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

Архітектура програмного застосунку питально-відповідальної підсистеми

Текстова частина до курсової роботи за спеціальністю „Комп'ютерні науки"

Керівник курсової роботи
д.т.н., доц. Глибовець А.М.

_____
(підпис)
"____" _____ 2020 р.

Виконав студент
Андрощук М.В
"____" _____ 2020 р.

Київ 2020

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

ЗАТВЕРДЖУЮ
Зав. кафедри мережних технологій,
проф., д.ф.-м.н. Г. І. Малашонок
_____
(підпис)
„____” _____2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Андрощуку М. В. факультету інформатики 1 курсу МП
ТЕМА Архітектура програмного застосунку питально-відповідальної підсистеми

Зміст ТЧ до курсової роботи:
Індивідуальне завдання
Календарний план
Анотація
Вступ
РОЗДІЛ 1: Питально-відповідальні системи
РОЗДІЛ 2: Огляд сервісів для створення питально-відповідальних систем
РОЗДІЛ 3: Створення питально-відповідального застосунку
Висновки
Список використаної літератури
Додатки

Дата видачі „___” _____ 2019 р. Керівник _____
(підпис)
Завдання отримав _____
(підпис)

Тема: Архітектура програмного застосунку питально-відповідальної підсистеми

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання теми курсової роботи.	08.10.2019	
2.	Пошук тематичної літератури	26.11.2019	
3.	Ознайомлення з тематичними матеріалами та побудова структури практичної та теоретичної частин курсової роботи	5.12.2019	
4.	Написання вступу та змісту роботи	11.12.2019	
5.	Ознайомлення з питально-відповідальними системами та написання першого розділу	20.01.2020	
6.	Ознайомлення з сервісами для створення питально-відповідальних систем та написання другого розділу	5.02.2020	
7.	Створення питально-відповідального застосунку	21.02.2020	
7.	Тестування та перевірка створеного застосунку	27.02.2020	
8.	Написання третього розділу роботи	13.03.2020	
9.	Оформлення роботи відповідно до вимог написання курсової роботи.	26.03.2020	
10.	Створення презентації та написання доповіді для захисту курсової роботи.	07.042020	
11.	Узгодження попередньої версії роботи з керівником	16.04.2020	
12.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	20.04.2020	
13.	Захист курсової роботи	травень 2020	

Студент Андрощук М.В.

Керівник Глибовець А.М.

“_____”_____

Зміст

Анотація

В даній курсовій роботі розглянуто питально-відповідальні системи, сервіси для створення питально-відповідальних систем та процес створення питально-відповідальної системи.

Ключові слова: питально-відповідальні системи, Dialogflow, Microsoft Bot Service, Microsoft QnA Maker, LUIS, IBM Watson Assistant

Вступ

Актуальність. Питально-відповідальні системи є важливою частиною сьогодення. Вони відіграють важливу роль при взаємодії з людьми, дозволяючи автоматизувати частину взаємодії між людьми за допомогою систем. Так поширеними стали рішення у сфері клієнтської підтримки, рекомендацій та замовлень.

Однак сьогодні існує багато підходів до реалізації питально-відповідальних систем. Існують різні технології та сервіси для створення таких систем. Порівняння важливе для вибору рішення, що найкраще відповідатиме вимогам задач.

Мета дослідження. За мету даної роботи було взято огляд принципів створення питально-відповідальних систем, сервісів для їх створення та створення прототипу програмного застосунку питально-відповідальної підсистеми.

Об’єкт дослідження. Питально-відповідальні системи.

Предмет дослідження. Підходи до створення питально-відповідальних систем, сервіси для створення питально-відповідальних систем.

Джерела дослідження. Програмна документація, електронні ресурси.

Робота складається з трьох розділів.

Перший розділ присвячено аналізу питально-відповідальних систем, способів і принципів їх створення.

В другому розділі наведено огляд сервісів для розробки питально-відповідальних систем.

Третій розділ присвячено створенню власної питально-відповідальної системи.

Створено прототип програмного застосунку для питально-відповідальної системи для надання відповідей про розклад в НаУКМА.

Розділ 1. Питально-відповідальні системи

1.1 Що таке питально-відповідальна система

Питально-відповідальна система - система, що автоматично відповідає на питання, задані людьми природною мовою, використовуючи попередньо структуровану базу даних або колекцію документів природною мовою.[1]

Питально-відповідальні системи можна розглядати як розвиток інформаційного пошуку. До таких систем існує зацікавленість, їх кількість постійно зростає. [1] Оскільки, об’єм інформації, що доступний в мережі Інтернет, постійно зростає і необхідні інструменти пошуку для виокремлення корисної інформації.

Як правило такі системи використовуються як альтернатива збіркам питань, що часто задаються. Іншим варіантом використанням є клієнтська підтримка, коли користувач спілкується з системою, а не живою людиною.

Першими питально-відповідальними системами вважаються BASEBALL та LUNAR. BASEBALL відповідав на питання пов’язані з баскетбольною лігою США за рік, а LUNAR на питання про геологічний аналіз каміння, привезеного місією Apollo з місяця.[2]

1.2 Класифікація за доменом

За пов’язаністю із сферою застосування питально-відповідальні системи поділяються на системи з обмеженим та відкритим доменом. Визначення відбувається за тим чи питання користувачів стосуються обмеженого домену чи відкритого домену. В обмеженому домені набір можливих питань обмежений, а у відкритих навпаки. Іншою відмінністю є відповіді, які надаються. Так система з обмеженим доменом спирається на

онтологію та термінологію домену, у відкритому домені ж це загальна онтологія та світові знання. До систем з обмеженим доменом відносяться системи Start, Webcoop. Системами з відкритим доменом Webclopedia, Answerbus, Mulder. [1]

1.3 Класифікація за типами питань

Відповіді системи, які очікуються, залежать від типів питань, що ставлять користувачі. Для різних типів питань система потребує різних стратегій для визначення відповіді. За цією ознакою системи можна поділити на такі: фактичні(що, хто, який, скільки), спискові(сутності за особливістю), підтверджуючі(чи), гіпотетичні(що б сталося), причинні(як, чому). [1]

Питально-відповідальна система може підтримувати різні питання, що дозволяє одну систему віднести до декількох категорій. До фактичних питально-відповідальних систем відносяться Webclopedia, Naluri, Start, Answerbus, Webcoop, Mulder, до спискових належать Naluri, Start, Webcoop. Підтверджуючими системами є Webclopedia, Webcoop, а причинним Webclopedia, Naluri, Answerbus, Webcoop, Mulder.[1]

1.4 Класифікація за способом аналізу питань

Питально-відповідальним системам потрібно обробити питання користувача, щоб з’ясувати вимоги і знайти релевантні відповіді. При цьому документи також обробляються і аналізуються таким самим чином.

Є три основні підходи: статистичний підхід, підхід на відповідність правилам та гібридний, який поєднує попередні. [1]

1.5 Класифікація за типом сховища даних

Питально-відповідальні системи потребують даних для надання відповіді. За типом джерел цих даних можна поділити системи на такі категорії: структуровані джерела даних, напівструктуровані джерела даних та неструктуровані джерела даних.[1]

В структурованих джерелах дані структуровані у смислові набори. Схожі сутності об’єднання у відношення(relations). Сутності в однакових відношеннях мають однакові атрибути. Опис всіх сутностей називається схемою. Дані узгоджені відповідно до визначеного формату. Запит користувача перетворюється у запит до сховища, що дає точну відповідність питання і відповіді.[1]

У напівструктурованих джерелах самі дані і визначають схему.

