МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ЧАТ-БОТІВ З ВИКОРИСТАННЯМ БІБЛІОТЕК PYTHON»**

Виконав: студент 4 курсу, групи 6.1221-з
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
Ю. Д. Доросевич
(ініціали та прізвище)
Керівник професор кафедри комп'ютерних наук, доцент, д.е.н., Полуектова Н.Р.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри програмної інженерії, к.ф.-м.н., доцент Лісняк А. О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.В..
(підпис)

" ____ " _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Доросевичу Юрію Дмитровичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка чат-ботів з використанням бібліотек Python

керівник роботи Полуектова Наталія Робертівна, д.е.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » 12 2024 року № 2002-с

2. Строк подання студентом роботи

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка інтелектуального чат-бота з використанням бібліотек Python

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 10.01.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	01.02.2025	
2.	Збір вихідних даних.	01.03.2025	
3.	Обробка методичних та теоретичних	25.03.2025	
	джерел.		
4.	Розробка першого та другого розділу.	10.04.2025	
5.	Розробка третього розділу.	05.05.2025	
6.	Оформлення та нормоконтроль	14.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	29.05.2025	

Студент _____ Ю.Д. Доросевич
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.Р. Полуектова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка чат-ботів з використанням бібліотек Python»: 38 с., 14 рис., 5 табл., 11 джерел, 6 додатків.

ЧАТ-БОТ, ШТУЧНИЙ ІНТЕЛЕКТ, PYTHON, RASA, WEB-АРХІТЕКТУРА.

Об’єкт дослідження – сучасні технології створення чат-ботів

Предмет дослідження – бібліотеки Python для реалізації інтелектуальних чат-ботів

Мета роботи: розробити інтелектуального чат-бота з використанням спеціалізованих бібліотек Python.

Для виконання мети реалізовані наступні завдання:

– аналіз існуючих технологічних рішень для розробки чат-ботів;
– вивчення технологій на базі Python;
– розробка власного інтелектуального чат-бота з використанням обраних засобів;
– тестування розробленого програмного продукту.

SUMMARY

Bachelor’s Qualification Thesis: "Development of Chatbots Using Python Libraries": 38 pages, 14 figures, 5 tables, 11 references, 6 supplements.

CHATBOT, ARTIFICIAL INTELLIGENCE, PYTHON, RASA, WEB ARCHITECTURE.

Object of research – modern technologies for chatbot development. Subject of research – Python libraries for implementing intelligent chatbots. Objective: To develop an intelligent chatbot using specialized Python libraries.

To achieve this objective, the following tasks were completed:

– analysis of existing technological solutions for chatbot development;
– study of Python-based technologies;
– development of a custom intelligent chatbot using the selected tools;
– testing of the developed software product.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз теоретичних аспектів чат-ботів 8
1.1 Поняття та класифікація чат-ботів 8
1.2 Архітектура та принципи роботи чат-ботів 9
1.3 Аналіз бібліотек Python для створення чат-ботів 11
1.4 Застосування чат-ботів у різних сферах 14
2 Інструменти та середовище розробки чат-бота 17
2.1 Обгрунтування вибору фреймворку Rasa..... 17
2.2 Можливості інтеграції чат-бота на Rasa з месенджерами 18
2.3 Використання штучного інтелекту та обробки природної мови (NLP) у чат-ботах на Rasa..... 19
2.4 Обгрунтування вибору середовища розробки 21
3 Проектування та розробка чат-бота 23
3.1 Постановка задачі 23
3.2 Настроювання середовища розробки..... 24
3.3 Опис основних файлів 25
3.4 Додавання web-інтерфейсу 29
Висновки 31
Перелік посилань..... 32
Додаток А Код файлу config.yml 33
Додаток Б Код файлу domain.yml..... 34
Додаток В Код файлу nlu.yml 35
Додаток Г Код файлу rules.yml 36
Додаток Д Код файлу stories.yml 37
Додаток Е Код файлу Action.py 38

ВСТУП

У сучасному цифровому світі, що стрімко розвивається, зростає потреба в ефективних засобах автоматизації взаємодії між користувачами та комп'ютерними системами. Одним із таких засобів є чат-боти – програмні системи, здатні імітувати спілкування з людиною за допомогою природної мови. Завдяки активному розвитку штучного інтелекту, машинного навчання та обробки природної мови, чат-боти перетворилися з простих скриптів у складні інтелектуальні системи, які знаходять широке застосування у сферах обслуговування, торгівлі, освіти, медицини та багатьох інших.

Однією з найпопулярніших мов програмування для створення чат-ботів є Python. Вона пропонує широкий спектр бібліотек і фреймворків, які значно спрощують процес розробки інтелектуальних агентів. Зокрема, бібліотеки, такі як Rasa, дозволяють створювати складні діалогові моделі з підтримкою контексту, інтеграцією зі сторонніми сервісами та адаптацією під специфічні потреби користувача.

Актуальність обраної теми зумовлена постійним зростанням попиту на інтелектуальні автоматизовані рішення, що забезпечують зручну та ефективну комунікацію в цифровому середовищі. Розробка чат-бота із використанням бібліотек Python дозволяє не лише продемонструвати практичне застосування сучасних технологій, а й дослідити можливості їх інтеграції в реальні інформаційні системи.

Метою цієї кваліфікаційної роботи є розробка інтелектуального чат-бота з використанням спеціалізованих бібліотек Python. Для досягнення поставленої мети виконано аналіз сучасних технологій створення чат-ботів, досліджено особливості фреймворків Python, реалізовано власного чат-бота та проведено тестування розробленого програмного продукту.

1 АНАЛІЗ ТЕОРЕТИЧНИХ АСПЕКТІВ ЧАТ-БОТІВ

1.1 Поняття та класифікація чат-ботів

Чат-боти (англ. *chatbots*) – це програмні агенти, які здатні автоматизовано вести діалог з користувачами за допомогою текстового або голосового інтерфейсу [2]. Основна мета чат-бота – імітувати людську розмову та забезпечувати інтерактивну взаємодію з користувачами без участі людини-оператора. Сучасні чат-боти використовуються в різних галузях, таких як електронна комерція, банківські послуги, освіта, охорона здоров’я та технічна підтримка.

