відладка та профілювання**: JetBrains Rider має вбудовані інструменти для відлагодження та профілювання коду. Він дозволяє відстежувати й аналізувати виконання програми, знаходити й виправляти помилки, а також оптимізувати код.
- **Продуктивність**: JetBrains Rider має вбудовані інструменти для проведення

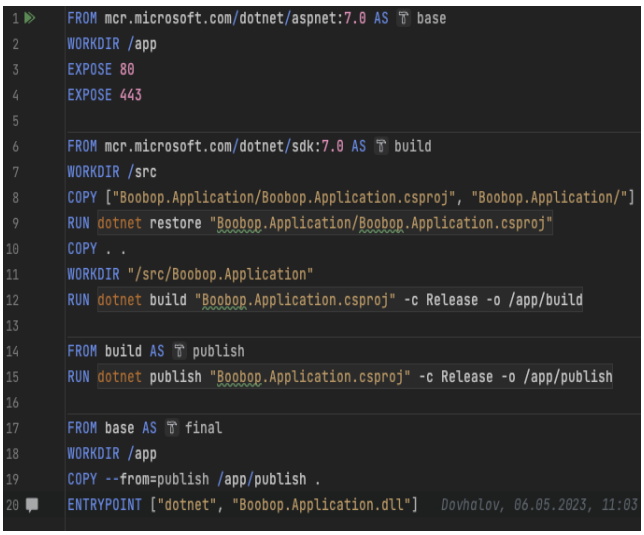
рефакторингу коду. Він дозволяє швидко перейменовувати змінні, методи, класи тощо, автоматично виправляти структуру коду.

- **Кросплатформеність**: JetBrains Rider підтримує різні операційні системи, включаючи Windows, macOS і Linux. Це дозволяє розробникам використовувати її на своїй улюбленій платформі.

JetBrains DataGrip - це кросплатформене інтегроване середовище розробки (IDE) для баз даних і SQL від JetBrains. Основні переваги та особливості:

- **Універсальність**: підтримує багато типів баз даних: PostgreSQL, MySQL, Oracle, SQLite і багато інших.
- **Продуктивність:** має потужний інструмент для аналізу SQL-коду з високоякісним автозаповненням.
- **Візуалізація баз даних**: можливість автоматичного створення діаграм для бази даних.

Docker - це відкритий проект, що полегшує та автоматизує розгортання додатків на серверних машинах використовуючи підхід контейнеризації.



```
FROM mcr.microsoft.com/dotnet/aspnet:7.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:7.0 AS build
WORKDIR /src
COPY ["Boobop.Application/Boobop.Application.csproj", "Boobop.Application/"]
RUN dotnet restore "Boobop.Application/Boobop.Application.csproj"
COPY . .
WORKDIR "/src/Boobop.Application"
RUN dotnet build "Boobop.Application.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "Boobop.Application.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "Boobop.Application.dll"]   Dovhalov, 06.05.2023, 11:03
```

Рисунок 3.3 - Docker файл для проєкту будується за допомогою команд docker, які детально описані в документації[11]

Основні особливості та переваги:

- **Уніфікація процесу розгортання**: Docker дозволяє розробникам пакувати додатки з усіма їх залежностями в стандартний контейнер, який можна запустити на будь-якому сервері з підтримуваною операційною системою та архітектурою процесора.
- **Відокремлення ресурсів**: кожен Docker контейнер працює відокремлено і має свою власну емульовану файлову систему. Також при розгортанні контейнера не залишається ніяких слідів в системі - для повного видалення ддодатку достатньо просто видалити контейнер
- **Висока продуктивність**: Docker контейнери легкі та швидкі, оскільки вони не мають накладних витрат віртуалізації
- **Docker-compose як частина Docker**: це інструмент для оркестрації та управління контейнеризованими додатками, який дозволяє легко описувати та запускати багатоконтейнерні середовища. Він базується на YAML-файлах для визначення компонентів, налаштування зв'язків та налаштування контейнерів.

```yaml
version: "3.9"

services:
  boobot.bot:
    container_name: boobot
    image: squikle/boobot:latest
    restart: always
    ports:
      - 5010:80
    environment:
      - Jwt:SecurityKey=b940ad2f-7cb4-4dc8-8482-1c60ecbefbdc
      - Database:ConnectionString=Host=database;Port=5432;Database=boobot_test;Username=admin;Password=SDFKjh3q487EBWe1q2k@!dlda32ase;
      - Kafka:Server=kafka:9092
      - Kafka:Prefix=boobotich.test
      - TBot:UpdatePath=api/telegram/update
      - TBot:BotToken=5872405524:AAFMlL8RlnIF3C9VmtOxDjnFTai6-VVWNCA
      - TBot:StoreConnectionString=redis,password=DSjh34o8DF3h41243kd,defaultDatabase=1,syncTimeout=300000
      - TBot:AdminId=243857110
      - App:Host=https://dev.boobot.xyz

networks:
  default:
    external:
      name: kafka_network   Dovhalov, 06.05.2023, 05:11 • added docker-compose file
```

Рисунок 3.4 - Конфігурація чат-бота в docker-compose.yaml файлі

Apache Kafka - це розподілена потокова платформа, яка дозволяє публікувати та підписуватися на потоки записів, зберігати потоки записів у відмовостійкий спосіб та обробляти потоки записів у міру їх надходження. Проте для досягнення максимальної ефективності та використання всього потенціалу технологія може бути доволі складною в налаштуванні та управлінні. Основні переваги та особливості:

- **Надійність**: kafka це розподілена система, яка є масштабованою, відмовостійкою та довговічною.
- **Швидкість:** одна з головних переваг Kafka полягає в його швидкості, а саме спроможності обробляти великі обсяги даних та надавати низьку затримку при передачі повідомлень.

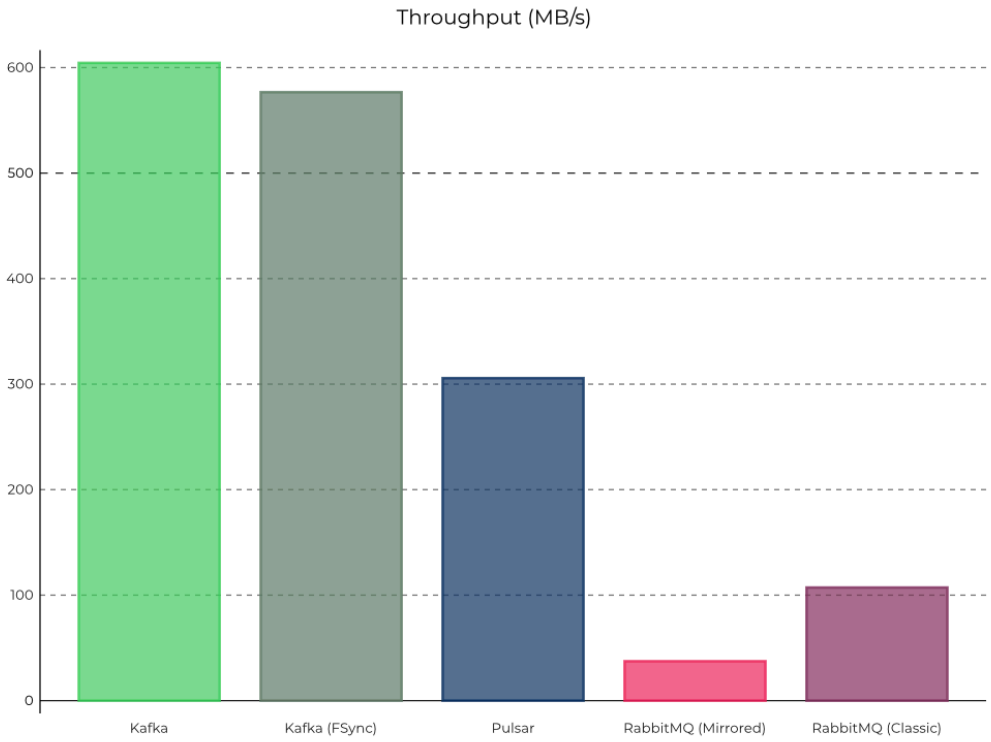


Рисунок 3.5 - Пропускна проможність Kafka порівнянні з конкуретними платформами[10]

3.2. Принципи побудови архітектури

Правильно продумана та побудована архітектура неймовірно важлива для майбутньої розробки проекту. Майбутній структурі коду варто приділити значну увагу ще на етапі проектування додатку. Правильно підібраний варіант архітектури забезпечує простоту масштабуванн та лагодження потенційних або реальних проблем та помилок в коді. Щонайменш варто дотримуватися доволі простих та дуже популярних 5 принципів SOLID, це гарантує правильне розмежування зон відповідальності, модульності додатку, а також значно зменшить шанс допущення помилок в старому функціоналі при додаванні нового.

3.3. Принципи написання коду

Існує багато принципів написання коду, але є трійка найпопулярніших з якими важко сперечатися. Ці принципи допомагають робити код системи простішим для розуміння і, відповідно, редагування та це дуже позитивно впливає на подальшу підтримку та доробку системи. Серед них:

YAGNI - *You Aren't Gonna Need It /* Вам це не знадобиться

Цей принцип простий і очевидний, але не всі його дотримуються. Якщо ви пишете код, то будьте впевнені, що він вам знадобиться. Не пишіть код, якщо думаєте, що він знадобиться пізніше.

Цей принцип був використаний при розробці проекту для уникнення непотрібного коду, щоб не доводилося розбиратися в тому що навіть не використовується і не згадувати як це планувалося. Розробити щось нове з нуля та адаптувати під актуальний конекст задачі значно простіше.

DRY - Don’t repeat yourself / Не повторюйтеся

Цей принцип ґрунтується на ідеї єдиного джерела правди (single source of truth — SSOT). Він означає, що всі фрагменти даних обробляються (або редагуються) тільки в одному місці.

В даній системі ций принцип був дотриманий, щоб забезпечити надійність та однорідність коду, що позбавить від можливих неприємних ситуацій коли потрібно підправити код одночасно в декількох місцях і якщо забути це зробити, то можна допустити помилку і створити баг.

KISS - *Keep It Simple, Stupid / Не* ускладнюйте

Цей принцип був розроблений ВМС США в 1960 році. Він говорить про те, що прості системи працюють краще і надійніше.

При розробці даного проекту був використаний принцип KISS, вибираючи найпростіші та найефективніші рішення для задач, що виникали. Це допомогло нам уникнути непотрібної складності та забезпечити читабельність коду без зайвих ускладнень.

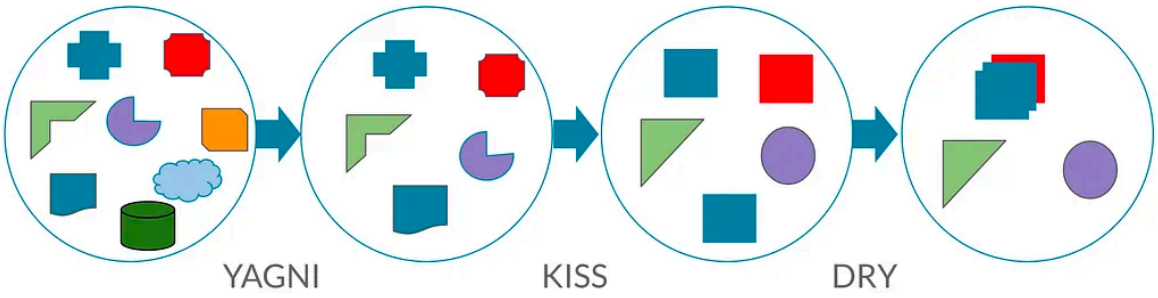


Рисунок 3.6 - Найпопулярніші принципи написання коду[12]

3.4. Дотримання Domain-driven Design

При розробці системи було дотримано архітектуру згідно DDD паттерну. В центрі уваги неначе ядром ставиться бізнес-модель, що являється доменом. Навкруги ядра додаються менш суттєві шари з точки зори бізнес-логіки шари, які мають доступ до внутрішніх шарів, але не навпаки. Основною задачею DDD є побудова єдиної моделі розуміння бізнес-моделі між розробником та бізнес-аналітиками, щоб максимально наближити код до предметної області, що дозволяє правильно ізолювати слої системи забезпечуючи чіткі межі відповідальності різних шарів коду.