Неструктуровані джерела даних допускають будь-які типи даних. Відсутні семантичні зв’язки та строгі правила організації даних. Для роботи з таким форматом використовується NLP або методи інформаційного пошуку.[1]

До напівструктурованих належать системи Naluri, Start, Webcoop, а до неструктурованих Webclopedia, Answerbus, Mulder.[1]

1.6 Класифікація за властивостями даних

Питально-відповідальні системи можна розділити за властивостями джерел даних: розмір, мова, гетерогенність, стиль, тип даних.[1]

Розмір джерела даних впливає на час пошуку та вибір правильних відповідей. Великий розмір збільшує шанси і точність надання правильної відповіді але, ризиками є збільшення NLP обробки та можливі проблеми при індексації.[1]

Документи різними мовами вимагають різних правил для обробки, при цьому універсальних правил для всіх мов не існує. Така особливість може дати перевагу у випадку комбінації даних, що містяться в різних документах різними мовами, але не всі мови мають чіткі граматичні правила, що ускладнить їх аналіз.[1]

Гетерогенність визначає об’єм різних сховищ даних і форматів з якими працює питально-відповідальна система. Така неоднорідність дозволяє отримувати кращі відповіді при розподілі інформації по різним джерелам. Але виникає потреба у перетворенні природної мови у запит, який джерело даних може опрацювати.[1]

Стиль даних та запитів може бути як формальним так і неформальним. Обробка неформальних даних стикається з труднощами через можливість відсутність синтаксису або формалізму. Це може призводити до неправильного аналізу запиту і документів у сховищі.[1]

Ще однією властивість є тип даних з якими працює питально-відповідальна система. Більшість систем працюють із текстовими документами, бо отримання відповідей є складною задачею на сьогодні.[3]

1.7 Класифікація за способом видачі відповіді

Питально-відповідальні системи поділяються за формою відповіді, яку вони надають користувачу. Існує 2 основні способи: витяг і генерація.[1]

Витяг відповіді передбачає, що користувачу надається документ або його частина з джерела даних. Документи можуть бути поділені на речення, а система покаже найбільш релевантне з них. Найкраще підходить для фактичних і підтверджуючих питань. Іншим способом є

розбиття документів на параграфи і користувач отримує найрелевантніший з них. Найкраще підходить для причинних і гіпотетичних питань. Ще одним варіантом витягу є відповідь користувачу у форматі мультимедіа.[1]

Генерація відповіді є альтернативним способом надання відповіді. Для підтверджуючих питань це буде “так” або “ні” за допомогою перевірки і підтвердження, що система проведе. Оцінки або рейтинги передбачають, що система надасть оцінку об’єкту або характеристиці об’єкта. Іншим способом є діалогові відповіді, коли система надає відповіді у форматі діалогу з користувачем.[1]

РОЗДІЛ 2: Огляд сервісів для створення питально-відповідальних систем

2.1 Dalogflow

Dialogflow - платформа для взаємодії користувачів із додатками, сайтами, пристроями, ботами та іншими застосуваннями. Декларує підтримку обробки природної мови, можливість обробки текстових і аудіо запитів та текстові або згенеровані аудіо відповіді.[4]

Перша версія була випущена у вересні 2014 року під назвою api.ai. У вересні 2016 року компанію із сервісом придбала компанія Google та у вересні 2017 року переіменувала на Dialogflow.[5]

Робота з Dialogflow базується на основних концептах: агент(agent), інтент(intent), сутність(entity), контекст(context). Вони є основою для побудови взаємодії з користувачами.

Агент у Dialogflow є віртуальним агентом, що обробляє бесіду з користувачами системи. Він є модулем, що розуміє природну мову та структурує дані отримані в рамках бесіди для подальшої обробки.[4]

Інтент класифікує намір користувача для ведення бесіди. Кожен агент може мати багато інтентів, які можуть бути використані при взаємодії з користувачем. Кожна репліка користувача співставляється з наявними інтентами та обирається найкраща.[4]

Інтент містить тренувальні фрази, що визначають які репліки користувача викликають даний інтент. Кожен інтент може містити багато тренувальних фраз. При цьому система декларує розширення вказаного списку схожими фразами за рахунок машинного навчання. Загальною рекомендацією є використання не менше 10 тренувальних фраз.[4]

Коли репліка користувача співпадає з якимось інтентом, Dialogflow витягує параметри з репліки. Кожен такий параметр має тип сутності, який

визначає як дані будуть витягатися. Для підтримки такого витягання на тренувальних фразах потрібно розмітити параметри і вказати їх тип сутності. Отримані параметри можна використати для побудови відповіді та передачі власному сервісу за допомогою webhook.[4]

Кожен інтент вимагає відповіді. Так Dialogflow підтримує статичні і динамічні відповіді. Статичні відповіді вказуються в інтенті, їх може бути декілька для різноманітності відповідей. Для динамічних відповідей є можливість додати параметри отримані з репліки користувача або використання власного сервісу із створення відповіді там. Базовою відповіддю є текст, але інші типи(картинка, звук тощо) також можливі в залежності від платформи, де працюватиме агент. При цьому відповідь може містити уточнююче питання, якщо не всі дані отримані в рамках одного інтенту.[4]

Сутність це параметр, який може бути у репліці користувача. Кожна сутність має тип. Dialogflow має початкові типи такі як: час, дата, довжини та інші. При цьому є можливість визначти власний тип, якщо готові не підходять. Кожному типу сутності відповідають конкретні сутності, які потрібно вказати. При цьому декілька різних слів можуть відповідати одній сутності.[4]

Контексти в Dialogflow близькі до контекстів у природній мові. Вони потрібні для побудови сценаріїв діалогу. Так кожен інтент може приймати і повертати контексти. При активації інтенту повертається вихідний контекст. І при наступному виборі інтенту будуть обиратися ті, в яких вхідний контекст співпадає попереднім вихідним.[4]

Для початку роботи з Dialogflow потрібно мати Google-акаунт та перйти до консолі сервісу(https://dialogflow.cloud.google.com/). Після цього потрібно натиснути на кнопку створення агента. Після цього відкриється

вікно створення, де потрібно вказати ім’я агента, мову за замовчуванням та часовий пояс за яким запити будуть вказувати час. Приклад такого вікна на рисунку 2.1.

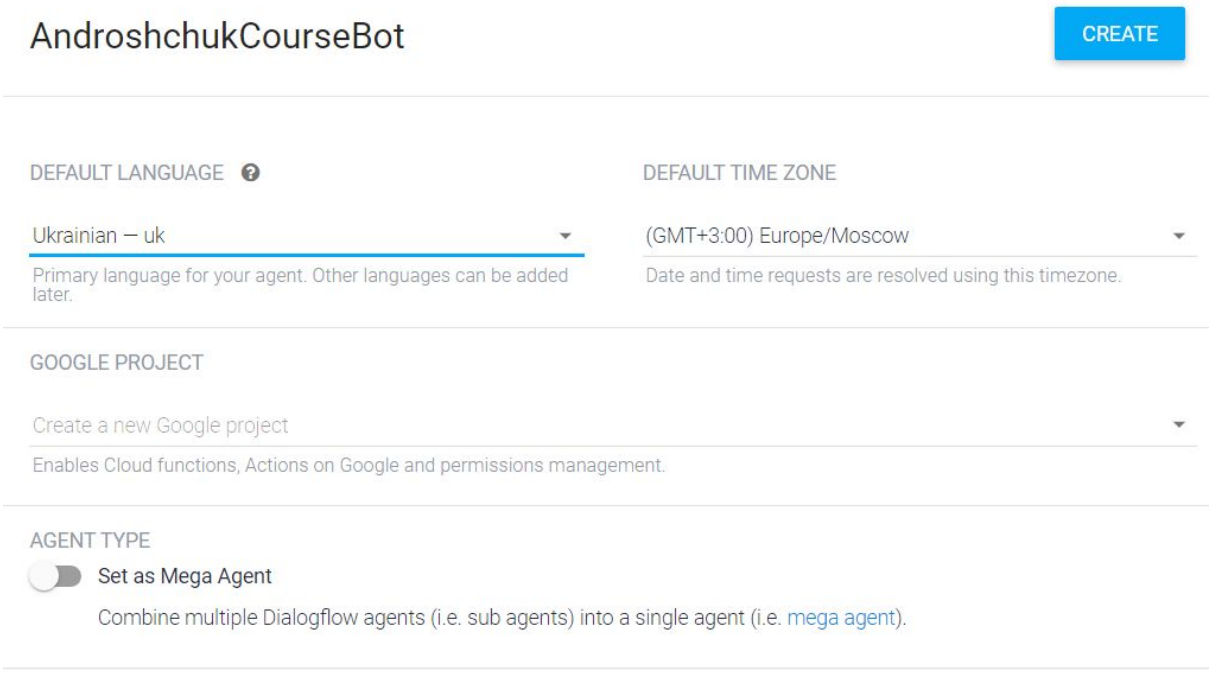

Рисунок 2.1

Після натискання на кнопку створення агент буде створений і відкриється вікно з налаштуваннями.