Чат-боти класифікуються за кількома критеріями[3]:

а) за ступенем складності та інтелектуальності
   - прості (rule-based) – працюють за заданими правилами або сценаріями. Вони реагують на ключові слова або конкретні команди та не здатні до гнучкої обробки мови;
   - інтелектуальні (AI-based) – використовують машинне навчання, обробку природної мови (NLP) та штучний інтелект для розуміння контексту та формування відповідей. Такі боти можуть навчатися з часом, аналізуючи історію взаємодії з користувачем;

б) за функціональним призначенням:
   - інформаційні боти – надають користувачам інформацію (наприклад, новини, погоду, розклад);
   - обслуговуючі боти – допомагають у вирішенні задач, таких як оформлення замовлення, реєстрація, технічна підтримка;
   - розважальні боти – створені для розваг, ігор або спілкування без чітко визначеної мети;

в) за середовищем використання:

- вбудовані в месенджери – Telegram, Facebook Messenger, Viber тощо;
- веб-боти – інтегруються у вебсайти як частина клієнтського обслуговування;
- голосові боти – взаємодіють із користувачами через голосові інтерфейси (наприклад, Alexa, Google Assistant, Siri).

Завдяки своїй універсальності чат-боти стали потужним інструментом автоматизації комунікації та підвищення ефективності обслуговування користувачів. У наступних підрозділах буде розглянуто історію розвитку чат-ботів, технології, що лежать в їх основі, та сучасні підходи до їх реалізації.

1.2 Архітектура та принципи роботи чат-ботів

Архітектура чат-бота визначає його внутрішню структуру, компоненти та принципи взаємодії між ними. Вона може змінюватися залежно від складності та призначення системи, однак більшість сучасних чат-ботів побудовані за схожою схемою.

Типова архітектура чат-бота включає такі основні компоненти:

- інтерфейс користувача (User Interface) – середовище, через яке користувач взаємодіє з ботом. Це може бути месенджер (Telegram, Facebook Messenger), веб-сайт, мобільний додаток або голосовий інтерфейс;
- сервер додатку (Application Server) – логічний центр системи, який відповідає за обробку запитів, виклик зовнішніх API, керування діалогами та логікою відповіді;
- модуль обробки природної мови (Natural Language Processing, NLP) – компонент, що відповідає за аналіз тексту, який ввів користувач. Цей модуль розпізнає наміри (intent) та витягує сутності (entities) з

повідомлення. Для цього часто використовуються бібліотеки або сервіси, як-от Dialogflow, Rasa, Microsoft LUIS або spaCy;

- база знань / система керування контентом – джерело інформації, з якого чат-бот черпає дані для формування відповідей. Це можуть бути бази даних, API зовнішніх сервісів або власні сценарії;
- менеджер діалогу (Dialogue Manager) – модуль, який визначає, як має розвиватися розмова з користувачем. Він зберігає контекст, обирає шаблони відповідей та управляє переходами між станами діалогу;
- модуль генерації відповіді – формує підсумкове повідомлення для користувача, використовуючи шаблони або засоби генерації природної мови (NLG – Natural Language Generation).

Якщо узагальнити інформацію, яка подана в джерелах [1,2], архітектуру можна представити наступним чином (рис. 1.1).

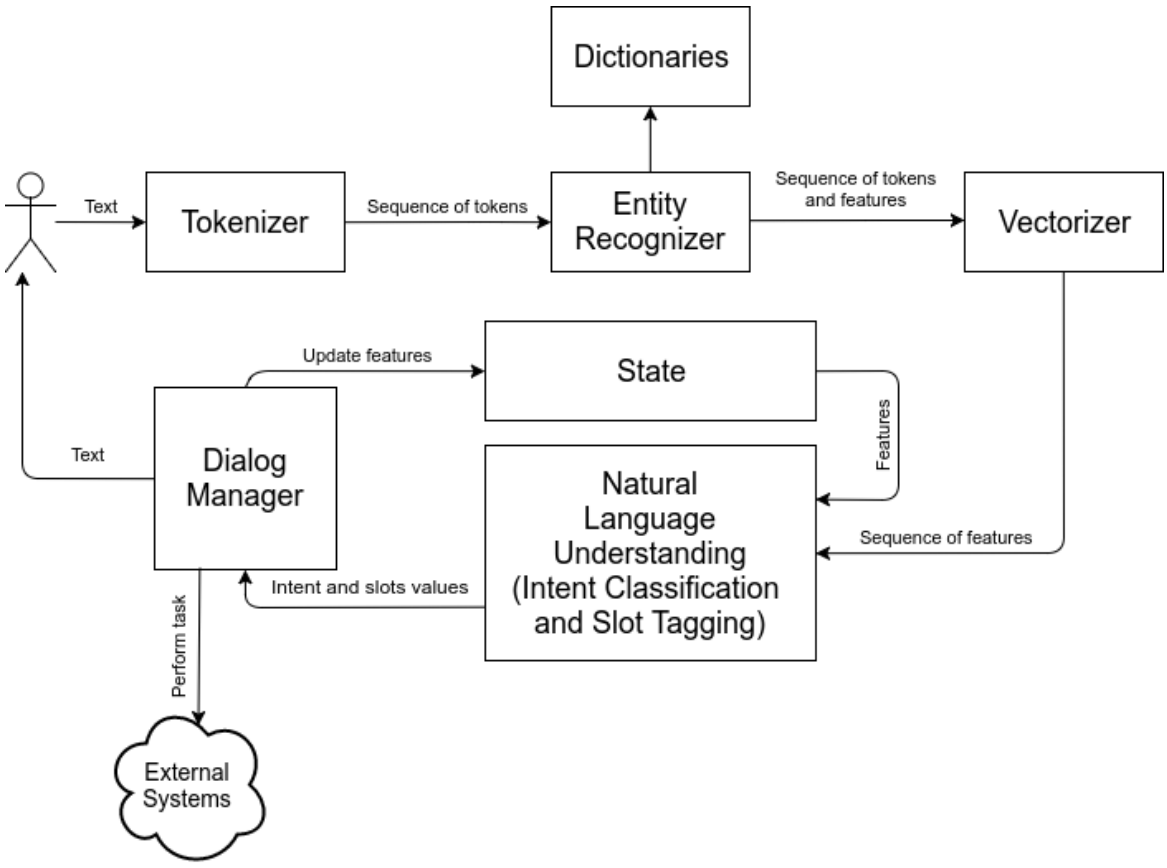


Рисунок 1.1 – Архітектура чат-бота з використанням інтелектуальної обробки природної мови

Принцип роботи чат-бота зазвичай виглядає наступним чином:

- користувач надсилає повідомлення через інтерфейс;
- повідомлення передається на сервер додатку;
- сервер передає текст в NLP-модуль, де аналізуються наміри користувача;
- визначений намір та витягнуті дані передаються в діалоговий менеджер;
- діалоговий менеджер обирає відповідну дію (наприклад, звернення до бази даних або формування відповіді);
- система генерує відповідь і надсилає її користувачу через той самий інтерфейс.