Вцілому це популярний нині підхід, який незважаючи на додаткові витрати часу при розробці проєкту, дає значну користь в процесі подальшої розробки та розростання кодової бази системи.

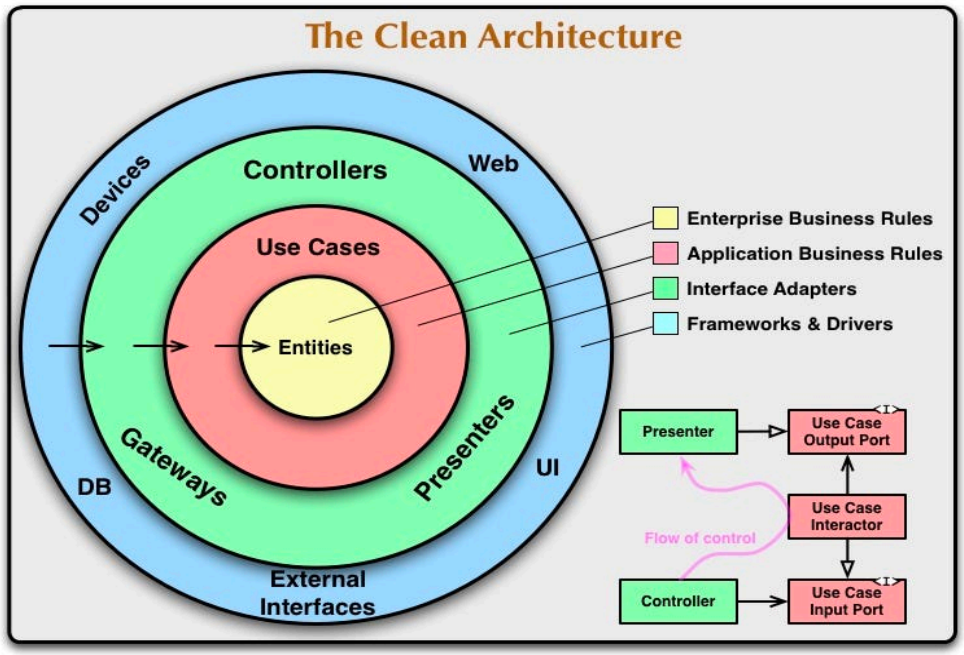


Рисунок 3.7 - Діаграма DDD архітектури[13]

3.5. Структура проєкту

Таблиця 3.1 - Модулі системи

Назва	Тип модуля	Призначення
Boobop.Application	Веб-додаток	Обробляє http запити з frontend та Telegram Bot API, збирає всю інфраструктуру до купи і є двигуном системи
Boobop.Domain	Бібліотека	Доменні домени системи
Boobop.Services	Бібліотека	Сервіси з бізнес логікою, управління та агрегування доменними моделями
Boobop.Infrastructure	Бібліотека	Шар доступу до сховища данних за допомогою платформи Entity Framework Core
Boobop.Utils	Бібліотека	Універсальні допоміжні методи які можуть знадобитися в різних місцях проєкту
Boobop.Domain	Бібліотека	Зберігаються домені моделі та інтерфейси
Boobop.Dto	Бібліотека	Об’єкти-моделі транспортного рівня для комунікування з Frontend частиною проєкту
Boobop.Exceptions	Бібліотека	Об’єкти даних виключень які можуть статися від час роботи системи у випадку нештатних ситуацій
Boobop.Middlewares	Бібліотека	Шари обробки запитів для рівню веб-додатку (обробка помилок, авторизація і т.п.)
Boobop.Settings	Бібліотека	Об’єкти конфігурації системи

Продовження таблиці 3.1 - Модулі системи

Boobop.Constants	Бібліотека	Константи для використання в різних частинах системи
Boobop.Database	Бібліотека	Сервіс-конфігуратор підключення до бази даних, code-first оновлення схеми бази даних
Boobop.Dal	Бібліотека	Об’єкти даних що зберігаються в базі даних. Використовуються для доступу до даних і подальшого конвертуваня в доменні моделі
Boobop.Logs	Бібліотека	Конфігуратор логування системи
Boobop.CommandHandlers	Бібліотека	Об’єкти-обробники команд за замовчуванням та рівень абстракції для динамічних обробників
Boobop.Sdk	Бібліотека	Проект з необхідними бібліотеками що дозволяє розробляти код команд за допомогою IDE для подальшого експорту в динамічні команди бота
Boobop.Domain.Tests	Бібліотека з тестами	Юніт-тести доменних моделей
Boobop.Services.Tests	Бібліотека з тестами	Юніт-тести системи шару сервісів

3.6. Схема бази даних

Для функціонування системи достатньо двух таблиць в базі даних:

Таблиця команд - зберігає необхідну інформацію про динамічну команду бота;

Таблиця статистики для кожної команди - статистика стосовно кожної з динамічних команд бота для можливості аналізування використання кожної з налаштованих команд. Таблиця статистики відноситься як 1 до 1 відносно таблиці команд.

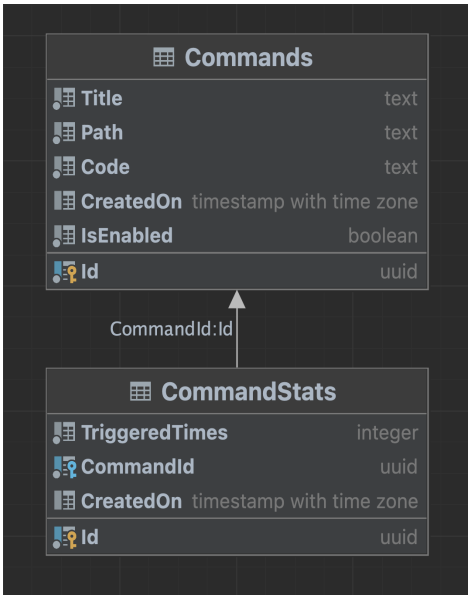


Рисунок 3.8 - Таблиці бази даних. Commands і CommandStats

3.7. Мікросервісна взаємодія

Розробляєма система використовує допоміжні засоби серед яких бази даних, агрегатори логів та брокери повідомлень тощо. Зазвичай подібні мікросервіси розгортаються на різних фізичних серверах.

Проектування міжсерверної архітектури важливий аспект. Потрібно створити правильний баланс безпеки, швидкості доступу та можливості горизонтального масштабування, при чому потрібно враховувати бюджет, адже аренда серверів це доволі значні витрати для бізнесу.

Для розгортання даної системи пропонується 2 схеми:

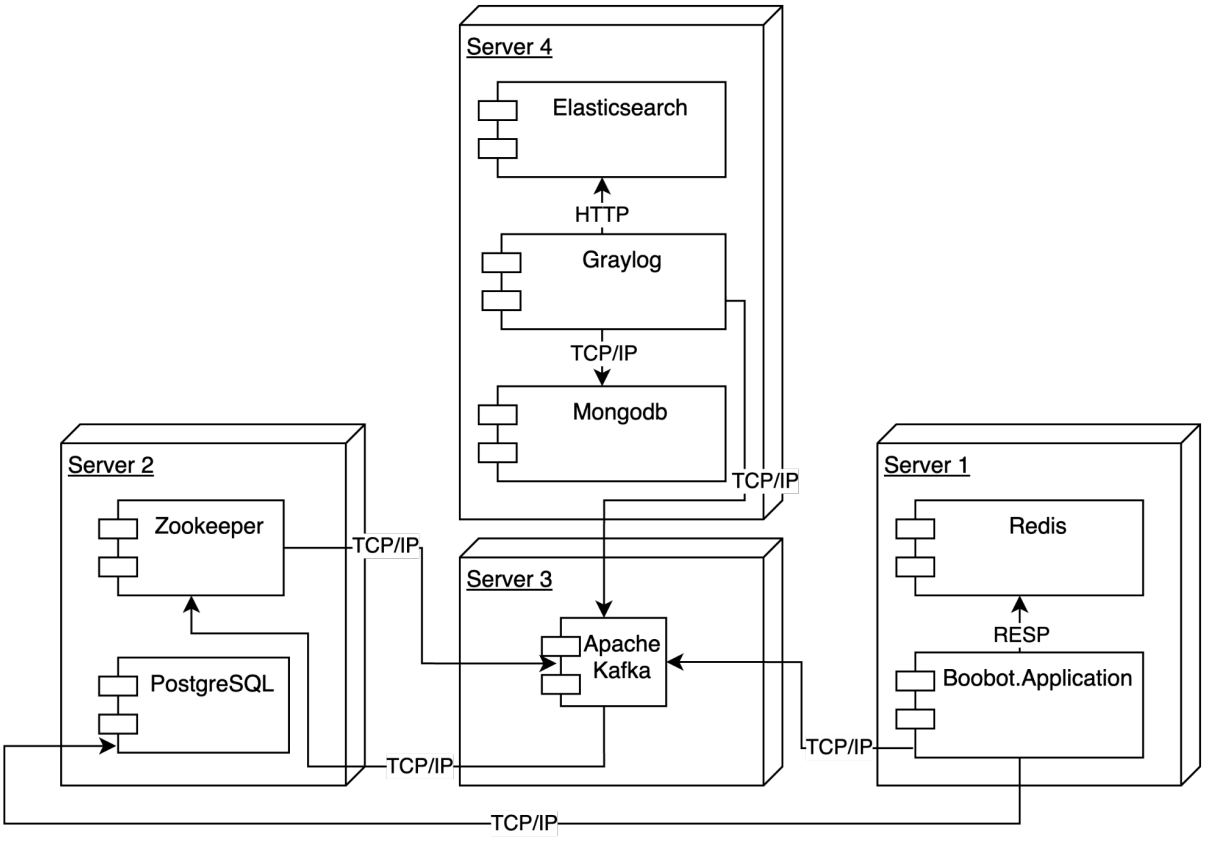


Рисунок 3.9 Рекомендована схема розгортання для забезпечення гнучкості розгорненної інфраструктури

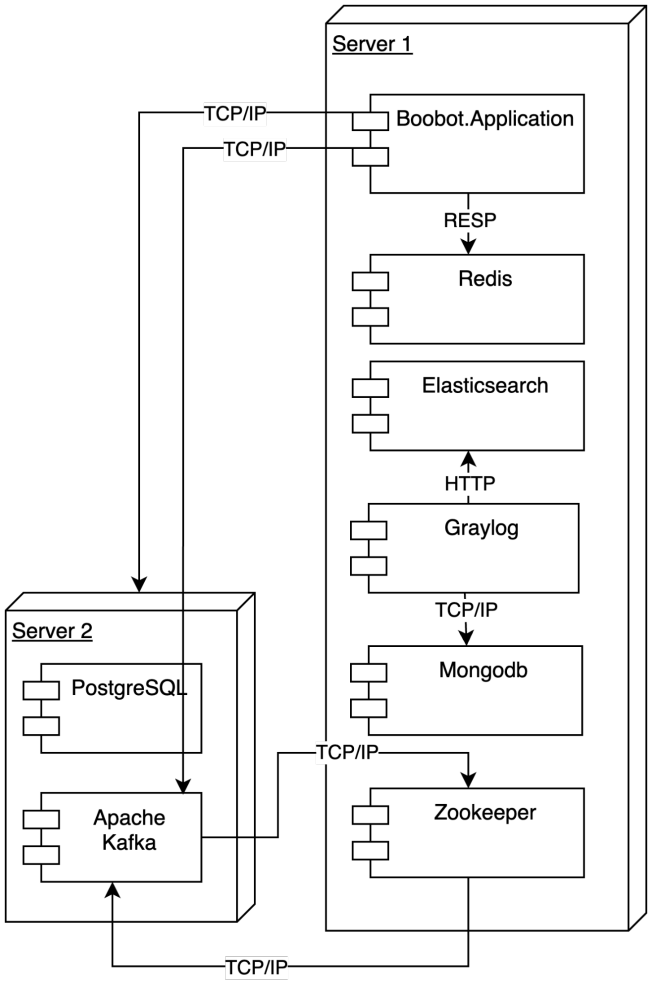


Рисунок 3.10 - Бюджетна схема розгортання для мінімізування витрат на обслуговування серверів