Далі потрібно обрати на яких платформах буде працювати агент. При цьому частина платформ 6 червня 2020 зникне з Dialogflow. Серед платформ, що зникнуть такі: Kik, Skype, Spark, Twilio IP Messaging, Twilio (Text Messaging), Twitter, Viber. Для роботи було обрано інтеграцію з месенджером Telegram.[4]

При обиранні з’являється діалогове вікно з інструкціями приклад зображено на рисунку 2.2. У випадку Telegram потрібно ввести токен для бота, який буде використовувати агент Dialogflow. Інтеграцію можна зупинити в будь-який момент змінивши значення перемикача.

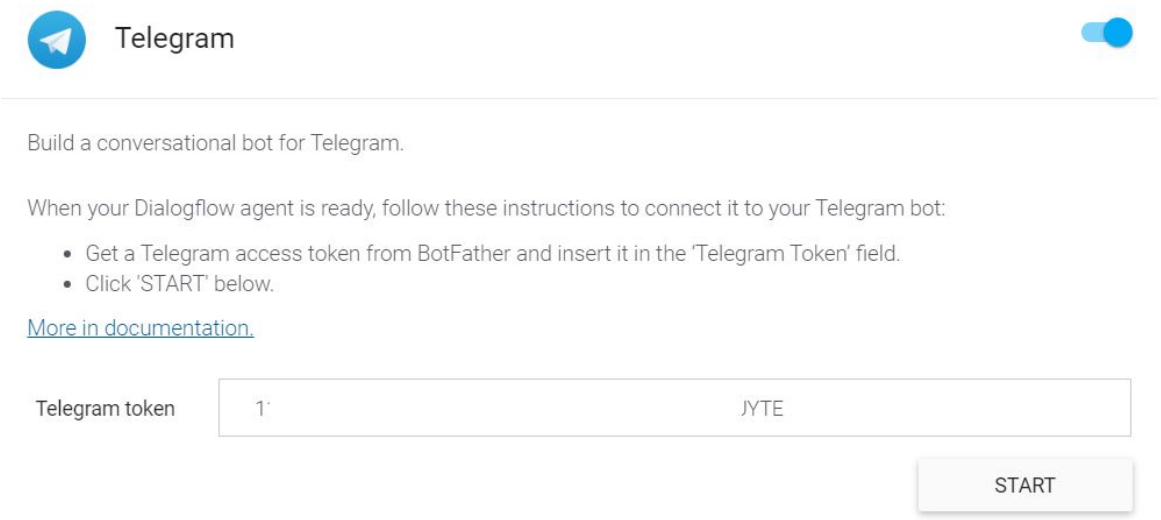

Рисунок 2.2

На початку підтримуються 2 види інтентів: привітання та невдача(Fallback). Інтент невдачі використовується, коли решта інтентів не підійшли. Цих інтентів досить для перевірки конфігурації та інтеграції. На рисунку 2.3 зображено фрагмент спілкування з системою через Telegram.

Рисунок 2.3

В даному випадку при надсиланні повідомлення системі, на першу репліку співставлено інтент привітання, а на другу фразу визначено інент невдачі.

2.2 IBM Watson Assistant

IBM Watson Assistant - платформа для побудови питально-відповідальних систем. Створена компанією IBM в рамках проекту комп'ютерного штучного інтелекту IBM Watson .[6]

Основні концепти, якими оперує IBM Watson Assistant: інтенти та сутності

Інтент є колекцією реплік користувачів з однаковим значенням. Якщо у репліці користувача розпізнає інтент, то він піде по сценарію діалога. [6]

Сутність представляє інформацію із репліки користувача, яка стосується його наміру.[6]

Для початку роботи потрібно мати IBM Cloud аккаунт. Далі вибрати створення Watson Assistant. Після цього потрібно обрати регіон, де буде знаходитися сервіс, вказати тарифний план та обрати назву. Після цього натиснути на кнопку створення. Після цього відкриється вікно налаштувань сервіса. Приклад такого вікна зображено на рисунку 2.4.

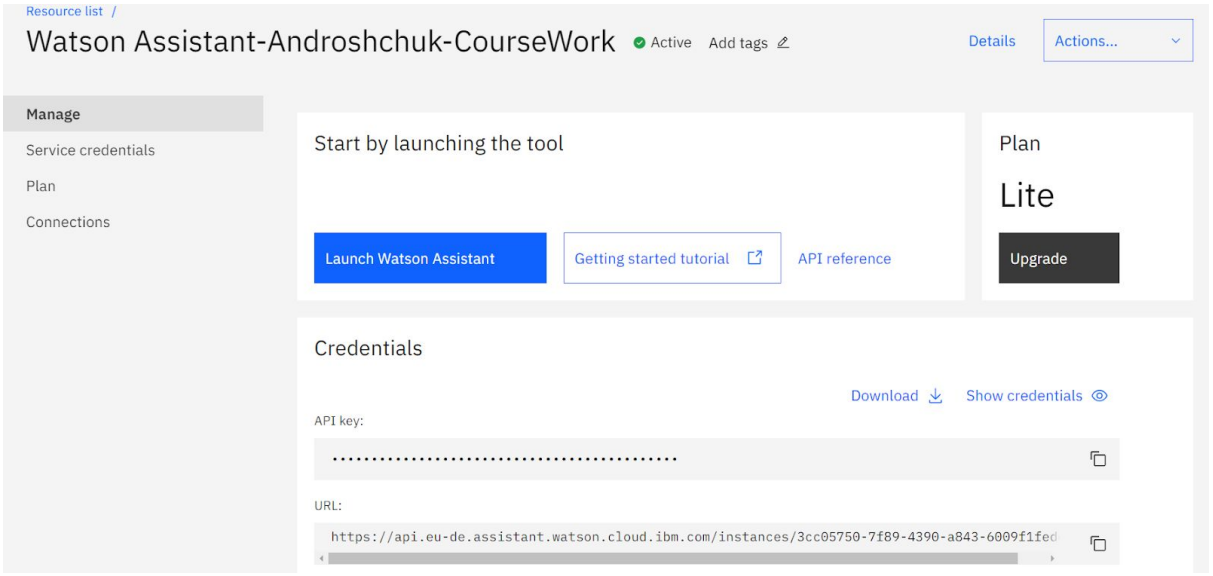

Рисунок 2.4

Після цього можна перейти до налаштувань за кнопкою запуска Watson Assistant. Відкриється список поточних асистентів з можливістю додати нового на рисунку 2.5 зображено частину користувацького інтерфейсу.

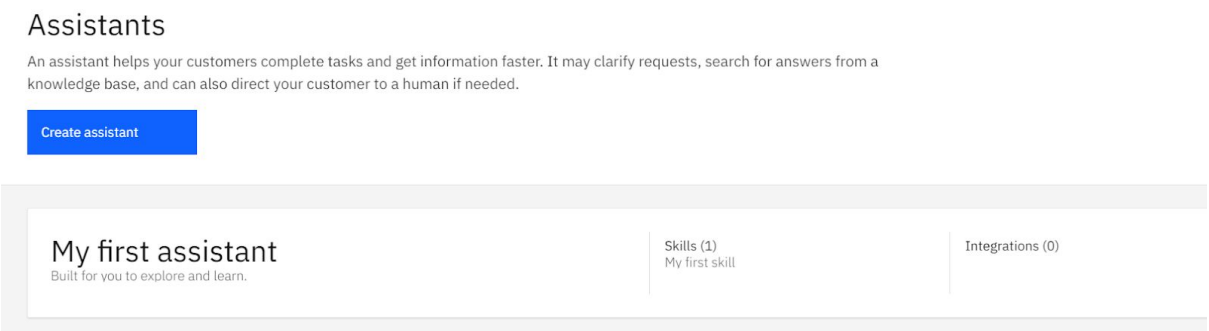

Рисунок 2.5

При виборі асистента відкриються його налаштування.