1.3 Аналіз бібліотек Python для створення чат-ботів

Огляд бібліотек та технологій для створення чат-ботів на Python показує різноманітність інструментів, доступних розробникам. Для обробки природної мови (NLP) часто використовують такі бібліотеки, як NLTK та SpaCy, які надають функціональність для токенізації, визначення частин мови, розпізнавання іменованих сутностей та аналізу синтаксису. Бібліотека TextBlob також є популярною завдяки своїй простоті та можливостям аналізу тексту.

Для створення самого чат-бота існують різні фреймворки. ChatterBot є бібліотекою, яка дозволяє створювати чат-ботів, що можуть навчатися на основі розмов. Rasa є більш комплексним фреймворком, який надає інструменти для побудови просунутих чат-ботів з використанням машинного навчання для розуміння намірів користувача та керування діалогом.

Для інтеграції чат-ботів з веб-додатками часто використовують веб-фреймворки, такі як Flask та Django, які дозволяють створювати API для взаємодії чат-бота з зовнішніми системами. Для підключення до конкретних

месенджерів існують спеціалізовані бібліотеки, наприклад, python-telegram-bot для Telegram та discord.py для Discord.

Крім того, для більш складних завдань, таких як глибоке навчання в NLP, можуть використовуватися бібліотеки машинного навчання, такі як TensorFlow та PyTorch. Також існують хмарні сервіси, такі як Dialogflow від Google, IBM Watson Assistant та Azure Bot Service, які надають готові NLP-моделі та інфраструктуру для розгортання чат-ботів.

Кожен інструмент має свої сильні та слабкі сторони, орієнтовані на різні потреби та рівні складності проєктів (таблиця 1.1).

Таблиця 1.1 – Результати порівняльного аналізу бібліотек Python для створення чатботів

Характеристика	NLTK	SpaCy	TextBlob	ChatterBot	Rasa
Основне призначення	Навчання та дослідження в NLP	Швидка та ефективна обробка NLP у виробництві	Простий інтерфейс для базових задач NLP	Створення чат-ботів, що навчаються на розмовах	Створення складних чат-ботів з ML для розуміння
Швидкість обробки	Повільніша	Швидша, оптимізована	Середня	Залежить від розміру бази знань	Залежить від складності моделі
Простота використання	Потрібне розуміння основ NLP	Відносно просте API	Дуже просте API	Просте API для базових ботів	Потребує розуміння ML концепцій
Можливості NLP	Широкий набір алгоритмів та ресурсів	Токенізація, лематизація, NER, POS, залежності	Токенізація, POS, іменні сутності, аналіз настрою	Базова обробка тексту для навчання	Розширене розуміння намірів та сутностей, керування діалогом
Навчання	Вручну або використання існуючих корпусів	Попередньо навчені моделі, можливість донавчання	Використовує NLTK/SpaCy для базових задач	Навчання на основі розмов	Навчання моделей машинного навчання
Керування діалогом	Відсутнє	Відсутнє	Відсутнє	Базове	Потужне керування контекстом та потоком діалогу

Продовження таблиці 1.1

Характеристика	NLTK	SpaCy	TextBlob	ChatterBot	Rasa
Інтеграція	Базова	Легка інтеграція з іншими бібліотеками Python	Легка інтеграція з іншими бібліотеками Python	Легка інтеграція з Python-проектами	Інтеграція з веб-фреймворками та месенджерами
Масштабованість	Залежить від реалізації	Залежить від реалізації	Залежить від реалізації	Залежить від реалізації	Добре підходить для великих проєктів
Вартість	Безкоштовно (open-source)	Безкоштовно (open-source)	Безкоштовно (open-source)	Безкоштовно (open-source)	Безкоштовно (open-source)
Складність	Середня до високої	Низька до середньої	Низька	Низька до середньої	Висока
Випадки використання	Академічні дослідження, глибокий аналіз тексту	Виробничі NLP-задачі, швидка обробка тексту	Швидкі прототипи, базові NLP-задачі	Прості чат-боти для FAQ та базової підтримки	Інтелектуальні чат-боти з розумінням намірів

NLTK та SpaCy є потужними бібліотеками для обробки природної мови. NLTK відрізняється широким набором алгоритмів та ресурсів, що робить її чудовим вибором для академічних досліджень та глибокого розуміння NLP. Однак, її використання може бути менш інтуїтивним для швидкої розробки. SpaCy, навпаки, орієнтована на швидкість та ефективність у виробничих умовах, пропонуючи оптимізовані моделі та більш сучасний API. TextBlob є простішою обгорткою над NLTK та SpaCy, що робить її ідеальною для швидких прототипів та базових завдань NLP.

Щодо фреймворків для створення чат-ботів, ChatterBot вирізняється своєю здатністю до навчання на основі розмов, що спрощує створення простих діалогових систем. Однак, його можливості для складного керування діалогом та розуміння намірів користувача є обмеженими. Rasa є більш комплексним рішенням, яке використовує машинне навчання для розуміння природної мови та керування контекстом діалогу. Це робить її потужним інструментом для створення інтелектуальних чат-ботів, але вимагає глибшого розуміння машинного навчання та більших зусиль на початковому етапі.

Для інтеграції з веб-додатками, Flask та Django надають гнучкі можливості для створення API, але вибір між ними залежить від масштабу та складності веб-частини проєкту. Flask є легким та мінімалістичним, що підходить для невеликих API, тоді як Django є повноцінним фреймворком з великою кількістю вбудованих функцій, що краще підходить для складних веб-додатків. Бібліотеки, такі як python-telegram-bot та discord.py, спрощують інтеграцію з конкретними платформами обміну повідомленнями, надаючи зручні API для надсилання та отримання повідомлень.

Для завдань, що вимагають глибокого навчання, TensorFlow та PyTorch є провідними бібліотеками, які надають потужні інструменти для побудови та навчання складних нейронних мереж. Однак, їх використання вимагає значних знань у галузі машинного навчання. Хмарні сервіси, такі як Dialogflow, IBM Watson Assistant та Azure Bot Service, пропонують готові NLP-моделі та інфраструктуру, що значно спрощує процес створення та розгортання чат-ботів, особливо для тих, хто не має глибоких знань у машинному навчанні або потребує швидкого розгортання.

Підсумовуючи, вибір технологій залежить від конкретних вимог проєкту. Для простих чат-ботів з базовою обробкою мови підійдуть NLTK або TextBlob у поєднанні з ChatterBot. Для більш складних сценаріїв з глибоким розумінням намірів користувача та керуванням діалогом кращим вибором буде Rasa або хмарні сервіси. Для інтеграції з веб-додатками необхідно використовувати відповідні веб-фреймворки, а для підключення до месенджерів – спеціалізовані бібліотеки. У випадках, коли потрібне глибоке навчання, TensorFlow або PyTorch стануть незамінними інструментами.