3.8. Frontend маршрути

/swagger/index.html - документація за допомогоюn технології Swagger

/login - сторінка авторизації

/commands - сторінка налаштування динамічних команд бота

3.9. Backend REST Api маршрути

Таблиця 3.2 – api маршрути

HTTP метод	**Маршрут**	**Призначення**
POST	api/v1/auth/get-token	Отримати токен
POST	api/v1/auth/refresh-token	Оновити токен
GET	api/v1/commands	Отримати всі налаштовані динамічні команди бота
POST	api/v1/commands	Створити нову динамічну команду бота
PUT	api/v1/commands	Оновити існуючу динамічну команду бота
DELETE	api/v1/commands/{id}	Видалити існуючу динамічну команду бота
PUT	{cardId}/switch?enabled={bool}	Включити або виключити (залежно від параметра enabled) інуючу динамічну команду бота

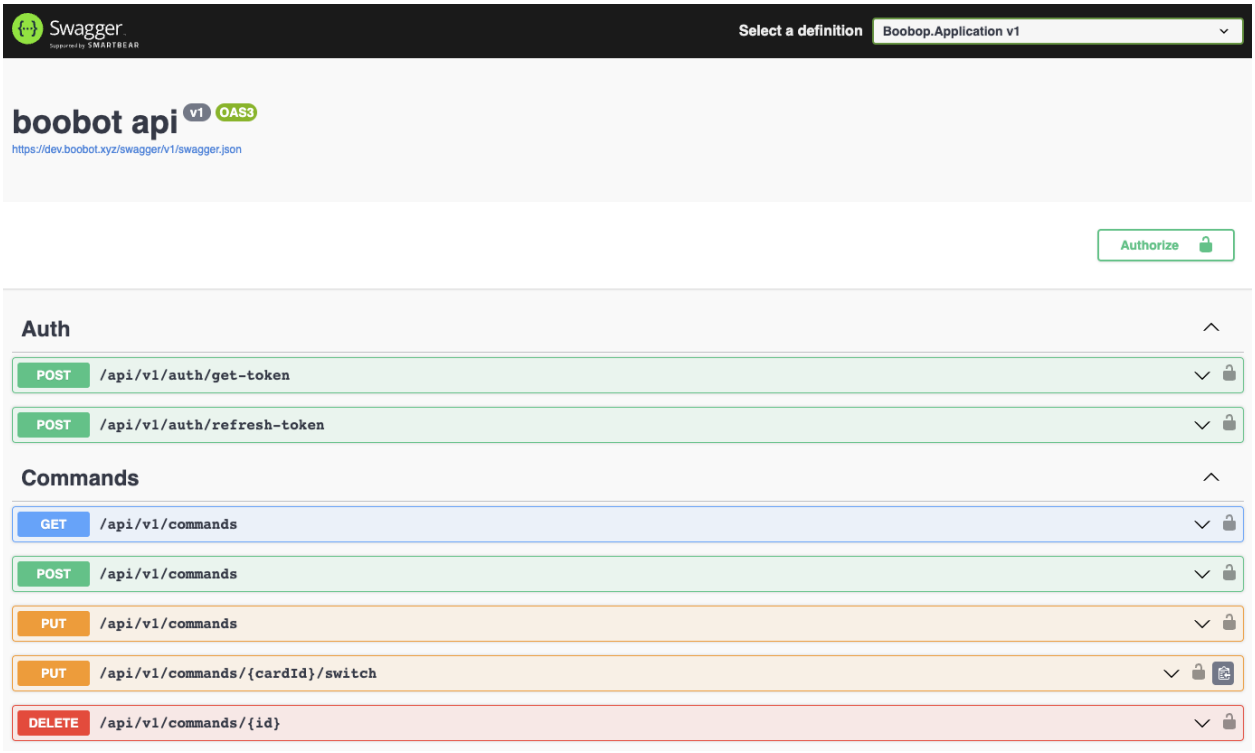


Рисунок 3.11 документація boobot api в Swagger

3.10. Тестування програмного забезпечення

Тестування програмного забезпечення є важливим етапом у процесі розробки. Воно дозволяє виявляти та усувати помилки, перевіряти відповідність програми вимогам, а також покращувати якість та надійність продукту ще на етапі розробки, запобігаючи деяких проблем при розвертанні і користуванні системою користувачами.

Юніт-тестування - це процес перевірки найменших одиниць програмного забезпечення, зазвичай окремих функцій або методів. Це дозволяє розробникам автоматично тестувати коректність роботи окремих частин програми, що не потребує великих часових витрат. Юніт тести можна запускати при кожному оновленні системи, або навіть в процесі розробки щоб відстежувати чи не зламали нові зміни існуючи код. Крім того ідеєю підходу TDD (Test-drive design) є розробка через тестування. Тобто спочатку проектується абстрактний контур системи (класи,

імена методів тощо), створюються юніт тести і далі пишеться реалізація системи до тих пір поки всі юніт тести не будут пройдені.

Переваги юніт-тестування:

- Раннє виявлення помилок. Помилки, знайдені на ранніх стадіях розробки, зазвичай легше і дешевше виправляти.
- Покращення дизайну. Процес написання юніт-тестів може допомогти розробникам краще зрозуміти вимоги до функцій та їх взаємодію.
- Спрощення модифікації коду. З юніт-тестами розробники можуть змінювати код, впевнені, що вони не зламають вже існуючу функціональність.

Недоліки юніт-тестування:

- Не може виявити інтеграційні помилки або проблеми з взаємодією компонентів.
- Написання юніт-тестів може бути трудомістким та являється доволі рутинною та одноманітною задачею.

Проєкт на 42% покритий юніт тестами. Цього недостатньо щоб повністю полягатися на автоматичне тестування і бути впевненим у відсутності помилок, але 100% теж не гарантують ідеально робочої системи через можливість впущення деяких особливих випадків тестування, або помилок в самих тестах.

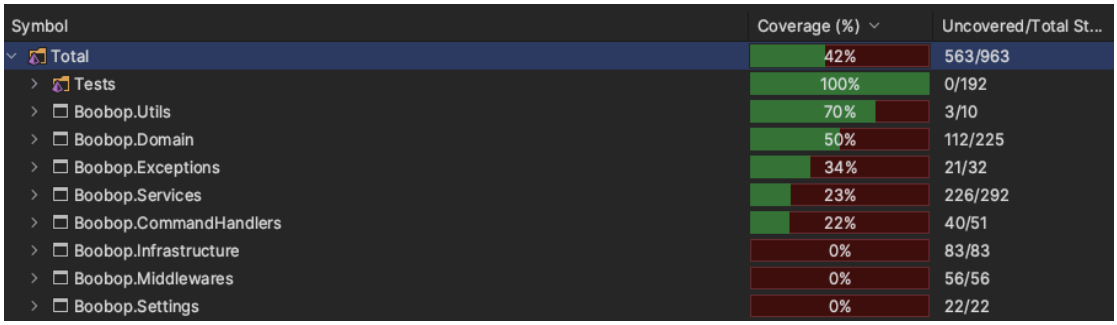


Рисунок 3.12 Покриття проєкту юніт-тестами

```
3 tests OK * Mykhailo Dovhalov More...
public CommandHandlingServiceTests()
{
    _commandsControlRepository = Substitute.For<ICommandsControlRepository>();
    _dynamicHandlerRunner = Substitute.For<IDynamicHandlerRunner>();
    _defaultCommandHandler = Substitute.For<ICommandHandler>();
    _defaultCommandHandler.IsResponsible(text: Arg.Any<string>()).Returns(returnThis: true);
    _defaultCommandHandler.HandleRequestAsync(message: Arg.Any<Message>()) // Task<HandlerResult>
        .Returns(returnThis: Task.FromResult((HandlerResult) new IgnoredHandlerResult()));
    _defaultHandlers = new List<ICommandHandler>();
    _bot = Substitute.For<ITBot>();

    _commandHandlingService = new CommandHandlingService(_commandsControlRepository, _dynamicHandlerRunner, _defaultHandlers, _bot);
}

[Fact]
1 test OK * Mykhailo Dovhalov
public async Task ProcessMessageAsync_MessageWithDefaultCommand_HandledByDefaultHandler()
{
    // Arrange
    var message = GetTestCommandMessage();
    _commandsControlRepository.GetByPathAsync(Arg.Any<string>())!.Returns(returnThis: Task.FromResult((Command)null!));
    _defaultHandlers.Add(_defaultCommandHandler);

    // Act
    await _commandHandlingService.ProcessMessageAsync(message);

    // Assert
    await _dynamicHandlerRunner.DidNotReceive().ExecuteCodeAsync(Arg.Any<string>(), message: Arg.Any<Message>());
    await _defaultCommandHandler.Received(1).HandleRequestAsync(message: Arg.Any<Message>());
}

[Fact]
Mykhailo Dovhalov
public async Task ProcessMessageAsync_MessageWithoutText_MessageNotHandled()
{
    // Arrange
    _defaultCommandHandler.HandleRequestAsync(message: Arg.Any<Message>()) // Task<HandlerResult>
        .Returns(returnThis: Task.FromResult((HandlerResult)new IgnoredHandlerResult()));
    var message = GetTestCommandMessage(string.Empty, inputText: string.Empty);
    _commandsControlRepository.GetByPathAsync(Arg.Any<string>())!.Returns(returnThis: Task.FromResult((Command)null!));
    _defaultHandlers.Add(_defaultCommandHandler);

    // Act
    await _commandHandlingService.ProcessMessageAsync(message);

    // Assert
    await _dynamicHandlerRunner.DidNotReceive().ExecuteCodeAsync(Arg.Any<string>(), message: Arg.Any<Message>());
    await _defaultCommandHandler.DidNotReceive().HandleRequestAsync(message: Arg.Any<Message>());
}
```

Рисунок 3.13 Приклад коду юніт-тестів з використанням фреймворку xunit та бібліотеки Nsubstitute

Ручне тестування - це процес перевірки програмного забезпечення та його функцій вручну. Це може включати перевірку відповідності інтерфейсу вимогам, тестування функціональності, тестування надійності та відмовостійкість, тестування безпеки і т.п.

Переваги ручного тестування:

- Гнучкість. Ручне тестування дозволяє легко адаптуватися до змін вимог або специфікацій.
- Можливість тестування в реальних умовах. Ручне тестування дозволяє перевіряти програму в умовах, які найближчі до реального використання.

Недоліки ручного тестування:

- Потребує більше часу і ресурсів, ніж автоматизоване тестування.
- Вірогідність помилок через людський фактор. Тестувальник може пропустити деякі помилки через неуважність або некомпетентність.
- В кінцевому рахунку, комбінація юніт-тестування та ручного тестування може допомогти досягти більшої надійності та якості програмного забезпечення.

Всі функції системи були ретельно протестовані вручну, виявлені дефекти були виправлені та повторно протестовані.

3.11. Функціонал сторінки аутентифікації

Аутентифікуватися може лише адміністратор з іменем користувача та паролем вказаними в конфігурації системи (значення за замовчуванням “*admin*” та “*password*” відповідно, але рекомендується змінити значення при першому розгортанні бота). При невдалій спробі увійти у систему поля вводу даних підсвічуються червоним щоб адміністратор мав змогу зрозуміти що причина в невалідних даних. Після вдалої авторизації користувач потрапляє на сторінку налаштування динамічних команд.