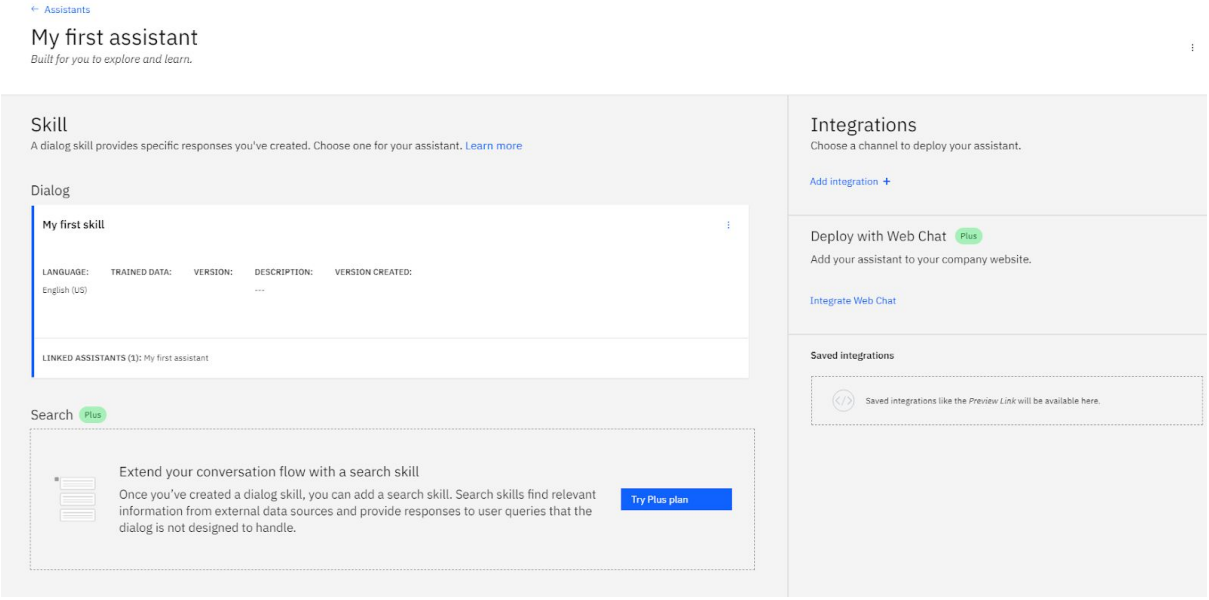

Рисунок 2.6

Підтримується інтеграція з власним сайтом, через окрему веб-сторінку, Facebook Messenger, Slack, Intercom. Також є можливість створити власну інтеграцію за допомогою API.[6] Для подальшої роботи обрано інтеграцію через окрему веб-сторінку, що дозволить швидко розгорнути систему.

На початку є лише діалог привітання та невдачі. Можливо додати вузол до дерева діалогу, як дочірній вузол іншого вузла.

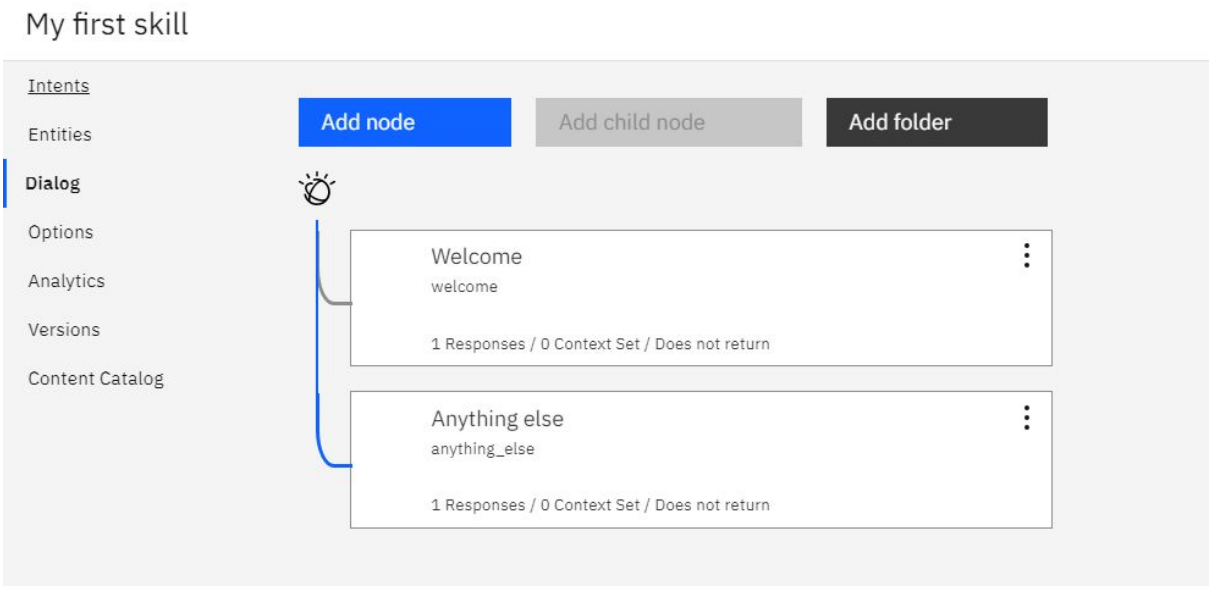

Рисунок 2.7

Потрібно вказати ім’я діалогу, чи повинно користувачу пропонуватися перехід на цей діалог, умова запуску діалогу, відповідь користувачеві та дія після надання відповіді. Після надання відповіді може звичайне очікування наступної репліки або перехід на інший діалог. Перехід може бути з очікуванням репліки, відповіддю при відповідності попередньої репліки та безумовною відповіддю.

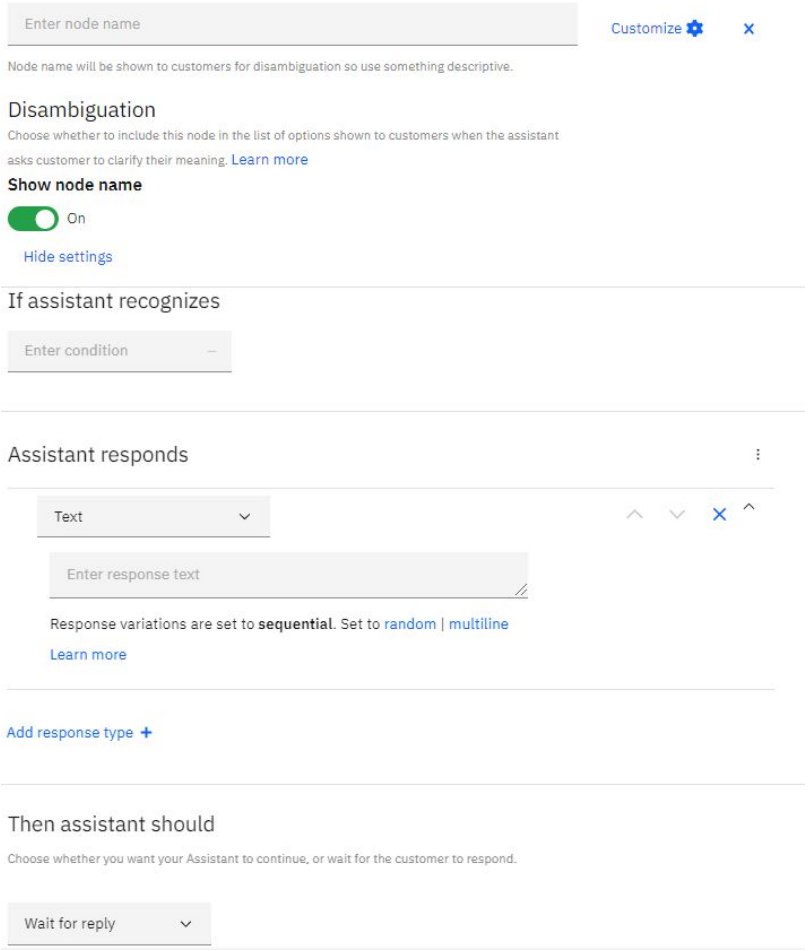

Рисунок 2.8

Для прикладу додамо вузли загальне привітання та питання про найкращий університет. Вони не зв’язані між собою, тому ніякої залежності в порядку виклику.