1.4 Застосування чат-ботів у різних сферах

Застосування чат-ботів охоплює широкий спектр сфер, демонструючи їхню універсальність та здатність оптимізувати різноманітні процеси.

Найпоширеніша сфера застосування -це обслуговування клієнтів. Чат-боти можуть надавати миттєву підтримку клієнтам 24/7, відповідати на поширені запитання, допомагати з навігацією по сайту, обробляти прості запити та скеровувати складніші питання до операторів-людей. Це підвищує задоволеність клієнтів, зменшує навантаження на службу підтримки та знижує витрати.

Чат-боти можуть використовуватися для залучення потенційних клієнтів, збору лідів, надання інформації про продукти та послуги, персоналізації пропозицій, допомоги у виборі товарів, оформленні замовлень та навіть для проведення маркетингових кампаній. Вони можуть реактивно відповідати на запитання користувачів, спонукаючи їх до покупки.

У корпоративному середовищі чат-боти можуть автоматизувати рутинні завдання HR-відділу, такі як відповіді на запитання співробітників щодо політик компанії, надання інформації про пільги, допомога в оформленні документів, збір зворотного зв’язку та організація навчальних процесів. Вони також можуть використовуватися для внутрішніх комунікацій, розповсюдження новин та організації подій.

Чат-боти можуть виступати в ролі віртуальних асистентів для студентів, надаючи інформацію про курси, розклади, дедлайни, відповідаючи на навчальні запитання, надаючи персоналізовані рекомендації щодо навчання та навіть допомагаючи з підготовкою до іспитів. Для викладачів вони можуть автоматизувати перевірку простих завдань та надавати статистику успішності студентів.

У сфері охорони здоров’я чат-боти можуть використовуватися для запису на прийом до лікаря, надання інформації про симптоми та захворювання (з застереженням щодо самодіагностики), нагадування про прийом ліків, моніторингу стану пацієнтів (через інтеграцію з носимими пристроями) та надання психологічної підтримки.

Чат-боти можуть допомагати клієнтам банків та фінансових установ отримувати інформацію про свої рахунки, здійснювати прості транзакції,

отримувати консультації щодо інвестицій, подавати заявки на кредити та отримувати персоналізовані фінансові поради.

Чат-боти можуть також допомагати туристам у бронюванні готелів та авіаквитків, надавати інформацію про визначні місця, складати маршрути подорожей, відповідати на питання про візові вимоги та надавати підтримку під час подорожі.

Чат-боти можуть надавати громадянам інформацію про державні послуги, допомагати в оформленні документів, відповідати на поширені запитання щодо законодавства та надавати консультації з різних державних питань.

Вони широко використовуються в логістиці, можуть відстежувати статус доставки, надавати інформацію про розклади руху транспорту, допомагати з оформленням замовлень на перевезення та відповідати на запитання щодо логістичних процесів.

З розвитком технологій штучного інтелекту їхня роль у різних галузях буде лише зростати, відкриваючи нові можливості для автоматизації, покращення обслуговування та оптимізації бізнес-процесів.

2 ІНСТРУМЕНТИ ТА СЕРЕДОВИЩЕ РОЗРОБКИ ЧАТ-БОТА

2.1 Обгрунтування вибору фреймворку Rasa

Фреймворк Rasa є одним із провідних інструментів для розробки інтелектуальних чат-ботів, орієнтованих на використання технологій обробки природної мови (NLP). Його вибір обумовлено низкою наступних технічних, функціональних та практичних переваг.

Rasa є фреймворком з відкритим кодом, що надає повний контроль над логікою бота, даними та процесами машинного навчання. Це особливо важливо для застосувань, де необхідна гнучкість, кастомізація або інтеграція зі специфічними системами.

Фреймворк підтримує:

- розпізнавання намірів (intent recognition);
- витяг сутностей (entity extraction);
- діалогові моделі з використанням ML-політик (machine learning policies), які дозволяють боту адаптуватися до складних сценаріїв спілкування.

Rasa дозволяє розгортати рішення локально або на власних серверах, що гарантує безпеку та конфіденційність даних користувачів – критично важливу вимогу для багатьох галузей (медицина, фінанси, держустанови тощо).

Архітектура фреймворку містить наступні частини:

- Rasa NLU – це інструмент з відкритим кодом для обробки природної мови, призначений для класифікації намірів (визначає, що саме хоче користувач), витягу сутностей у вигляді структурованих даних, а також допомагає чат-боту розуміти, що говорить користувач;
- Rasa Core – це фреймворк для створення чат-ботів з управлінням діалогом на основі машинного навчання. Він приймає структуровані дані, отримані від Rasa NLU, і прогнозує наступну найкращу дію,

використовуючи ймовірнісну модель, таку як нейронна мережа LSTM, замість традиційних умовних операторів (if/else). У своїй основі Rasa Core також застосовує методи навчання з підкріпленням для покращення передбачення наступної дії (рис.2.1.). Це забезпечує модульність і легкість масштабування проєкту.

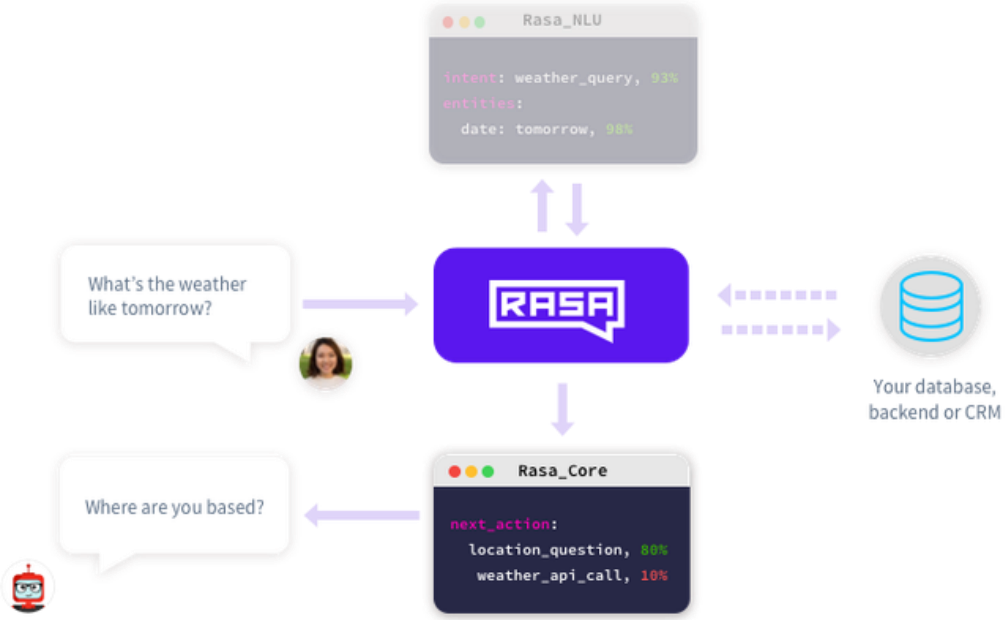


Рисунок 2.1 – Архітектура Rasa

Rasa має активну спільноту розробників, численні приклади, шаблони, а також офіційну документацію, що спрощує навчання та впровадження.