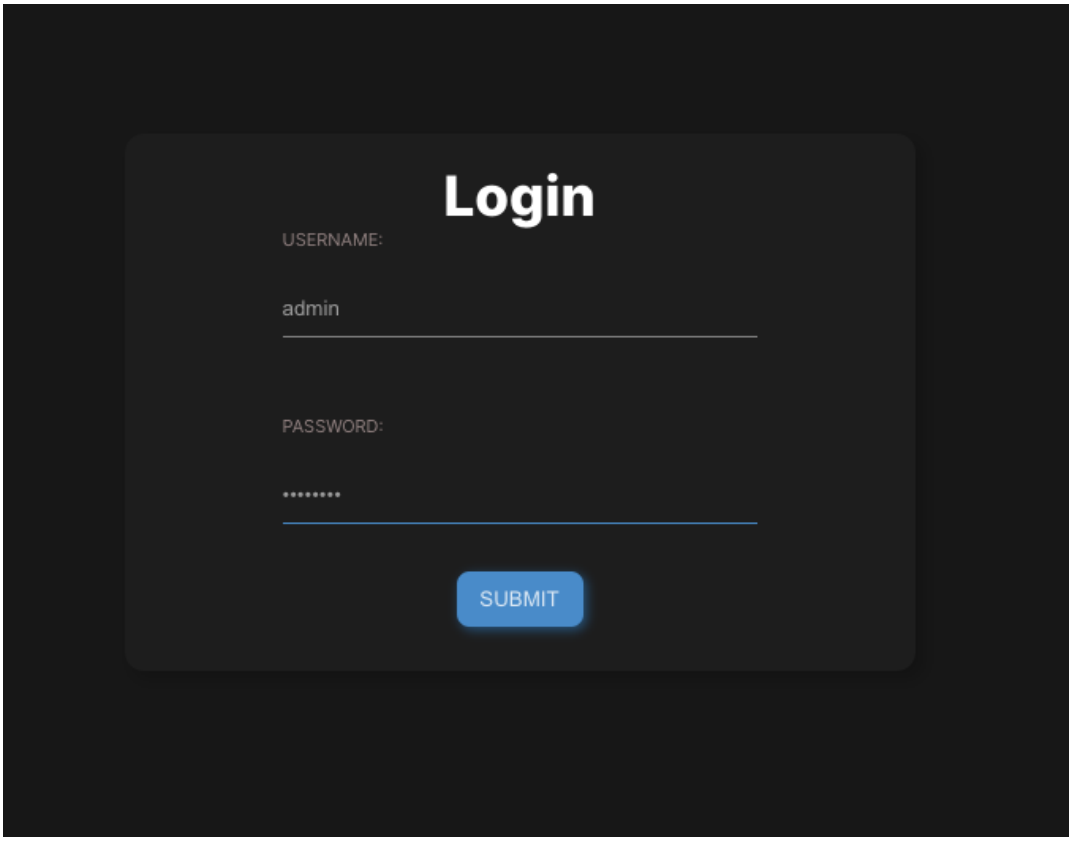


Рисунок 3.14 Сторінка аутентифікації до адмін-панелі

3.12. Функціонал сторінки конфігурації динамічних команд

На сторінці конфігурації користувач бачить усі налаштовані команди у вигляді карток та кнопку додавання нової команди. Кожна картка відповідає за певну команду.

Картка містить такі дані як: назва команди для ідентифікування, аліас команди (ім’я за яким можна визвати команду), статистика скільки раз командою скористувалися користувачі. Також користувач може увімкнути або вимкнути команду за допомогою перемикача на картці.

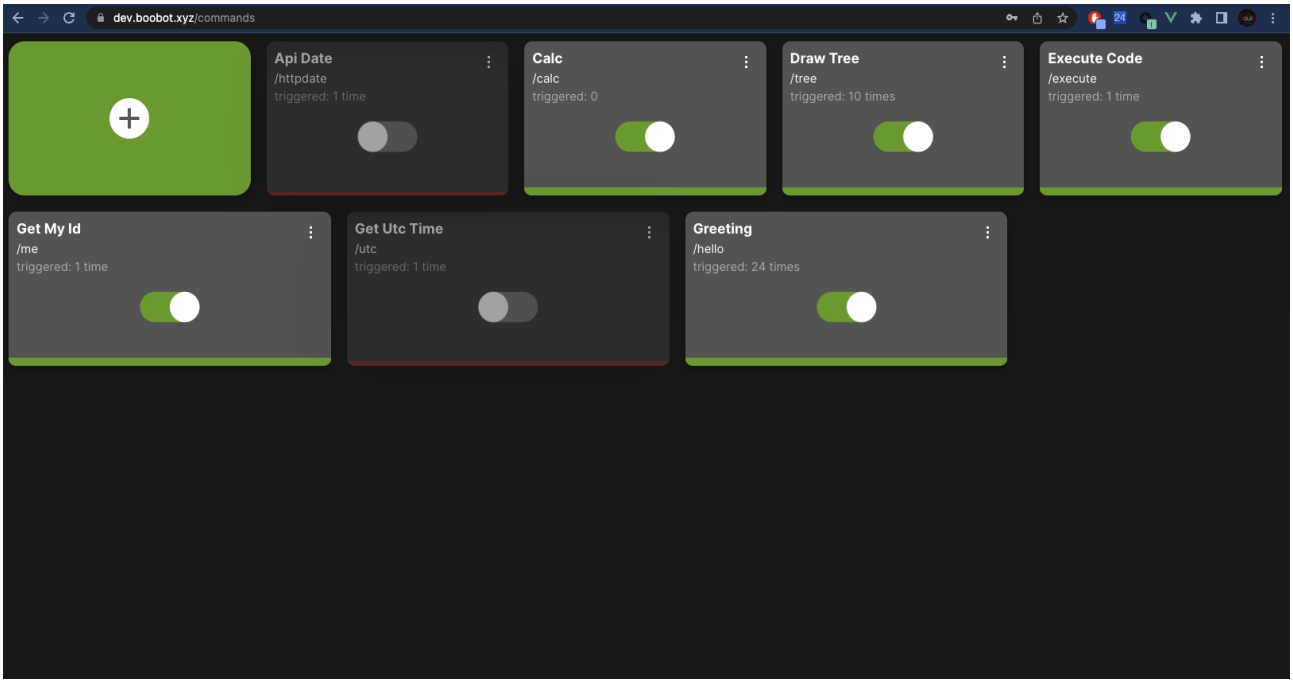


Рисунок 3.15 Конфагурація динамічних команд

Кожна картка має кнопку випадаючого меню яке дає змогу видалити команду або редагувати її. При видаленні команди вона зникає і бот більше не опрацьовує її.

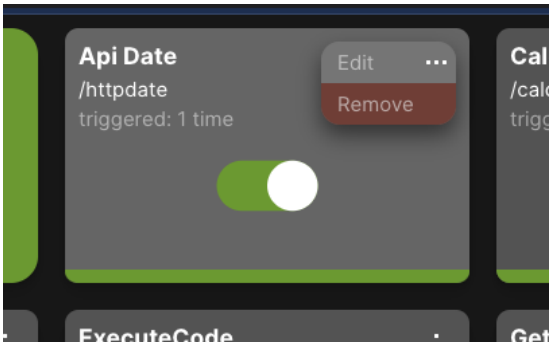


Рисунок 3.16 Випадаюче меню для редагування та видалення команди

При додаванні нової команди або редагуванні існуючої відкривається pop-up вікно з полями налаштування команди. Серед полів для редагування доступні поле назви, поле аліасу та, власне, виконуваний код, який буде запускатися при реагуванні на команду.

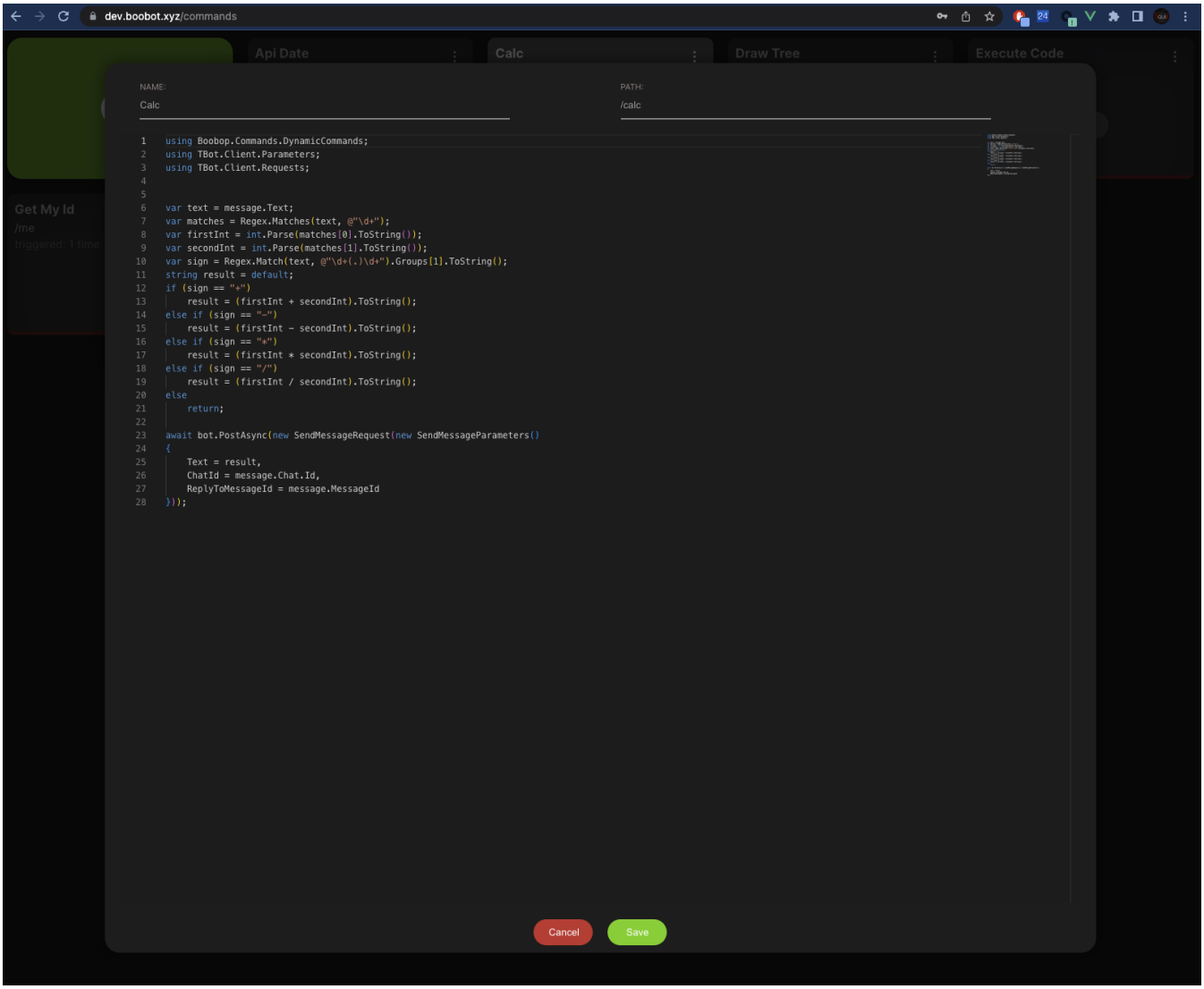


Рисунок 3.17 Pop-up вікно налаштування команди

При редагуванні або написанні коду доступні підказки та розмальовка коду згідно контексту. Поля імені команди та аліасу валідуються і не можуть бути пустими. При спробі збереження або оновлення команди з невалідними значеннями цих полів вони підсвічуються червоним кольором та миготять щоб адміністратор мав змогу зрозуміти що помилка в невалідних полях. При спробі зберегти код з помилками компіляції адміністратор побачить alert повідомлення браузера з описом помилок які сталися при спробі компіляції і команда не буде збереженою щоб зазделегідь запобігти помилок в коді ще на етапі конфігурації бота.