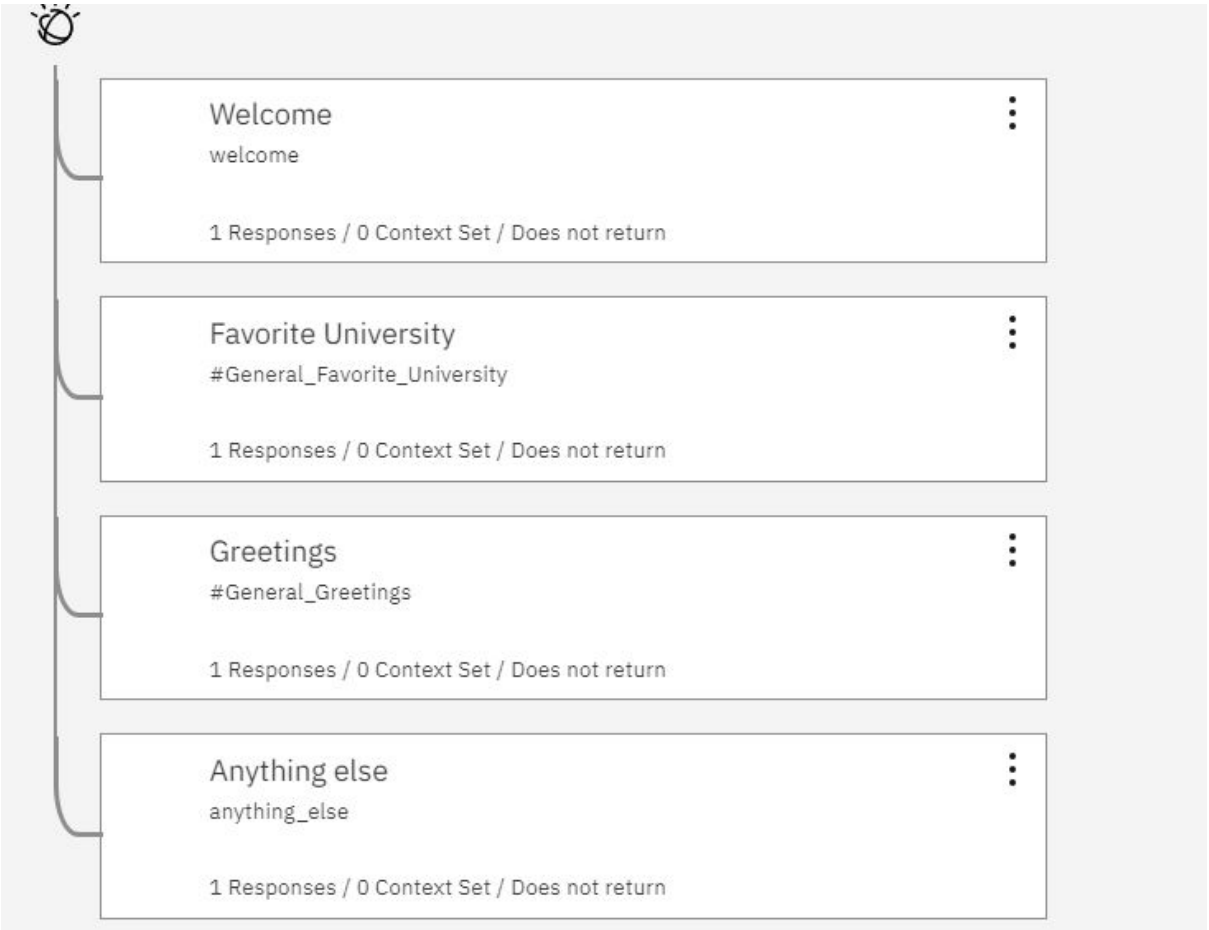

Рисунок 2.9

Тепер можна перевірити, які відповіді надасть система, на рисунку 2.10 зображена частина інтерфейсу з питаннями та відповідями.

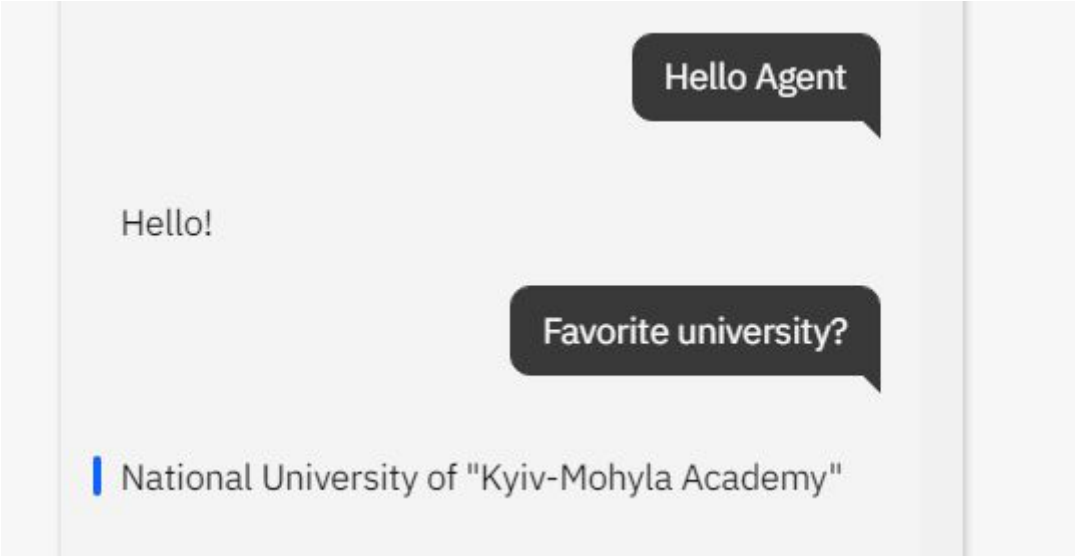

Рисунок 2.10

2.3 Сервіси Microsoft Azure

Microsoft Azure - платформа хмарних обчислень, створена компанією Microsoft. Пропонує SaaS, PaaS, IaaS рішення. Для створення питально-відповідальної системи має декілька сервісів: Personalizer, для персоналізованих відповідей користувачу, QnA Maker, база знань для відповідей, Language Understanding, для виділення намірів у репліках користувача, Text Analytics, для виділення настрою, ключових фраз та іменованих сутностей, Azure Bot для взаємодії з користувачем та викликів інших сервісів.[7]

Language Understanding (LUIS) - хмарний сервіс, що надає API для обробки реплік користувача. Використовує машинне навчання для визначення значення та отримання релевантної детальної інформації з фраз користувача. Будь-який розмовний додаток, що працює з природною мовою, може використовувати LUIS, наприклад чат-боти. Сервіс підтримує англійську, китайську, французьку, італійську, німецьку, іспанську, японську мови та деякі інші, але української серед них немає.[8]

Додаток-клієнт відправляє текст, введений користувачем на ендпоінт LUIS. Після чого сервіс співставляє текст з можливими інтентами і вибирає найбільш імовірний. Далі намагається виділити сутності наявні в повідомленні. Після чого передає ці дані додатку-клієнту для їх подальшої обробки. [8]

QnA Maker - хмарний NLP сервіс, який створює шар природної мови над даними. Може використовуватися для найбільш доречної відповіді користувачу з бази знань при запиті природною мовою. Користувачами сервісу є розмовні додатки, які взаємодіють з користувачами природною мовою.[8]

QnA Maker рекомендують використовувати у випадку наявності статичної інформації у базі знань для відповідей або при однаковій відповіді різним користувачам на однакові питання. Прикладами такої інформації є FAQ, документація, брошури, тощо. наявність статичної інформації, наприклад pdf документи, веб ресурси з відповідями, тощо.[8]

Додаток-клієнт відправляє питання на ендпоінт QnA Maker. Після цього сервіс, використовуючи базу знань, надає відповідь і можливі уточнення для видачі найкращої відповіді. Додаток-клієнт отримує відповідь надану сервісом з оцінкою відповідністю і може перенаправити її користувачу або запропонувати уточнення.[8]

Для початку роботи потрібно мати аккаунт Microsoft, кошти на рахунку Azure для розміщення сервісів та увійти у Microsoft Azure.

Основою для роботи є Azure Bot Service Bot. Цей сервіс безпосередньо взаємодіє з користувачами, отримує та відправляє повідомлення, наприклад чат-боти.[8]

В Azure Portal потрібно обрати вікно Bot Services. Після цього відкриється список наявних ботів з можливістю створити нового.