Фреймворк легко інтегрується з різними каналами комунікації (Telegram, Slack, Facebook Messenger тощо), а також із зовнішніми сервісами та API, що дозволяє будувати повноцінні бізнес-рішення.

2.2 Можливості інтеграції чат-бота на Rasa з месенджерами

Фреймворк Rasa надає широкі можливості для інтеграції чат-ботів із популярними платформами обміну повідомленнями, що дозволяє забезпечити

зручну та ефективну взаємодію з користувачами на звичних для них каналах комунікації.

Інтеграція з Telegram є однією з найпоширеніших завдяки простоті реалізації: бот підключається через Telegram Bot API, підтримує як текстові, так і мультимедійні повідомлення, а також надає можливість використання клавіатур, кнопок, команд і меню.

Інтеграція з Facebook Messenger дозволяє розгортати бота на сторінках компаній із підтримкою розширених можливостей, таких як шаблони повідомлень, quick replies, каруселі; також забезпечується авторизація користувачів через Facebook профілі, що сприяє високій інтерактивності та широкому охопленню аудиторії.

Інтеграція з WhatsApp реалізується за допомогою офіційного API через провайдерів, таких як Twilio, Vonage або Meta Cloud API, що дає змогу надсилати повідомлення користувачам у реальному часі, використовувати шаблонні повідомлення та інтерактивні елементи, а також широко застосовується бізнесом для підтримки клієнтів.

Серед основних переваг інтеграції чат-ботів із месенджерами можна виокремити охоплення великої кількості користувачів на зручних для них платформах, миттєвий обмін повідомленнями та підтримку в реальному часі, покращення користувацького досвіду завдяки інтерактивним елементам і швидким відповідям, а також легку масштабованість без необхідності створення окремих застосунків для кожної платформи.

2.3 Використання штучного інтелекту та обробки природної мови (NLP) у чат-ботах на Rasa

Фреймворк Rasa забезпечує потужну підтримку технологій штучного інтелекту (ШІ) та обробки природної мови (Natural Language Processing, NLP),

що дозволяє створювати інтелектуальні чат-боти, здатні розуміти людську мову, адаптуватися до контексту та вести осмислений діалог з користувачем.

Основна мета NLP у чат-ботах – навчити бота розпізнавати, що саме хоче сказати користувач, і правильно реагувати на запити у природній формі. Завдяки NLP бот може розпізнавати наміри користувача (intents), виділяти з повідомлення сутності (entities), наприклад дати, імена, локації чи інші параметри, а також будувати логіку розмови залежно від контексту.

Фреймворк Rasa складається з двох основних компонентів:

- Rasa NLU (Natural Language Understanding) – відповідає за обробку вхідного повідомлення користувача. За допомогою алгоритмів машинного навчання Rasa NLU визначає намір (наприклад, «замовити їжу», «записатися до лікаря») і витягує сутності, тобто важливі частини тексту, які допомагають зрозуміти деталі запиту (наприклад, дату або категорію);
- Rasa Core – відповідає за керування діалогом. Вона приймає структуровану інформацію від NLU і, на основі історії діалогу, визначає, яку дію має виконати бот далі. Замість жорстко заданих if-else конструкцій, Rasa Core використовує моделі машинного навчання (наприклад, LSTM, трансформери або політики з підкріпленням), які навчаються на прикладах діалогів і здатні обирати оптимальну відповідь навіть у складних сценаріях.

Завдяки використанню ШІ та NLP у Rasa, чат-боти:

- можуть розуміти різні формулювання одних і тих самих запитів, не обмежуючись чітко визначеними фразами;
- мають гнучку логіку поведінки, яка змінюється залежно від контексту та попередніх реплік;
- підтримують багатомовність, включаючи українську мову;
- здатні навчатися на власному досвіді: із кожною взаємодією бот може вдосконалювати свої відповіді;

– дозволяють розробникам швидко змінювати та розширювати функціональність, не змінюючи всієї структури коду.

Розробка NLP-модуля в Rasa передбачає створення навчального датасету у форматі nlu.yml, де вказуються приклади фраз користувачів з відповідними намірами та сутностями. Далі цей модуль тренується за допомогою вбудованих або користувацьких моделей машинного навчання. Потім створюється файл stories.yml або rules.yml, де описується очікувана поведінка бота при певному сценарії розмови. Кінцевий результат – адаптивний чат-бот, який не просто відповідає за шаблоном, а веде логічний діалог, розуміючи користувача.

2.4 Обгрунтування вибору середовища розробки

Для розробки інтелектуального чат-бота з використанням фреймворку Rasa доцільно обрати середовище програмування, яке забезпечує зручну роботу з Python, підтримку роботи з конфігураційними файлами та засоби відлагодження. Саме таким середовищем є PyCharm, інтегроване середовище розробки (IDE), створене компанією JetBrains спеціально для мови Python.

Основні переваги використання PyCharm у проєктах на Rasa наступні:

– повна підтримка Python. PyCharm надає підсвічування синтаксису, автодоповнення коду, рефакторинг і контроль помилок у реальному часі. Оскільки Rasa повністю написана на Python, це значно полегшує написання і тестування коду;

– підтримка YAML, JSON та Markdown. Rasa широко використовує файли у форматах YAML (nlu.yml, rules.yml, domain.yml тощо), а також Markdown (stories.md). PyCharm розпізнає ці формати, надає автодоповнення ключових слів, виділення структури та виявлення синтаксичних помилок, що дозволяє уникнути критичних помилок у конфігурації;

- вбудований термінал і управління середовищами. PyCharm дозволяє запускати команди Rasa (наприклад, rasa train, rasa shell, rasa run) безпосередньо з IDE через вбудований термінал. Також можна легко інтегрувати віртуальні середовища (venv, conda), що забезпечує ізольовану та контрольовану розробку.
- інструменти налагодження та тестування. PyCharm містить потужний відлагоджувач (debugger), який дозволяє покроково переглядати виконання програм, встановлювати точки зупину, перевіряти значення змінних, що особливо корисно при створенні користувацьких дій (custom actions) у Rasa;
- інтеграція з системами контролю версій. Для командної роботи над проєктом надзвичайно важливою є інтеграція з Git. PyCharm має зручний інтерфейс для роботи з Git, що дозволяє ефективно керувати репозиторієм, виконувати злиття гілок, переглядати зміни та історію комітів;
- гнучкість і розширюваність. PyCharm підтримує плагіни для інтеграції з базами даних, API, Docker, SSH, Jupyter Notebook тощо. Це дозволяє розширити функціональність середовища розробки відповідно до потреб проєкту.