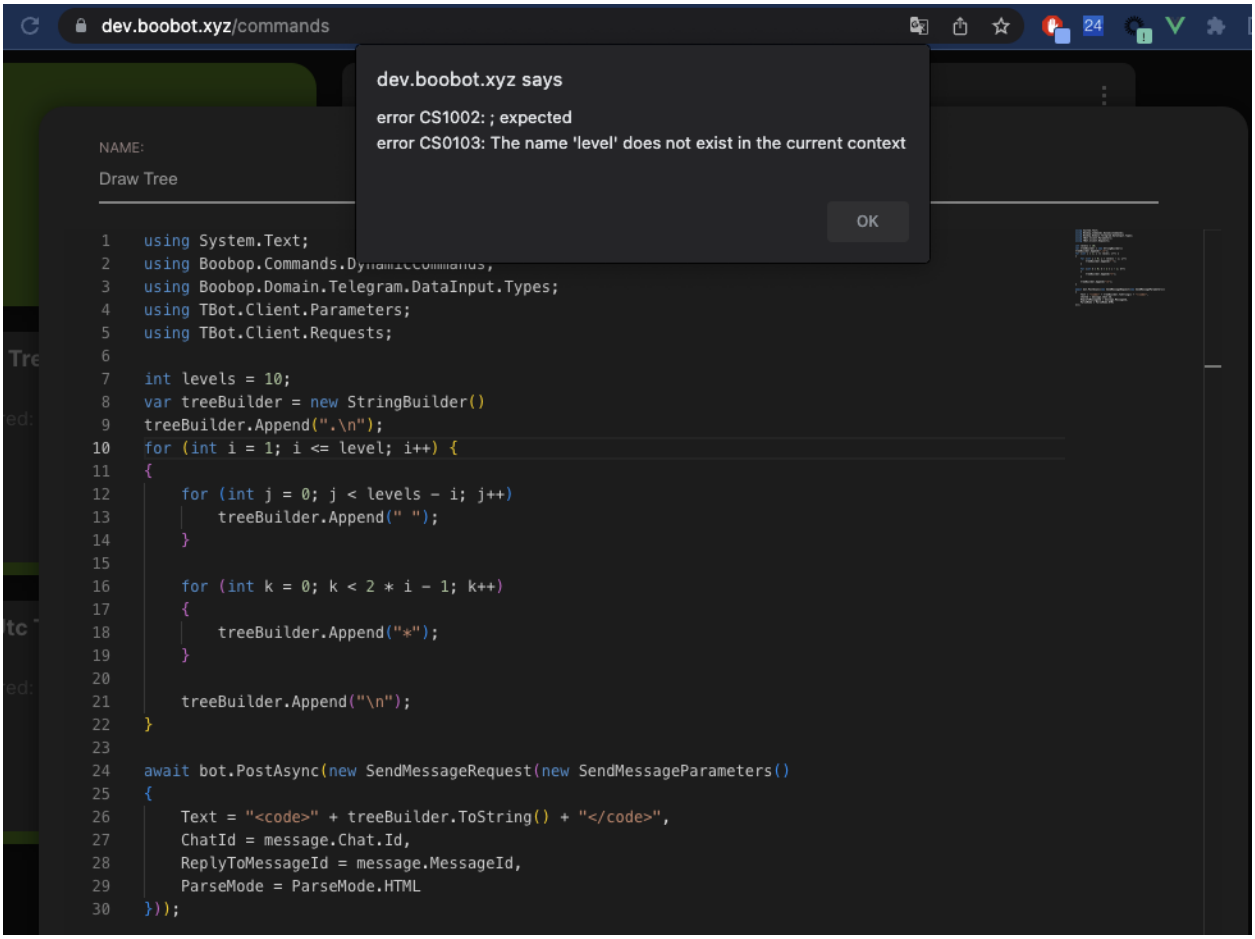


Рисунок 3.18 Відображення помилок компіляції при невдалій спробі збереження

В меню є 2 кнопки: відміна та збереження, функції яких відповідають їх назвам. Також можлива відміна натисканням кнопки “*escape*” та збереження натисканням кнопки “*enter*”.

3.13. Написання коду системи

Бот підтримує динамічні команди з кодом написаним на мові програмування C#.

Код пишеться без об’яви методів, неймспейсів чи класів. Спершу зверху є можливість вказати імена модулів для зручності їх використання, а далі код обробки самої команди.

В контексті коду доступні такі властивості:

- **ServiceProvider** - провайдер сервісів зареєстрованих через менханізм ін’єкції залежностей;
- **logger** - ILogger<T> для логування процесу виконання коду;
- **message** - дані повідомлення користувача, яка опрацьовується даним кодом;
- **bot** - клієнт бота для відправки повідомлень.

Для спрощення написання коду для бота існує nuget пакет (модуль на платформі .net з відкритим доступом) ***Boobop.Sdk*.** Завантаживши його з репозиторія nuget.org в повноцінній IDE можна швидше та простіше розробляти код використовуючи продвинути інструменти сучасних серед розробки, після чого скопіювати написаний код в команд на сторінці конфігурування та зберегти її.

NAME: Get Time in City

PATH: /time

```
using System.Globalization;
using System.Text.Json;
using Boobop.Commands.DynamicCommands;
using Newtonsoft.Json.Linq;

logger.LogInformation("The date fetch command was triggered!");
HttpClient client = new HttpClient();
var city = message.GetInputText();
var request = new HttpRequestMessage(HttpMethod.Get, $"http://worldtimeapi.org/api/timezone/Europe/{city}");
var response = await client.SendAsync(request);
JObject responseObject = JObject.Parse(await response.Content.ReadAsStringAsync());
var region = responseObject["timezone"]!.ToString();
var utcOffset = DateTimeOffset.ParseExact(responseObject["utc_offset"]!.ToString(), "zzz",
    CultureInfo.InvariantCulture,
    DateTimeStyles.None);
await SendMessageAsync(message.FromUser?.Username + $", current date in {region}"
 $" is: {DateTime.UtcNow.AddHours(utcOffset.Offset.TotalHours)}");
logger.LogInformation("Utc command handling was successfully finished!");
```

Рисунок 3.19 Приклад коду команди що отримує UTC дату зі стороннього API та відправляє її користувачу

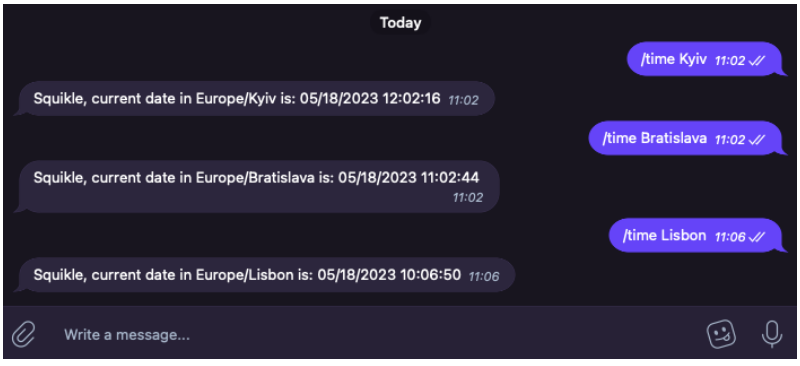


Рисунок 3.20 Результат виконання команди для отримання дати [рис. 3.16] для різних міст

ВИСНОВКИ

У результаті виконання дипломної роботи розроблено telegram чат-бота з можливістю динамічної конфігурації. Задача полягала в спрощенні процесу створення чат-ботів для оптимізації процесів комунікації малого та середнього бізнесу.

1. Проведено аналіз існуючих аналогічних систем, виявлено їх переваги та недоліки.
2. Базуючись на аналізі переваг та недоліків проаналізованих аналогів розроблено функціональні та нефункціональні вимоги до розробки системи.
3. Досліджено програмні засоби та технології, які можуть бути використані при розробці системі відповідно до поставленої задачі.
4. Розроблено програмне забезпечення для спрощення створення та конфігурування власного чат-бота згідно з поставленою задачею.
5. Проведено тестування розробленої системи.
6. Робота подана на апробацію: II Всеукраїнської наукової конференції студентів та молодих вчених "Наукові досягнення та відкриття сучасної молоді" за темою «Месенджери як основний засіб спілкування в сучасному суспільстві»; II Всеукраїнської науково-практичної інтернет-конференції молодих учених, аспірантів, студентів, учнів "Перший крок у науку: Конотопські наукові студії – 2023" за темою «Redis - швидка nosql база даних в оперативній пам’яті з даними ключ-значення»
7. Розроблена система оптимізує процес комунікації малого та середнього бізнеса з користувачами за допомогою месенджера telegram використовуючи функціонал чат-ботів.

СПИСОК ЛІТЕРАТУРИ

1. McTear, M., Callejas, Z., & Griol, D. (2016). The Conversational Interface: Talking to Smart Devices. Springer International Publishing. https://doi.org/10.1007/978-3-319-32967-3
2. Shawar, B. A., & Atwell, E. (2007). Chatbots: Are they Really Useful? LDV Forum - Linguistic Databases and Machine Translation, 22(1), 29-49. https://www.researchgate.net/publication/228566184_Chatbots_Are_they_Really_Useful
3. Dialogflow Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/dialogflow/docs/
4. ManyChat Help Center [Електронний ресурс] – Режим доступу до ресурсу: https://support.manychat.com/
5. Chatfuel Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.chatfuel.com/
6. Microsoft Bot Framework Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/azure/bot-service/
7. Amazon Lex Developer Guide [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/lex/latest/dg/what-is.html
8. Chatbot News Daily [Електронний ресурс] – Режим доступу до ресурсу: https://chatbotsmagazine.com/
9. Node.js vs .NET Core the winner [електронний ресурс] – Режим доступу до ресурсу: https://levelup.gitconnected.com/node-js-vs-net-core-the-winner-5ba06efb4c35
10. Benchmarking Apache Kafka, Apache Pulsar, and RabbitMQ: Which is the Fastest? [Електронний ресурс] – Режим доступу до ресурсу: https://www.confluent.io/blog/kafka-fastest-messaging-system/

11. Docker docs [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/
12. A DRY KISS and a SOLID base with YAGNI [Електронний ресурс] – Режим доступу до ресурсу: https://ranjula29.medium.com/a-dry-kiss-and-a-solid-base-with-yagni-a106bfade582
13. Clean Architecture & DDD, a mixed approach [Електронний ресурс] – Режим доступу до ресурсу: https://mastanca.medium.com/clean-architecture-ddd-a-mixed-approach-773ab4623e14