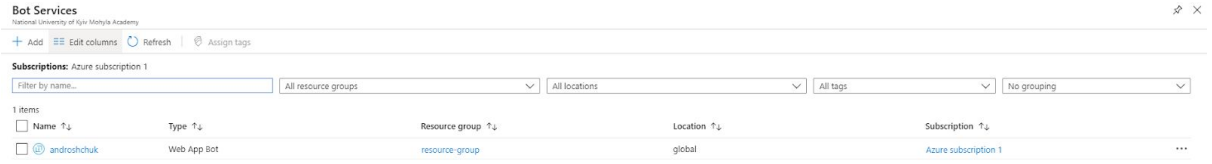

Рисунок 2.11

Далі потрібно обрати створення нового бота, після чого сервіс пропонує обрати тип бота: Web App Bot або Bot Channels Registration. Перший варіант передбачає, що бот буде створений та розгорнутий всередині Azure як Azure App Service Web App. Другий же передбачає наявність бота розгорнутого поза Azure та його під’єднання до Bot Service, цей варіант в рамках роботи не розглядається.

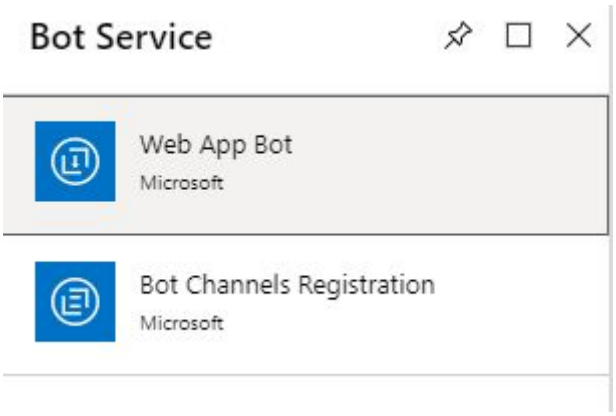

Рисунок 2.12

Після обрання Web App Bot та переходу на створення, буде відкрито вікно з налаштуваннями. Потрібно вказати назву, параметри ресурсів Azure, регіон розгортання сервісу, та шаблон бота. На вибір дається Echo Bot, який повертає написане повідомлення, та Basic Bot, який містить LUIS та Bot Analytics. В рамках цієї роботи розглядатиметься варіант бота написаного на платформі Node.js.

Після цього можна перевірити поведінку бота у інтерфейсі Azure інструментом Web Chat, що доступний в консолі сторінки бота. Оскільки базова версія бота не має достатніх можливостей для створення власної системи, то потрібно завантажити код бота та розпакувати в окрему директорію. Можливе використання шаблонів для ботів, які присутні в Інтернеті.

Перед початком роботи потрібно створити LUIS сервіс. При створенні потрібно вказати назву, мову, групу авторів, та ключ.

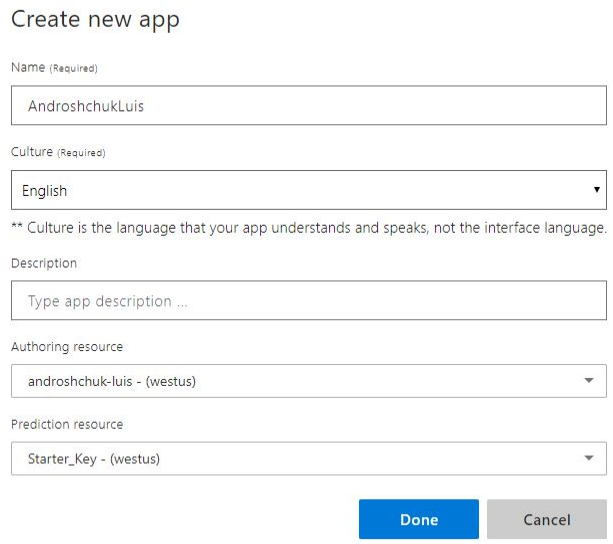

Рисунок 2.13

Після створення потрібно додати інтенти, які повинні розпізнавати сервіс. Є стандартні інтенти для погоди, автоматизації будинку, замовлення квитків та інших. Якщо ці інтенти не підходять, потрібно створити нові і додати фрази, які йому відповідають, і сутності, що можуть бути передані. Створимо два інтенти FavoriteUniversity та DeanComputerScience.

Далі необхідно запустити тренування моделі та опублікувати сервіс. Інтерфейс LUIS надає можливість відразу перевірити модель. Користувач надсилає повідомлення, а у відповідь йому повертається інтент з оцінкою ймовірності.

Наступним сервісом є QnA Maker. Потрібно створити нову базу знань, де вказати параметри розміщення сервісу в Azure, мову бази знань, її назву та документи для її наповнення.

Після наповнення базу знань можна перевірити і доповнити іншими парами питання-відповідь. Для кожної такої пари можна додати підказку для подальших запитань.

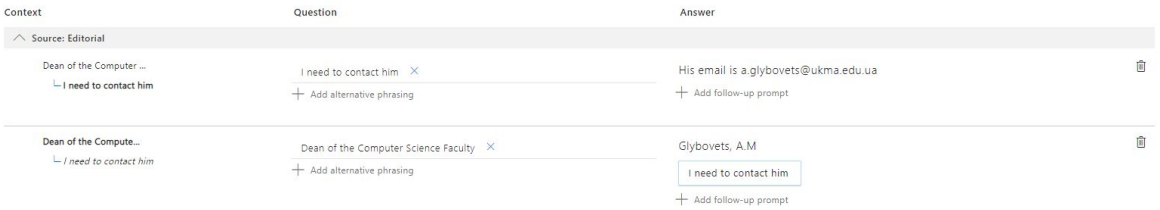

Рисунок 2.14

Після наповнення потрібно опублікувати сервіс для повноцінного використання.

Робота з ботом передбачає наявність директорії вихідних кодів сервісу. У директорії містяться файли, які відповідають за налаштування бота та виклики інших сервісів. Файл .env містить ідентифікатори, ключі, та адреси про сам сервіс бота та сторонні сервіси, що буде використовувати бот, в даному випадку LUIS та QnA Maker.

MicrosoftAppId=0c3aec0a-bfe9-460c-891e-6e7a5107d3f8
MicrosoftAppPassword=f{;41u(Akq{1eyMmh}xYipgBcaUDj68D
QnAKnowledgebaseId=4982d16a-8fd6-4992-ab2b-72d7cb9c5079
QnAEndpointKey=c55ebe2b-05ec-428f-ae5e-dfdefaeaa63b
QnAEndpointHostName=https://androshchuk-knowledge-base.azurewebsites.net/qnamaker
LuisAppId=7d1e07f3-2490-4089-ba9d-f3af9e1d32d6
LuisAPIKey=2d6c67d7c4904483a2893740566251bb
LuisAPIHostName=westus

Поєднання декількох сервісів вимагає наявність диспетчеризації між цими файлами. Для цього є Dispatch tool, який потрібно додати через менеджер пакетів Node.js - npm.

npm i -g npm
npm i -g botdispatch

Далі потрібно створити сам файл диспетчеризації через команду dispatch init та додати сервіси, які будуть використовуватися через dispatch add. Після додавання потрібно створити модель командою dispatch create.

Для запуску бота потрібно додатково встановити пакети

npm install --save botbuilder
npm install --save botbuilder-ai
npm install --save dotenv

Далі потрібно налаштувати код бота для правильного процесу.

```
const { ActivityHandler } = require('botbuilder');
const { LuisRecognizer, QnAMaker } = require('botbuilder-ai');

class DispatchBot extends ActivityHandler {
  constructor() {
    super();
    const dispatchRecognizer = new LuisRecognizer({
      applicationId: process.env.LuisAppId,
      endpointKey: process.env.LuisAPIKey,
                        endpoint: `https://${ process.env.LuisAPIHostName
}.api.cognitive.microsoft.com`
    }, {
      includeAllIntents: true,
      includeInstanceData: true
    }, true);

    const qnaMaker = new QnAMaker({
      knowledgeBaseId: process.env.QnAKnowledgebaseId,
      endpointKey: process.env.QnAEndpointKey,
      host: process.env.QnAEndpointHostName
    });

    this.dispatchRecognizer = dispatchRecognizer;
    this.qnaMaker = qnaMaker;

    this.onMessage(async (context, next) => {
                                    const recognizerResult = await
dispatchRecognizer.recognize(context);
      const intent = LuisRecognizer.topIntent(recognizerResult);
                      await this.dispatchToTopIntentAsync(context, intent,
recognizerResult);
      await next();
    });

    this.onMembersAdded(async (context, next) => {
          const welcomeText = 'Type any message or a question to get
started.';
```

```
        const membersAdded = context.activity.membersAdded;
        for (const member of membersAdded) {
          if (member.id !== context.activity.recipient.id) {
                await context.sendActivity(`Welcome to bot ${ member.name }.
${ welcomeText }`);
          }
        }
        await next();
      });
    }

    async dispatchToTopIntentAsync(context, intent, recognizerResult) {
      switch (intent) {
      case 'FavoriteUniversity':
                                    await    this.processFavoriteUniversity(context,
recognizerResult.luisResult);
        break;
      case 'DeanComputerScience':
        await this.processQnA(context);
        break;
      default:
        console.log(`Dispatch unrecognized intent: ${ intent }.`);
          await context.sendActivity(`Dispatch unrecognized intent: ${ intent
}.`);
        break;
      }
    }

    async processFavoriteUniversity(context, luisResult) {
      const result = luisResult.connectedServiceResult;
      await context.sendActivity(`My favorite university NaUKMA `);
    }

    async processQnA(context) {
      const results = await this.qnaMaker.getAnswers(context);
      if (results.length > 0) {
        await context.sendActivity(`${ results[0].answer }`);
      } else {
        await context.sendActivity('Sorry, could not find an answer');
      }
    }
}
```

```
module.exports.DispatchBot = DispatchBot;
```