Таким чином, вибір PyCharm як середовища розробки для створення чат-бота на Rasa забезпечує ефективність, зручність та високу продуктивність розробки, дозволяє уникати типових помилок та зосередитися на логіці бота й покращенні його інтелектуальної поведінки.

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА ЧАТ-БОТА

3.1 Постановка задачі

Метою проєкту є розробка інтелектуального чат-бота на базі фреймворку Rasa, який зможе спілкуватися з користувачами українською мовою на тему українських національних традицій, звичаїв, свят, обрядів і культурних особливостей. Бот має бути здатним інформувати, відповідати на запитання, вести прості діалоги, а також надавати освітні й довідкові дані в зручній формі.

Функціональні вимоги:

- підтримка розуміння природної мови (NLP) – бот повинен розпізнавати наміри користувачів (наприклад: «розкажи про Різдво», «які є народні свята», «що таке колядки») і витягувати ключові сутності (наприклад: назва свята, місяць, дія);
- база знань про традиції – чат-бот має містити інформацію про народні свята (Різдво, Великдень, Івана Купала, Покрова тощо), обряди та звичаї (колядування, щедрування, весільні традиції), національний одяг, символіку, ремесла, страви української кухні, пов’язані зі святами;
- інтерактивне спілкування – бот повинен вести діалог у дружній формі, ставити уточнювальні питання при потребі, реагувати на некоректні запити ввічливо та коректно;
- мовна підтримка – бот має працювати повністю українською мовою, з урахуванням граматичних особливостей;
- можливість інтеграції з месенджерами – передбачена можливість підключення до Telegram або Facebook Messenger для зручності спілкування з широкою аудиторією;
- масштабованість – архітектура бота повинна дозволяти легко додавати нові теми, традиції або відповіді без необхідності значних змін у коді.

Нефункціональні вимоги:

– зручний інтерфейс для взаємодії з користувачем;
– висока точність розпізнавання намірів та контексту;
– швидкий час реакції та відповіді;
– можливість локального або хмарного розгортання.

3.2 Настроювання середовища розробки

Для початку розробки було завантажено та встановлено середовище PyCharm Community Edition з офіційного сайту JetBrains, після чого створено новий проєкт для чат-бота. Під час його створення було обрано інтерпретатор Python 3.11 та створено ізольоване віртуальне середовище venv для керування залежностями.

Далі через вбудований термінал оновлено pip і встановлено бібліотеку Rasa. Після цього ініціалізовано новий проєкт Rasa, що автоматично сформувало необхідну структуру з конфігураційними файлами (рис. 3.1), та виконано початкове тренування моделі.

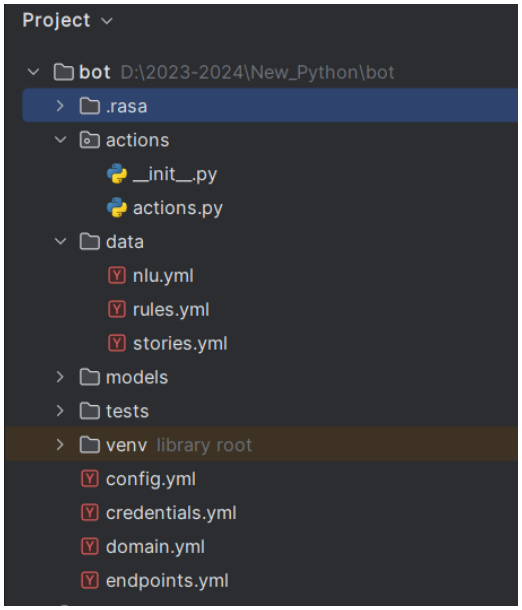


Рисунок 3.1 – Структура проєкту

У середовищі PyCharm було налаштовано зручну структуру проєкту: папку actions/ позначено як джерело коду для кращої навігації, а також створено файл actions.py для реалізації власних дій. Модель чат-бота було навчено на базі наданих прикладів за допомогою команди rasa train. Після цього бот був запущений у консольному режимі за допомогою команди rasa shell і протестований через інтерактивну взаємодію, окремо було запущено сервер дій (rasa run actions) для обробки відповідей користувача. Для перевірки логіки виконання кастомних дій використано вбудовані інструменти налагодження в PyCharm, зокрема встановлено точки зупину у файлі actions.py для покрокового аналізу.

3.3 Опис основних файлів

В цьому розділі описані основні файли проєкту.

Файл config.yml містить налаштування, що відповідають за обробку природної мови та управління діалогом у проєкті Rasa. У ньому зазначено версію конфігурації – 3.1, що вказує на використання стабільного релізу, а також визначено мову обробки – українську (uk).

Центральним елементом файлу є pipeline – послідовність компонентів, що забезпечують аналіз вхідних повідомлень. Текст розбивається на токени за допомогою WhitespaceTokenizer, а структура речень уточнюється за допомогою LexicalSyntacticFeaturizer. Векторні представлення створюються CountVectorsFeaturizer, у тому числі з аналізом на рівні символів за допомогою параметрів analyzer: char_wb та діапазону n-грамів від 1 до 4.

Для класифікації намірів і витягування сутностей використовується DIETClassifier, який навчається впродовж 100 епох. Важливою є також підтримка синонімів для сутностей, яку забезпечує EntitySynonymMapper. Компонент ResponseSelector дозволяє обрати найбільш релевантну відповідь у контексті діалогу, також тренується 100 епох.

Окрему роль у конфігурації відіграють політики, які визначають поведінку чат-бота під час взаємодії з користувачем. MemoizationPolicy дозволяє боту запам’ятовувати успішні сценарії з минулого досвіду. RulePolicy дає змогу задавати чіткі правила, які слід виконувати у певних умовах. TEDPolicy використовує глибоке навчання для визначення наступної дії, спираючись на останні 5 кроків діалогу, і також тренується протягом 100 епох. Параметр constrain_similarities, встановлений у значення true, обмежує вибір надто схожих дій, підвищуючи різноманітність реакцій.

Таким чином, цей файл відіграє ключову роль у формуванні логіки поведінки бота, забезпечуючи адаптивність, точність аналізу та ефективність взаємодії з користувачем, а також оптимізує роботу системи для української мови. Код конфігурації подано в додатку А.