ДОДАТОК А

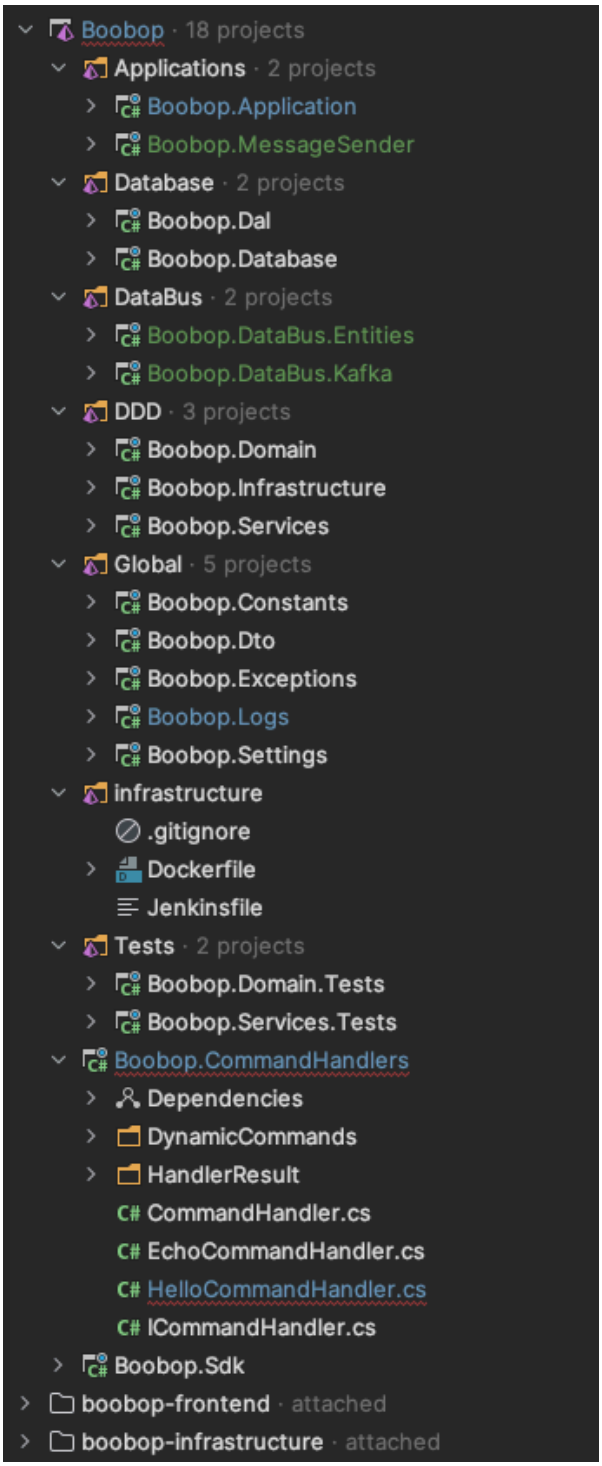
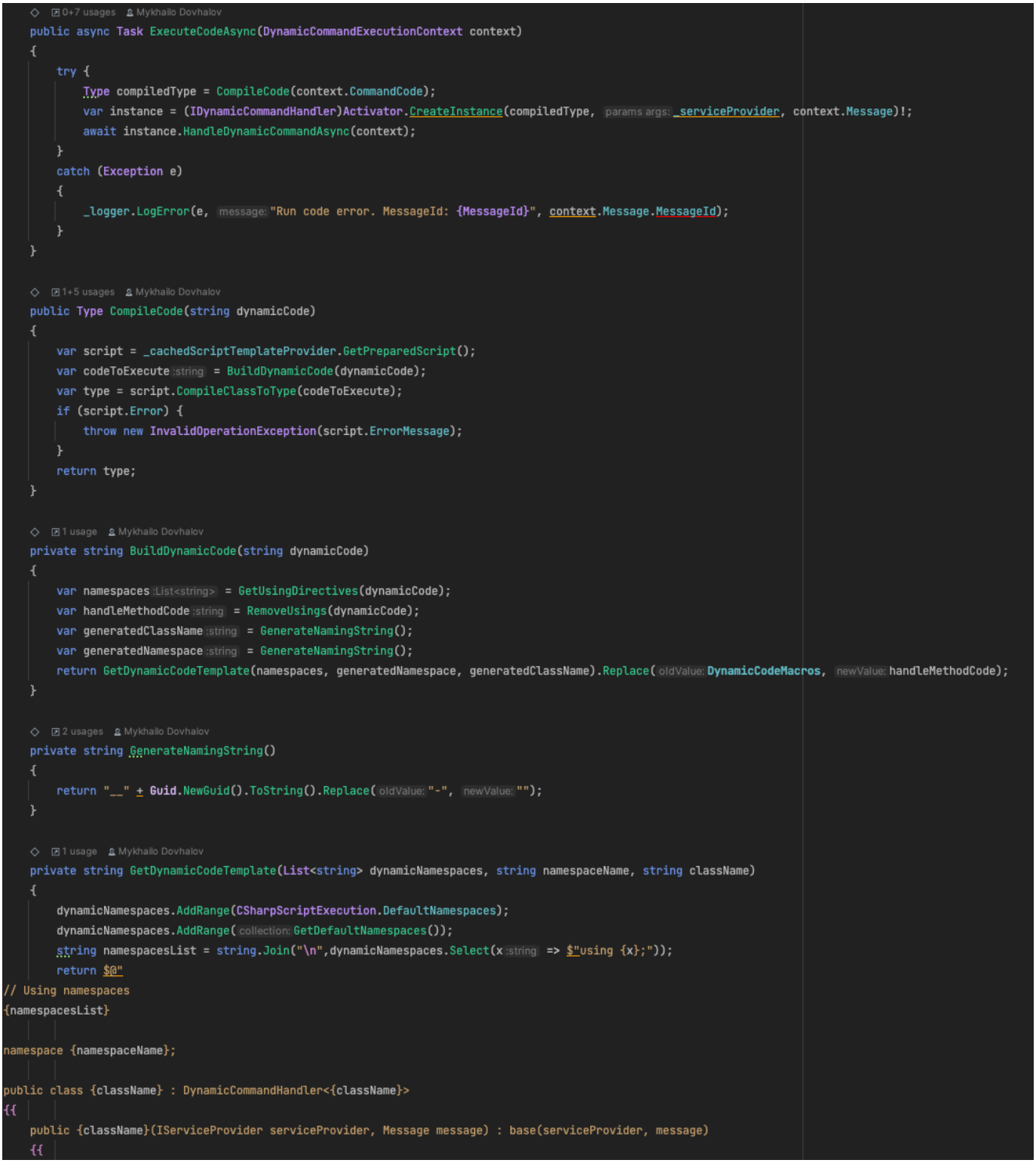


Рисунок 1. Структура проєкту

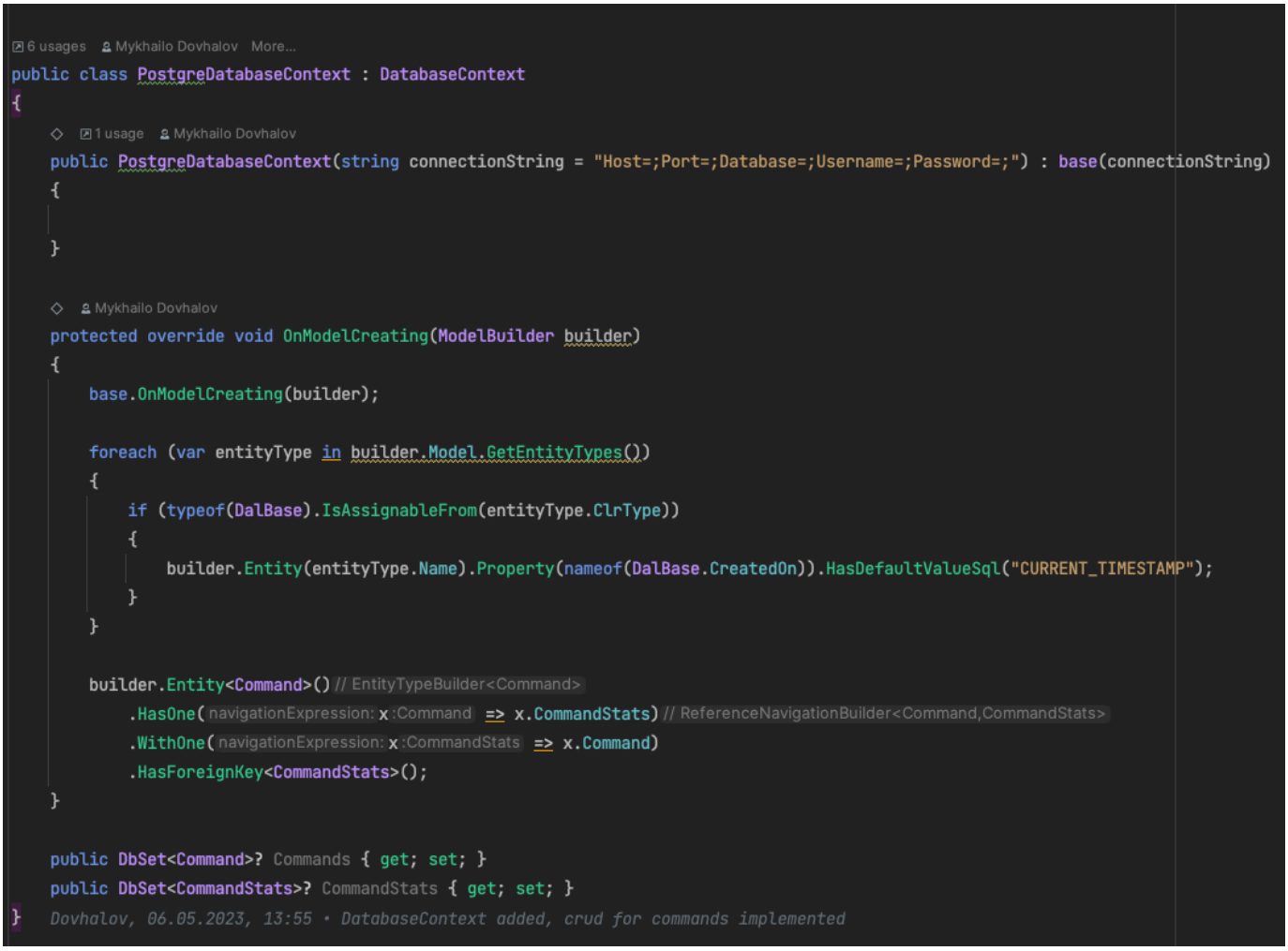


```
    ◇ 0+7 usages  Mykhailo Dovhalov
    public async Task ExecuteCodeAsync(DynamicCommandExecutionContext context)
    {
        try {
            Type compiledType = CompileCode(context.CommandCode);
            var instance = (IDynamicCommandHandler)Activator.CreateInstance(compiledType, params args: _serviceProvider, context.Message)!;
            await instance.HandleDynamicCommandAsync(context);
        }
        catch (Exception e)
        {
            _logger.LogError(e, message: "Run code error. MessageId: {MessageId}", context.Message.MessageId);
        }
    }

    ◇ 1+5 usages  Mykhailo Dovhalov
    public Type CompileCode(string dynamicCode)
    {
        var script = _cachedScriptTemplateProvider.GetPreparedScript();
        var codeToExecute :string = BuildDynamicCode(dynamicCode);
        var type = script.CompileClassToType(codeToExecute);
        if (script.Error) {
            throw new InvalidOperationException(script.ErrorMessage);
        }
        return type;
    }

    ◇ 1 usage  Mykhailo Dovhalov
    private string BuildDynamicCode(string dynamicCode)
    {
        var namespaces :List<string> = GetUsingDirectives(dynamicCode);
        var handleMethodCode :string = RemoveUsings(dynamicCode);
        var generatedClassName :string = GenerateNamingString();
        var generatedNamespace :string = GenerateNamingString();
        return GetDynamicCodeTemplate(namespaces, generatedNamespace, generatedClassName).Replace( oldValue: DynamicCodeMacros, newValue: handleMethodCode);
    }

    ◇ 2 usages  Mykhailo Dovhalov
    private string GenerateNamingString()
    {
        return "__" + Guid.NewGuid().ToString().Replace( oldValue: "-", newValue: "");
    }

    ◇ 1 usage  Mykhailo Dovhalov
    private string GetDynamicCodeTemplate(List<string> dynamicNamespaces, string namespaceName, string className)
    {
        dynamicNamespaces.AddRange(CSharpScriptExecution.DefaultNamespaces);
        dynamicNamespaces.AddRange( collection: GetDefaultNamespaces());
        string namespacesList = string.Join("\n",dynamicNamespaces.Select(x :string => $"using {x};"));
        return $@"
// Using namespaces
{namespacesList}

namespace {namespaceName};

public class {className} : DynamicCommandHandler<{className}>
{{
    public {className}(IServiceProvider serviceProvider, Message message) : base(serviceProvider, message)
    {{
```

Рисунок 2. Клас що динамічно компілює налаштовні команди CSharpDynamicHandlerRunner

```
var builder = WebApplication.CreateBuilder();
builder.ConfigureOptions();
builder.ConfigureKafka();
builder.Services.AddKafkaDataBus();
builder.Logging.AddLogger(builder.Configuration);
builder.Services.ConfigureServices();   Dovhalov, 06.05.2023, 10:56 • Boobop added project structure, implemented authorization
builder.Services.AddControllers().SetCamelCaseResponse();

builder.AddTBotRedisLimiter();
builder.AddTelegramTBotConfigure();

builder.Services.AddBotCommandsProcessing();

builder.Services.AddSwagger();
builder.Services.AddBoobotAuthentication();
builder.Services.AddCors(options => options.AddDefaultPolicy(x :CorsPolicyBuilder =>
    {
        x.AllowAnyHeader()
            .AllowAnyMethod()
            .SetIsOriginAllowed(_ => true)
            .AllowCredentials();
    }));
builder.Services.AddTransient<IEnumerable<ICommandHandler>>(sp :IServiceProvider =>
    new List<ICommandHandler>()
    {
        new EchoCommandHandler(),
        new HelloCommandHandler( kafkaContext: sp.GetRequiredService<KafkaContext>())
    });

var app :WebApplication = builder.Build();
var databaseInit :Task = app.InitDatabaseAsync();

app.UseStaticFiles();

app.UseCors();
app.UseRouting();

app.UseMiddleware<ExceptionResponseMiddleware>();
app.UseAuthorization();
app.UseSwagger();
app.UseSwaggerUI();
app.MapControllers();

app.MapGet( pattern: "api/health",  requestDelegate: context => context.Response.WriteAsync( text: "Alive!"));
app.UseTBot( handler: async (context, update) =>
{
    context.RequestServices.GetRequiredService<ILogger<Program>>().LogInformation( message: "Got a message! {Message}", update.Message?.Text ?? "*Empty message*");
    if (update.Message != null) {
        var handler = context.RequestServices.GetRequiredService<ICommandHandlingService>();
        await handler.ProcessMessageAsync(update.Message.ToDomain());
    }
});

await databaseInit;
await app.SetWebhookAsync();
await app.NotifyAdminAsync( message: "The bot is working! Hello!");
app.Run();
```