Перевірити роботу бота можна розгорнувши його на власній машині або опублікувавши на Azure. Для локальної перевірки потрібно запустити бота командою npm start і використати Bot Framework Emulator - інструмент від Microsoft для локальної перевірки поведінки ботів. За замовчуванням бот стає доступний за адресою http://localhost:3978/api/messages. При відкритті бота необхідно вказати дані сервіса, що використовує бот: MicrosoftAppId та MicrosoftAppPassword. В іншому разі запити до сторонніх сервісів будуть невдалими через потребу в авторизації.

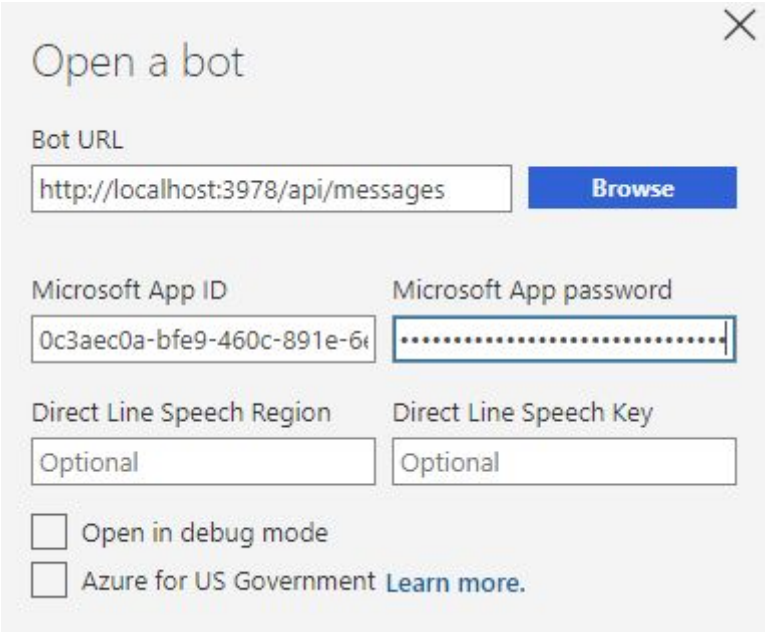

Рисунок 2.15

Після цього можна перевіряти роботу бота у відповідному вікні.

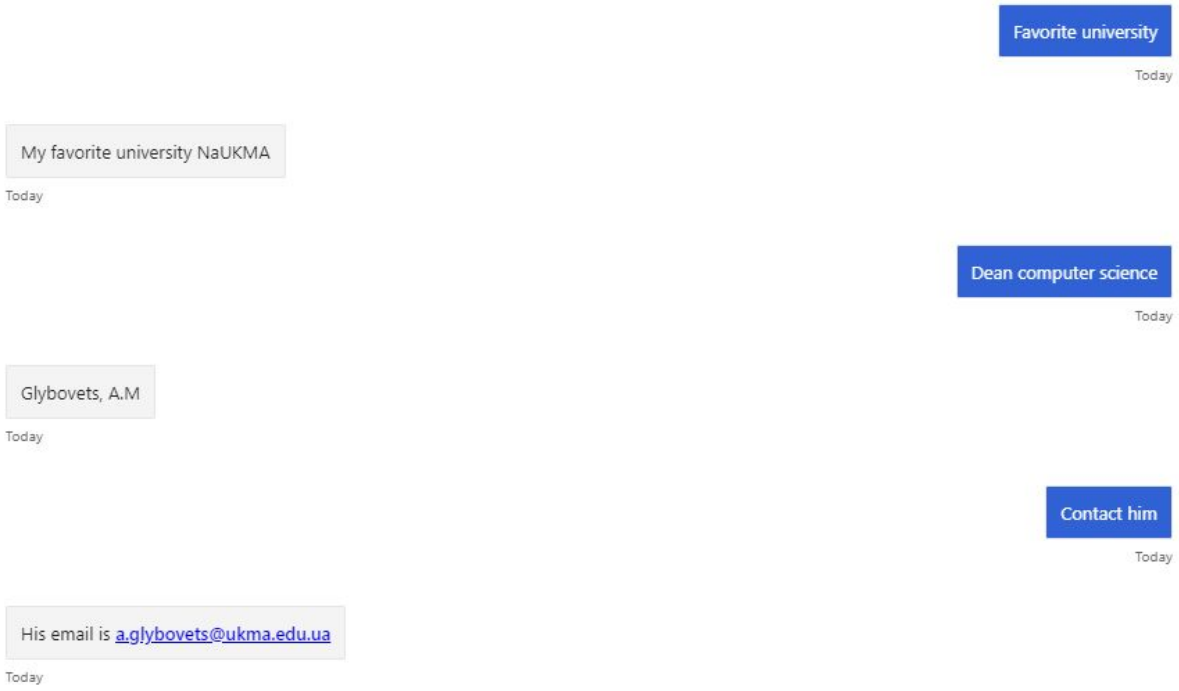

Рисунок 2.16

Для розгортання бота в Azure необхідно встановити Azure CLI та увійти у свій обліковий запис через команду az login. Далі архівувати директорію, де знаходяться вихідні коди бота та виконати команду az webapp deployment source config-zip, вказавши в параметрах групу ресурсів, назву сервісу та шлях до zip-архіву. Через декілька хвилин бот буде розгорнуто в Azure з можливістю взаємодії через Інтернет.

РОЗДІЛ 3: Створення питально-відповідального застосунку

3.1 Обґрунтування вибору сервіса

Для створення було обрано Dialogflow з огляду на підтримку української мови, можливості інтеграції з окремими сервісами та наявності плану з безкоштовними повідомленнями. Так в сервісах Azure українська мова доступна лише у QnA Maker, а IBM Watson Assistant взагалі не підтримує її.

В рамках безкоштовного плану надається 180 текстових запитів на хвилину, 60 запитів на зміну агента на хвилину, 100 запитів на запити в рамках сесії користувачів. Для прототипу цих обмежень достатньо.[9]

3.2 Інтеграції Dialogflow

У разі потреби динамічних відповідей Dialogflow дозволяє додати інтеграції з стороннім сервісом.Кожен інтент має налаштування, що вказує потребу в зовнішньому сервісі. Якщо воно не увімкнене, то буде використана статична відповідь. Тому для динамічних відповідей потрібно, щоб це поле було увімкнене. [4]

При визначення інтента з динамічною відповіддю Dialogflow надсилає запит на webhook сервісу з інформацією про інтент. Далі сторонній сервіс обробляє ці дані та повідомляє Dialogflow інформацію необхідну для відповіді через webhook. [4] На рисунку 3.1 зображено схему взаємодії користувача з Dialogflow та сторонньою системою через webhook.