Файл domain.yml є одним із ключових елементів проєкту Rasa, адже саме він визначає обсяг знань бота: у ньому описуються наміри користувача, сутності, слоти, відповіді, дії та інші компоненти, необхідні для ведення діалогу.

Наприклад, наміри (intents) відображають типові цілі користувача під час спілкування з ботом – вітання, загальні запити про українські традиції або інтерес до конкретної традиції, що передбачає використання відповідної сутності.

Сутності (entities) допомагають надати додатковий зміст висловлюванням користувача, зокрема через витяг назв конкретних традицій. Ці дані зберігаються у слотах (slots), які виконують роль змінних і можуть бути використані у відповідях бота, як-от слот tradition_name, що автоматично заповнюється на основі витягнутої сутності та застосовується у відповідних повідомленнях.

Відповіді (responses), заздалегідь визначені у файлі, використовуються ботом для реакції на ті чи інші наміри користувача або як частина діалогової стратегії. Серед них є як статичні репліки (наприклад, вітання), так і динамічні, що формуються з урахуванням значення слота.

У випадку незрозумілих чи невизначених запитів передбачено загальні повідомлення, що сигналізують про неможливість обробки звернення.

Окрім стандартних реакцій, у файлі також визначаються дії (actions), зокрема кастомні, реалізовані у файлі actions.py. Наприклад, дія action_розповісти_про_традицію може перевірити правильність назви традиції, здійснити пошук відповідної інформації у базі знань або файлі, а також передати згенеровану відповідь користувачу через метод dispatcher.utter_message(). Код конфігурації наведено в додатку Б.

Файл nlu.yml – це файл, який містить навчальні дані для компонента Rasa NLU (Natural Language Understanding). Код представлений в додатку В. Він визначає приклади фраз користувача для кожного наміру (intent) і, за потреби, витягування сутностей (entity).

Rasa використовує ці приклади для того, щоб навчитися розпізнавати, коли користувач вітається, прощається, цікавиться загальними або конкретними традиціями в спілкуванні з ботом. У цих прикладах вказано не лише намір, але й сутність назва_традиції, яку Rasa має витягнути для подальшої обробки (наприклад, щоб передати в слот tradition_name).

Файл rules.yml визначає чіткі правила поведінки чат-бота в конкретних ситуаціях. Повний код представлений в додатку Г. Він використовує RulePolicy для встановлення простих, передбачуваних реакцій на певні наміри користувача.

Якщо користувач вітається, бот відповідає заздалегідь визначеним повідомленням utter_привітання. Для загального запиту бот надає загальний список українських традицій. Коли бот не може класифікувати фразу або коли користувач не зрозумів відповідь, надсилається відповідь для уточнення.

Цей файл дозволяє зробити бот більш стабільним і передбачуваним, використовуючи фіксовані сценарії.

Файл stories.yml містить навчальні сценарії, які визначають можливі шляхи розвитку діалогу між користувачем і ботом. Він моделює різноманітні варіанти розмов, дозволяючи системі ефективно реагувати на запити відповідно до навченої послідовності дій. Усі сценарії задаються в структурованому вигляді, що дозволяє чітко відслідковувати логіку спілкування.

Якщо необхідно реалізувати функціональність, яка не підтримується стандартними засобами Rasa, така логіка прописується вручну у файлі actions.py. У цьому файлі імпортуються ключові компоненти Rasa, зокрема клас Action для створення кастомних дій, CollectingDispatcher для надсилання відповідей користувачу, Tracker для доступу до контексту діалогу. Хоча компонент SlotSet у цьому випадку не використовується, він може бути корисним у сценаріях, де потрібно оновлювати значення слотів.

У межах реалізації створюється словник, що виконує роль простої бази знань, де зберігаються описи українських традицій. Далі створюється клас, який реалізує дію для реагування на запит користувача про певну традицію. Назва цієї дії повинна відповідати тій, що зазначена у файлі domain.yml. Зі слота отримується назва традиції, після чого бот перевіряє наявність відповідної інформації в словнику. Якщо відповідь знайдено – вона повертається користувачу, інакше бот інформує, що не володіє такими даними.

Код сценаріїв наведено в додатку Д, а реалізація дії – у додатку Е.

Для налаштування підключення до кастомного сервера дій (custom action server) у проєкті Rasa в файлі endpoint.yml вказується:

```
action_endpoint:
  url: "http://localhost:5055/webhook"
```

В результаті, після запуску сервера (rasa run action) та клієнта (rasa shell), можна створити діалог (рис. 3.2).

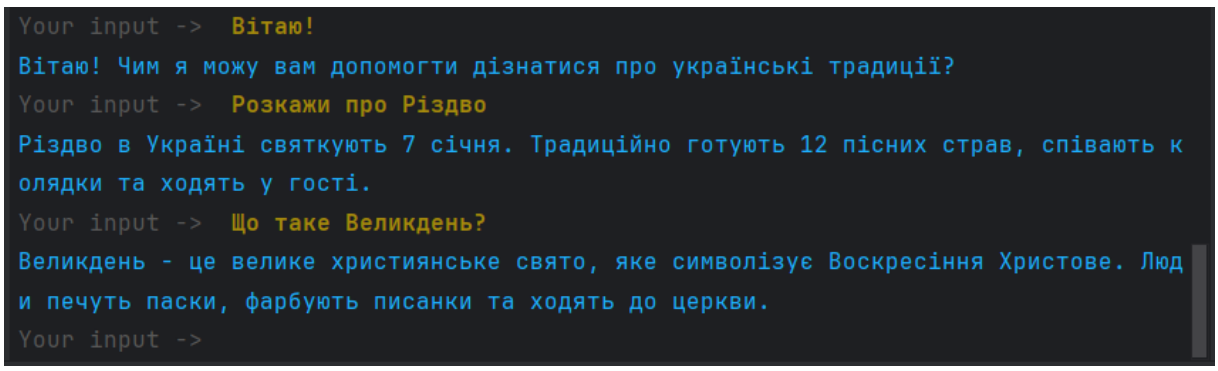


Рисунок 3.2 – Приклад діалогу

3.4 Додавання web-інтерфейсу

Веб-інтерфейс для чат-бота створюється для того, щоб надати користувачам зручний та доступний спосіб взаємодії з ботом через веб-браузер. Це особливо корисно, коли чат-бот має виконувати функції консультанта, помічника або інформатора на сайті.

Основні цілі:

- надати користувачам точку входу для спілкування з ботом;
- зробити чат-бот доступним на сайтах, порталах або в корпоративних системах;
- забезпечити зручний, візуально приємний інтерфейс без потреби встановлення окремого додатку;
- покращити взаємодію з клієнтами, надаючи підтримку 24/7.

Для створення web-інтерфейсу були виконані наступні дії.