Рисунок 3. Клас Program що відповідає за конфігурацію чат-бота



```
6 usages  Mykhailo Dovhalov  More...
public class PostgreDatabaseContext : DatabaseContext
{
    1 usage  Mykhailo Dovhalov
    public PostgreDatabaseContext(string connectionString = "Host=;Port=;Database=;Username=;Password=;") : base(connectionString)
    {

    }

    Mykhailo Dovhalov
    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);

        foreach (var entityType in builder.Model.GetEntityTypes())
        {
            if (typeof(DalBase).IsAssignableFrom(entityType.ClrType))
            {
                builder.Entity(entityType.Name).Property(nameof(DalBase.CreatedOn)).HasDefaultValueSql("CURRENT_TIMESTAMP");
            }
        }

        builder.Entity<Command>() // EntityTypeBuilder<Command>
            .HasOne( navigationExpression: x :Command => x.CommandStats) // ReferenceNavigationBuilder<Command,CommandStats>
            .WithOne( navigationExpression: x :CommandStats => x.Command)
            .HasForeignKey<CommandStats>();
    }

    public DbSet<Command>? Commands { get; set; }
    public DbSet<CommandStats>? CommandStats { get; set; }
}   Dovhalov, 06.05.2023, 13:55 • DatabaseContext added, crud for commands implemented
```

Рисунок 4. Клас PostgreDatabaseContext, що відповідає за конфігурацію взаємодії з базою даних

NAME: Get Time in City

PATH: /time

```
using System.Globalization;
using System.Text.Json;
using Boobop.Commands.DynamicCommands;
using Newtonsoft.Json.Linq;

logger.LogInformation("The date fetch command was triggered!");
HttpClient client = new HttpClient();
var city = message.GetInputText();
var request = new HttpRequestMessage(HttpMethod.Get, $"http://worldtimeapi.org/api/timezone/Europe/{city}");
var response = await client.SendAsync(request);
JObject responseObject = JObject.Parse(await response.Content.ReadAsStringAsync());
var region = responseObject["timezone"]!.ToString();
var utcOffset = DateTimeOffset.ParseExact(responseObject["utc_offset"]!.ToString(), "zzz",
    CultureInfo.InvariantCulture,
    DateTimeStyles.None);
await SendMessageAsync(message.FromUser?.Username + $", current date in {region}"
 $" is: {DateTime.UtcNow.AddHours(utcOffset.Offset.TotalHours)}");
logger.LogInformation("Utc command handling was successfully finished!");
```

Рисунок 5. Клас EuropeCityTimeCommand, динамічно налаштованої команди що відправляє користувачу час у європейських містах

```
2 usages  Mykhailo Dovhalov
public class CommandsControlRepository : ICommandsControlRepository
{
    private readonly DatabaseContext _databaseContext;

    ◇  Mykhailo Dovhalov
    public CommandsControlRepository(DatabaseContext databaseContext)
    {
        _databaseContext = databaseContext;
    }

    ◇  0+1 usages  Mykhailo Dovhalov
    public async Task<List<Domain.Command.Command>> GetAllAsync()
    {
        var result:List<Command> = await _databaseContext// DatabaseContext
            .Set<Dal.Sql.Command>() // DbSet<Command>
            .Include(navigationPropertyPath: x:Command => x.CommandStats) // IIncludableQueryable<Command,CommandStats?>
            .OrderBy(x:Command => x.Title) // IOrderedQueryable<Command>
            .ToListAsync(); // Task<List<...>>
        return result.ToDomain();
    }

    ◇  0+1 usages  Mykhailo Dovhalov
    public async Task<List<Domain.Command.Command>> GetEnabledAsync()
    {
        var result:List<Command> = await _databaseContext// DatabaseContext
            .Set<Dal.Sql.Command>() // DbSet<Command>
            .Include(navigationPropertyPath: x:Command => x.CommandStats) // IIncludableQueryable<Command,CommandStats?>
            .Where(x:Command => x.IsEnabled) // IQueryable<Command>
            .ToListAsync(); // Task<List<...>>
        return result.ToDomain();
    }

    ◇  0+10 usages  Mykhailo Dovhalov
    public async Task<Domain.Command.Command?> GetByPathAsync(string path)
    {
        var result:Command? = await _databaseContext.Set<Dal.Sql.Command>() // DbSet<Command>
            .Include(navigationPropertyPath: x:Command => x.CommandStats) // IIncludableQueryable<Command,CommandStats?>
            .Where(x:Command => x.Path == path)
            .Where(x:Command => x.IsEnabled) // IQueryable<Command>
            .OrderBy(x:Command => x.Title) // IOrderedQueryable<Command>
            .FirstOrDefaultAsync(); // Task<Command?>
        return result?.ToDomain();
    }   Dovhalov, 14.05.2023, 17:24 • Default commands added, dynamically compilated commands implemented

    ◇  0+2 usages  Mykhailo Dovhalov
    public async Task<Guid> CreateAsync(Domain.Command.Command command)
    {
        var dalCommand = command.ToDal();
```

Рисунок 6. Код репозиторію для роботи з динамічно-налаштованими командами CommandsControlRepository

ДОДАТОК Б

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА TELEGRAM ЧАТ-БОТУ З ДИНАМІЧНОЮ КОНФІГУРАЦІЄЮ МОВАМИ C# ТА JAVASCRIPT

Виконав студент 4 курсу
групи ПД-41
Довгальов Михайло Валерійович
Керівник роботи
К.т.н, доц, доцент кафедри ІПЗ Негоденко Олена Василівна

Київ - 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Об'єкт дослідження - процес комунікації малого бізнесу з клієнтами.

Предмет дослідження – чат-бот з динамічним конфігуруванням.

Мета роботи - підвищення ефективності комунікації клієнтів за рахунок Telegram чат-боту з динамічною конфігурацією, створеного мовами C# та JavaScript.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати існуючі сучасні аналогічні програмні забезпечення та виявити їх переваги та недоліки. І в результаті побудувати порівняльну таблицю.
2. Розробити вимоги до системи що розробляється базуючись на аналізі переваг та недоліках проаналізованих аналогів.
3. Дослідити програмні засоби та технології, які можуть бути використані при розробці системі відповідно до поставленої задачі.
4. Розробити програмне забезпечення для спрощення створення та конфігурування власного чат-бота згідно з поставленою задачею.
5. Провести тестування розробленої системи.
6. Пройти апробацію на Науково-технічних конференціях;

3

ПОРІВНЯННЯ АНАЛОГІВ

Характеристика	Dialogflow	ManyChat	Chatfuel	Microsoft Bot Framework	Amazon Lex	Boobot
Можливість інтегрування власного коду	+	-	-	+	+	+
Можливість розвертання на своїх серверах	-	-	-	-	-	+
Налаштування без знання мов програмування	-	+	+	-	-	-
Інтеграція з месенджером telegram	+	+	+	+	+	+
Наявність штучного інтелекту для обробки запитів	+	+	+	+	+	-
Аналітика користування чат-ботом	-	+	+	-	-	+
Інтегрування з будь-якими інфраструктурними сервісами (типу S3, Google Cloud Storage) інших провайдерів	-	-	-	-	-	+

4

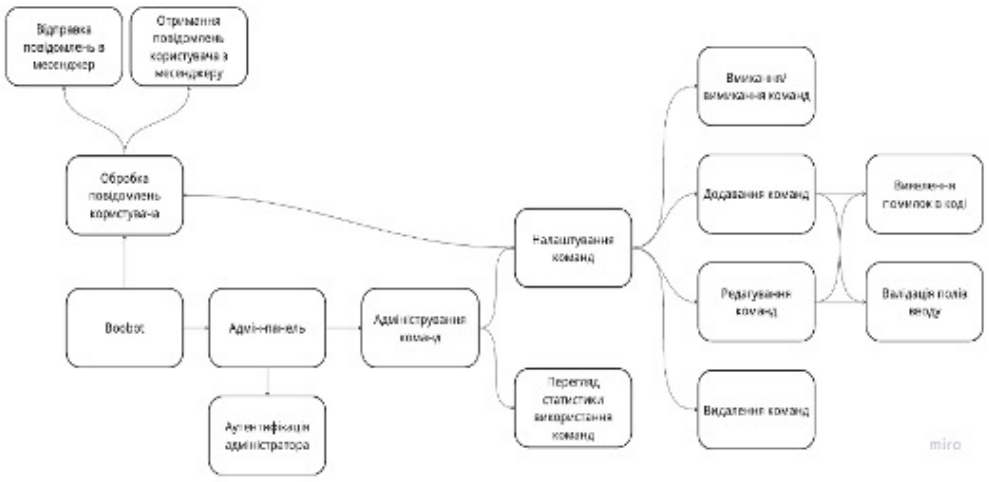
ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Основні функціональні вимоги:

1. Виконання динамічно створених команд.
2. Додавання, редагування та видалення нових команд.
3. Вимкнення та вимикання створених команд
4. Зручний редактор коду з автокоректуванням та валідацією.
5. Підрахунок статистики використання команди.

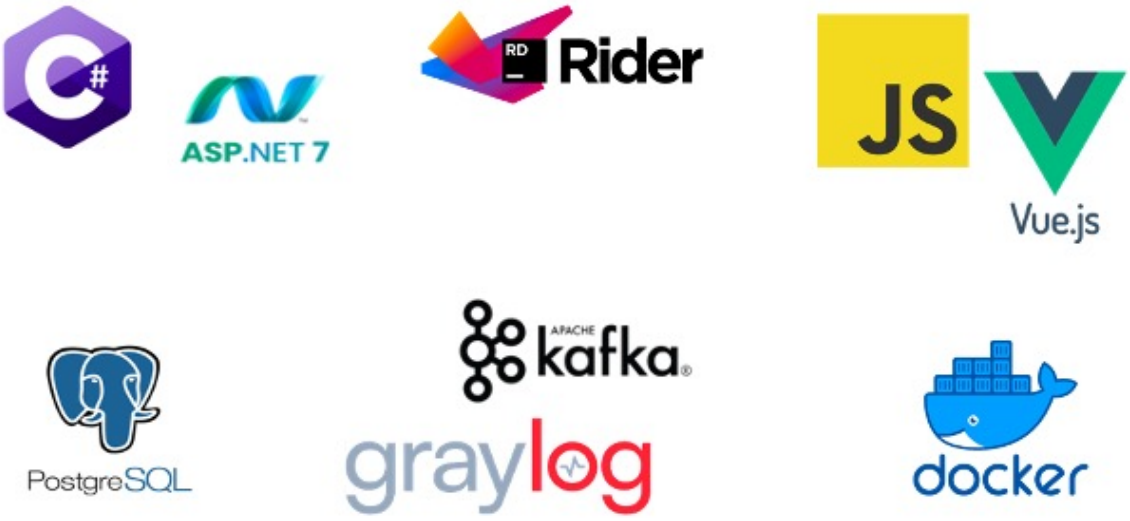
5

АСОЦІАТИВНА МАПА ФУНКЦІЙ ЧАТ-БОТУ



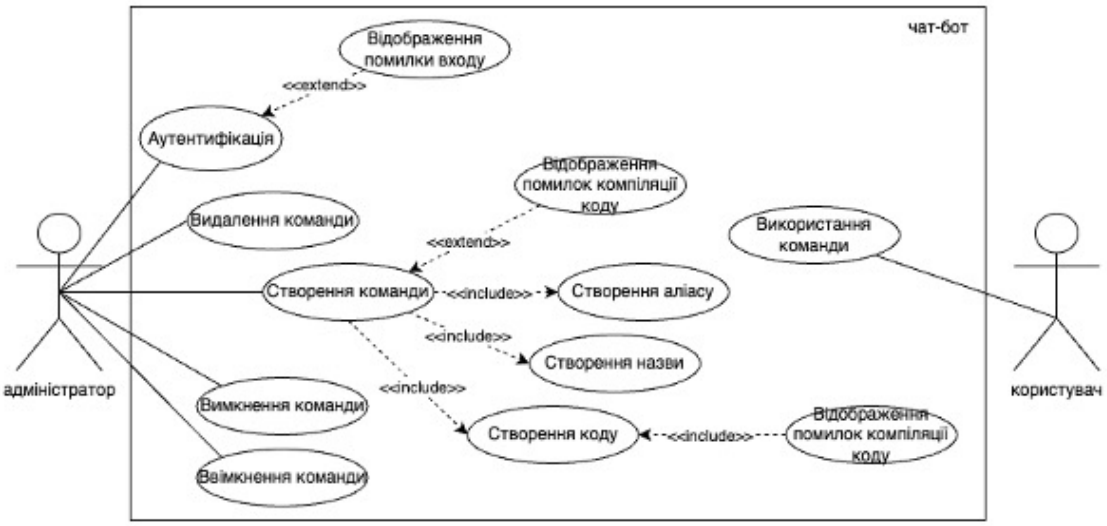
6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