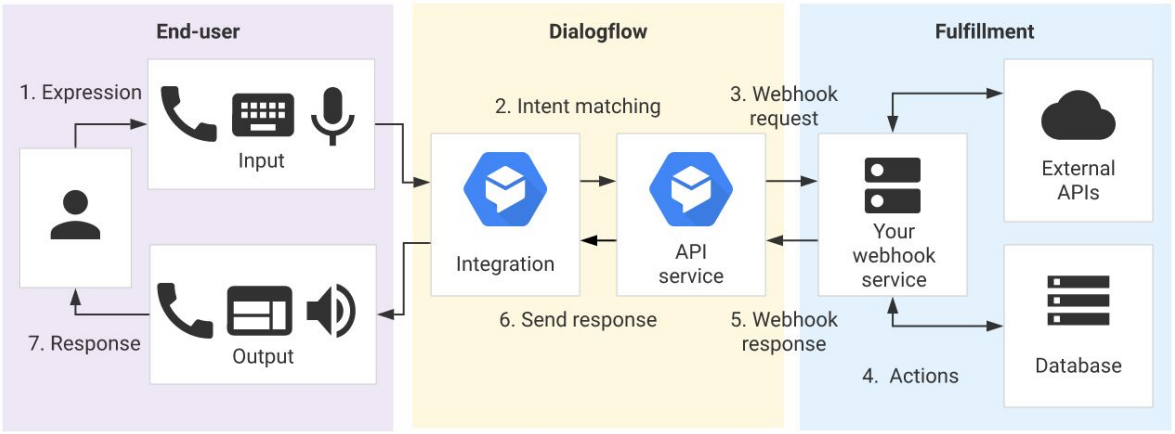

Рисунок 3.1[4]

Webhook стороннього сервісу повинен приймати та повертати повідомлення у форматі JSON відповідно до специфікації Dialogflow. Має приймати HTTPS запити, бути публічно доступним, приймати POST запити WebhookRequest та відповідати WebhookResponse.[4]

3.3 Створення власного сервіса

Архітектура системи передбачає наявність Spring boot сервісу інтегрованого з Dialogflow. Для спрощення розробки таких сервісів була створена бібліотека Actions on Google для Java. Там присутні класи, які імплементують частину логіки Dialogflow: DialogflowApp, ActionContext, DefaultApp та інтерфейси для можливості власної реалізації.

Архітектура застосунку містиме контрол для обробки webhook запитів. Далі дані із запиту передаються власному класу, який розширює CourseworkDialogflowApp. Всередині CourseworkDialogflowApp класу містяться методи для обробки інтентів. Анотація @ForIntent мітить параметром назву інтента, який буде перехоплюватися і оброблятися цим методом. Кожен метод повинен повернути у відповідь об’єкт класу ActionResponse з даними, які отримає користувач.

Для роботи системи потрібно мати дані для відповіді. Це можуть бути файли, СКБД чи просто значення в коді. При параметризованному інтенті система використовує ці дані для надання відповіді.

3.4 Розгортання застосунку

Для розгортання застосунку було обрано сервіс Heroku. Heroku - хмарний сервіс типу PaaS, що дозволяє розгортати застосування. Підтримуються Node.js, Java, Ruby та інші фреймворки з різними мовами програмування.

Сервіс надає можливість розгортати застосування через Heroku CLI або через інтеграцію з GitHub. Оскільки, вихідні коди сервіса розміщено на GitHub, то було обрано саме цей варіант. Для його використання потрібно надати Heroku доступ до GitHub аккаунта. Вказати параметри розгортання: гілку з якої буде вихідний код та чи увімкнути автоматичне розгортання. Так на рисунку 3.2 зображено частину користувацького інтерфейсу, що відповідає за розгортання.

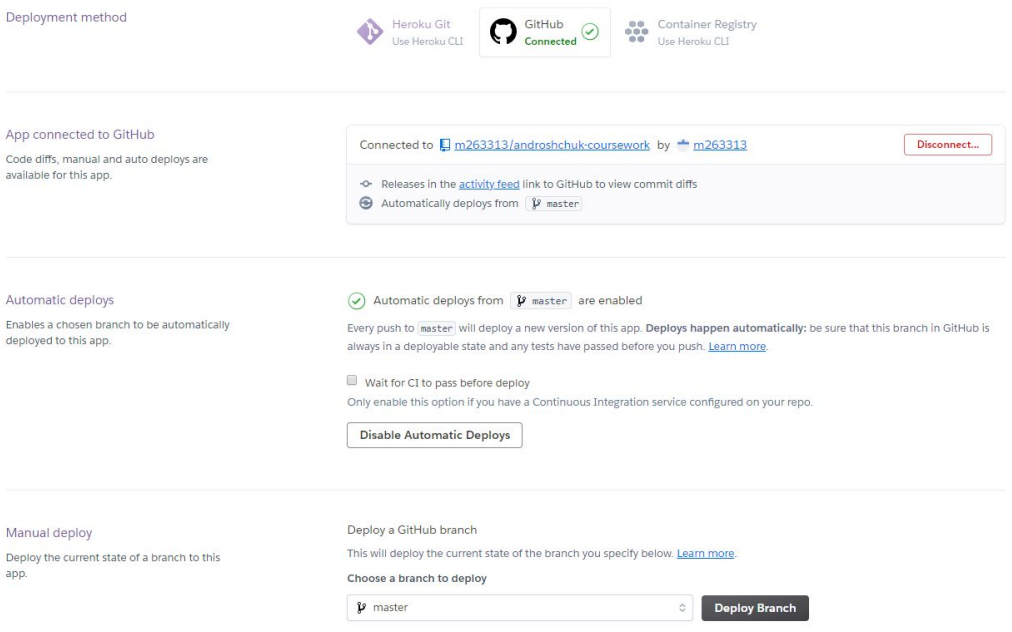

Рисунок 3.2

Після розгортання застосунок стає доступним за URL у форматі <назва проекту>.herokuapp.com/. Також надається можливість додати інше доменне ім’я, якщо таке є в наявності. На рисунку 3.3 зображено відповідну частину у інтерфейсі.

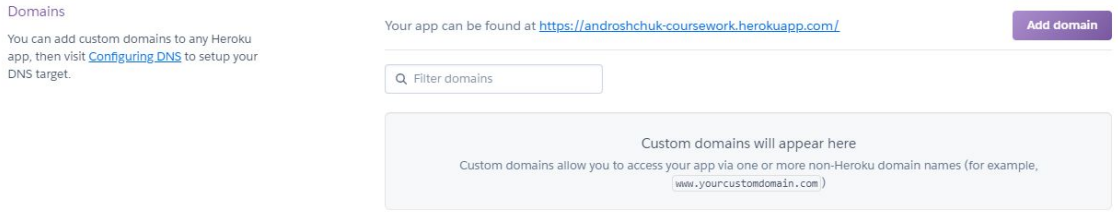

Рисунок 3.3

Після цього потрібно додати до Dialogflow адресу розгорнутого сервісу і застосунок питально-відповідальної системи буде доступний через вказані інтеграції.

Висновки

В результаті виконання курсової роботи отримано прототип програмного застосунку питально-відповідальної системи.

В ході написання курсової роботи було оглянуто способи класифікації питально-відповідальних систем. В рамках другого розділу було розглянуто сервіси для створення питально-відповідальних систем та створені тестові варіанти питально-відповідальних систем.

В третьому розділі було розглянуто створення програмного застосунку питально-відповідальної підсистеми. Описано архітектуру застосунку, використані технології та механізми розгортання.

Підсистема має потенціал для розширення при наповненні даними пов’язаними з університетом, факультетами та навчальним процесом загалом. Це дає можливість спростити отримання відповідної інформації через усунення потреби пошуку інформації у різних джерелах.

Список використаної літератури

1. Marco Antonio, Calijorne Soares, Fernando Silva Parreiras. “A literature review on question answering techniques, paradigms and systems”, 2018 [Електронний ресурс] URL: https://www.sciencedirect.com/science/article/pii/S131915781830082X?via%3Dihub
2. Question answering [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Question_answering
3. N. Indurkhya, F.J. Damereau (Eds.). “Handbook of Natural Language Processing” (second ed.), Chapman & Hall/CRC, Boca Raton (2010)
4. Dialogflow documentation [Електронний ресурс] URL: https://cloud.google.com/dialogflow/docs/
5. Dialogflow [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Dialogflow
6. IBM Cloud Docs [Електронний ресурс] URL: https://cloud.ibm.com/docs/assistant
7. Microsoft Azure [Електронний ресурс] URL: https://en.wikipedia.org/wiki/Microsoft_Azure
8. Azure Cognitive Services documentation [Електронний ресурс] URL: https://docs.microsoft.com/en-us/azure/cognitive-services/
9. Quotas and limits [Електронний ресурс] URL: https://cloud.google.com/dialogflow/quotas