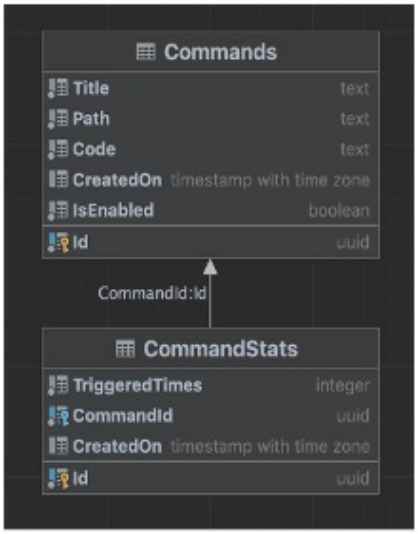
7

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



8

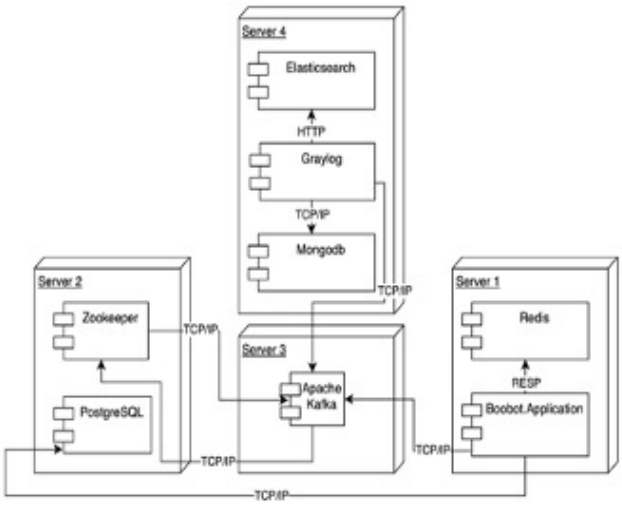
ДІАГРАМА ЗВ'ЯЗКІВ СУТНОСТЕЙ



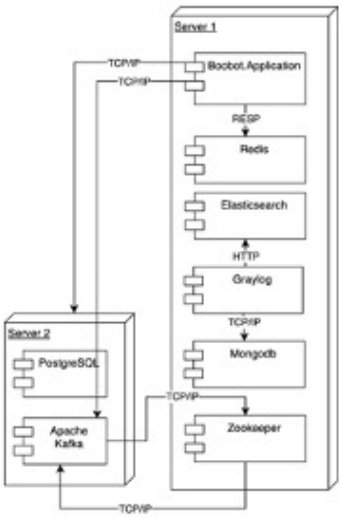
Commands і CommandStats з відношенням 1:1

9

СХЕМА РОЗГОРТАННЯ



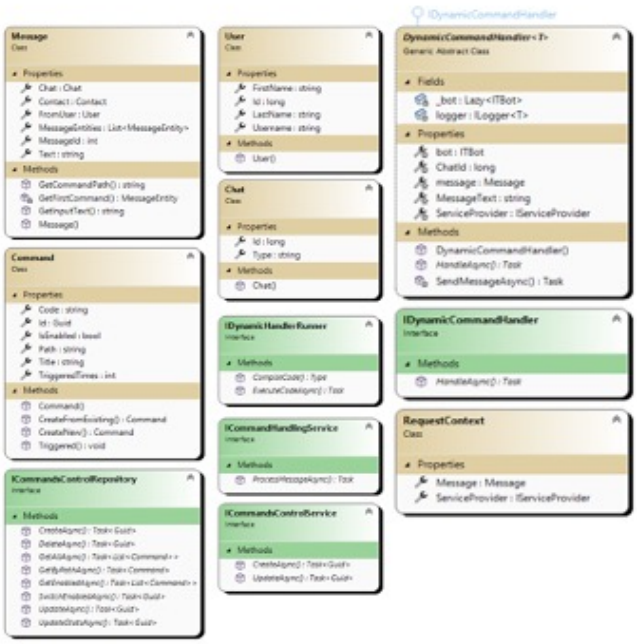
Рекомендована схема для забезпечення гнучкості розгорненної інфраструктури



Бюджетна схема для мінімізування витрат на обслуговування серверів

10

ДІАГРАМА КЛАСІВ



Boobot.Domain

11

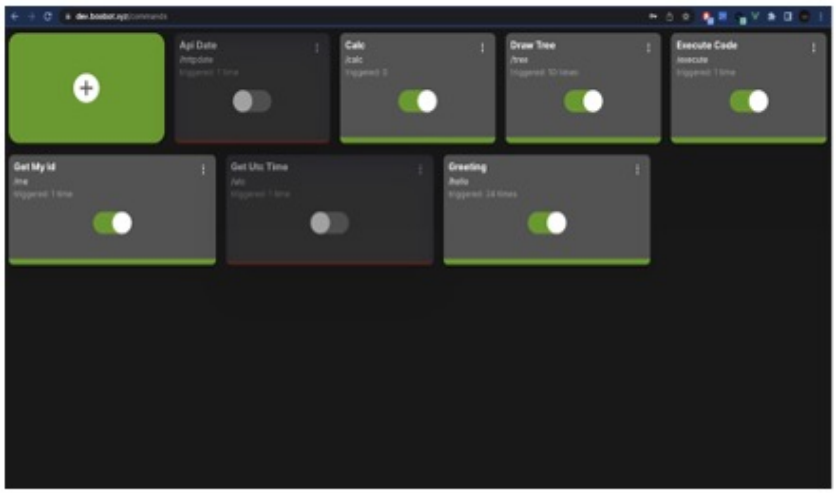
ДІАГРАМА КЛАСІВ

Boobot.Services

Boobot.Infrastructure

12

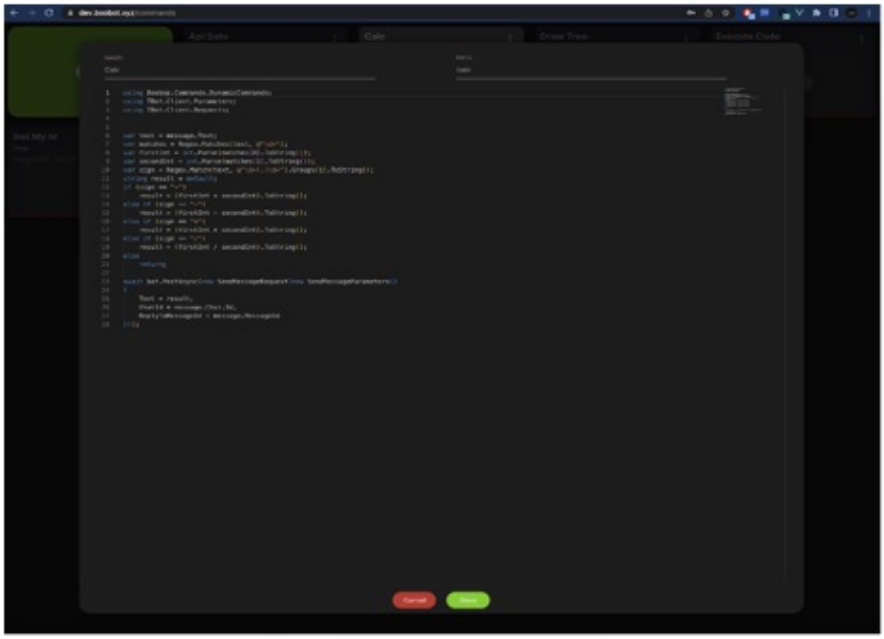
ЕКРАННІ ФОРМИ СИСТЕМИ



Сторінка конфігурації динамічних команд

13

ЕКРАННІ ФОРМИ СИСТЕМИ



Pop-up вікно налаштування команди

14

ЕКРАННІ ФОРМИ СИСТЕМИ

```
name:                                                    note:
Get Time in City                                         time
```

```
using System.Globalization;
using System.Text.Json;
using Bootsy.Commands.DynamicCommands;
using Newtonsoft.Json.Linq;

logger.LogInformation("The date fetch command was triggered!");
HttpClient client = new HttpClient();
var city = message.GetInputText();
var request = new HttpRequestMessage(HttpMethod.Get, $"http://worldtimeapi.org/api/timezone/Europe/{city}");
var response = await client.SendAsync(request);
JObject responseObject = JObject.Parse(await response.Content.ReadAsStringAsync());
var region = responseObject["timezone"].ToString();
var utcOffset = DateTimeOffset.ParseExact(responseObject["utc_offset"].ToString(), "zzz",
    CultureInfo.InvariantCulture,
    DateTimeStyles.None);
await SendMessageAsync($"{message.FromUser?.Username ?? ""}, current date in {region}"
+ $" is: {DateTime.UtcNow.AddHours(utcOffset.Offset.TotalHours)}");
logger.LogInformation("Utc command handling was successfully finished!");
```

Код команди що отримує дату зі стороннього API та відправляє її користувачу

Результат виконання команди для отримання дати та часу різних міст

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Довгальов М.В. Месенджери як основний засіб спілкування в сучасному суспільстві: Матеріали II Всеукраїнської наукової конференції студентів та молодих вчених "Наукові досягнення та відкриття сучасної молоді". Збірник тез.- м. Покровськ/м. Луцьк, Донецький національний технічний університет, 31 травня 2023
2. Довгальов М.В. Redis - швидка nosql база даних в оперативній пам’яті з даними ключ-значення: Матеріали II Всеукраїнської науково-практичної інтернет-конференції молодих учених, аспірантів, студентів, учнів "Перший крок у науку: Конотопські наукові студії – 2023". Збірник тез.-

ВИСНОВКИ

1. Проведено аналіз існуючих аналогічних систем, виявлено їх переваги та недоліки. Спільним недоліком всіх цих платформ окрім Bot Framework від Microsoft є неможливість або обмеження в написанні власного коду, що значно лімітує гнучкість налаштування боту
2. Базуючись на аналізі переваг та недоліків проаналізованих аналогів розроблено функціональні та нефункціональні вимоги до розробки системи.
3. Досліджено програмні засоби та технології, які можуть бути використані при розробці системі відповідно до поставленої задачі. Перевага була надана швидкодіючим та сучасним програмним засобами.
4. Розроблено програмне забезпечення для спрощення створення та конфігурування власного чат-бота згідно з поставленою задачею.
5. Проведено unit та ручне тестування розробленої системи.

17

ДЯКУЮ ЗА УВАГУ!

18