Розроблено сторінку index.html, на якій розміщується контейнер для чату і підключається JavaScript-бібліотека для webchat (рисунок 3.3).

Файл розміщується на локальному або віддаленому сервері. Можна запустити простий сервер за допомогою Python:

```
python -m http.server 8000,
```

і відкрити в браузері за адресою:

```
http://localhost:8000/index.html
```

Rasa-сервер має бути активний (rasa run), а також активний endpoint http://localhost:5055 для кастомних дій (rasa run actions).

В результаті бот буде працювати через браузер (рис. 3.4).

```
<!DOCTYPE html>
<html lang="uk">
  <head>
    <meta charset="UTF-8">
    <title>Rasa Chatbot</title>
  </head>
  <body>
    <h1>Привіт! Я твій Rasa-бот </h1>

    <div id="webchat" />

    <script
      src="https://cdn.jsdelivr.net/npm/rasa-
webchat@1.0.1/lib/index.min.js"
      type="application/javascript"
    ></script>

    <script type="application/javascript">
      WebChat.default.init({
        selector: "#webchat",
        initPayload: "/привітання",
        customData: {"language": "uk"},
        socketUrl: "http://localhost:5005",
        title: "Бот про українські традиції",
      });
    </script>
  </body>
</html>
```

Рисунок 3.3 – Сторінка для web-інтерфейсу

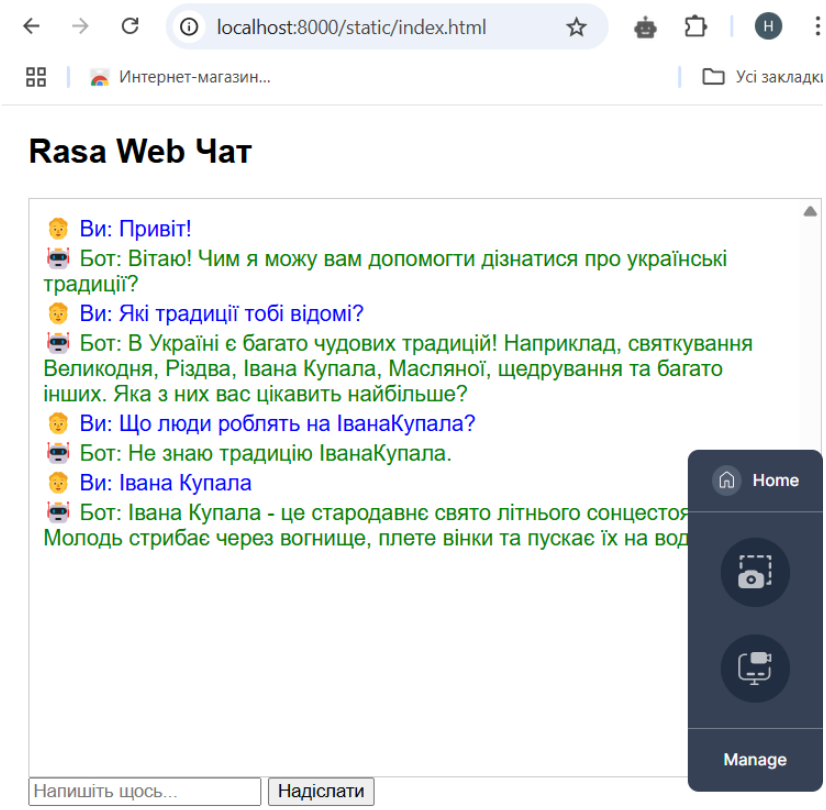


Рисунок 3.4 – Web-інтерфейс для бота

ВИСНОВКИ

У межах бакалаврської кваліфікаційної роботи було успішно реалізовано інтелектуального чат-бота, призначеного для інформування користувачів про українські традиції. Розробка здійснювалася з використанням мови програмування Python та фреймворку Rasa, що забезпечує гнучке управління діалогами, обробку природної мови та можливість навчання на базі прикладів.

У процесі дослідження було досягнуто наступних результатів:

- проведено аналіз сучасних технологій та інструментів, що застосовуються для створення чат-ботів, із акцентом на Python-бібліотеки;
- вивчено архітектуру та функціональні можливості фреймворку Rasa, включаючи компоненти NLU (розпізнавання намірів та витяг сутностей) та Core (побудова логіки діалогу);
- створено набір прикладів діалогів та базу знань, присвячену українським традиціям, зокрема святам, обрядами, звичаям і символіці;
- розроблено та протестовано чат-бота, який здатен відповідати на запити користувача у природній формі, підтримувати кілька тем розмови та надавати корисну інформацію;
- проведено тестування функціоналу чат-бота, яке підтвердило його працездатність та релевантність відповідей.

Результати роботи засвідчують доцільність використання фреймворку Rasa для створення тематичних інтелектуальних чат-ботів, а також демонструють ефективність Python як основного інструменту для розробки. Розроблений бот може бути використаний як освітній або культурно-просвітницький інструмент для популяризації українських традицій серед молоді та ширшого кола користувачів.

У перспективі можливе подальше розширення функціональності чат-бота: додавання мультимедійного контенту, інтеграція з месенджерами, розширення бази знань і підтримка діалогів на кількох мовах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Jurafsky D., Martin J. H. Speech and Language Processing. 3rd ed. Draft. Stanford University, 2023. 1024 p.

2. Rasa Technologies. Introduction to Rasa Open Source. Rasa Documentation. URL : https://rasa.com/docs/rasa/ (дата звернення : 10.05.2025).

3. Chollet F. Deep Learning with Python. 2nd ed. New York : Manning Publications, 2021. 504 p.

4. Brownlee J. How to Develop a Chatbot with Deep Learning. *Machine Learning Mastery*. URL : https://machinelearningmastery.com/how-to-develop-a-chatbot-with-deep-learning/ (дата звернення : 08.05.2025).

5. Воронін В. І., Гнатюк С. М. Основи штучного інтелекту. Київ : КНЕУ, 2020. 312 с.

6. Місюра А. В., Гуменюк І. М. Застосування чат-ботів в освітньому процесі. *Наукові записки. Серія: Педагогіка*. 2021. Вип. 193. С. 132–138.

7. Python Software Foundation. *The Python Language Reference*. Python.org. URL : https://docs.python.org/3/reference/ (дата звернення : 09.05.2025).

8. Овчаренко О. Ю. Розробка чат-бота для Telegram із використанням Python. *Комп'ютерні науки та інформаційні технології*. 2022. № 30. С. 45–50.

9. Devlin J. et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint*. URL : https://arxiv.org/abs/1810.04805 (дата звернення : 10.05.2025).

10. Stack Overflow. How to use Rasa with custom actions. *StackOverflow Q&A*. URL : https://stackoverflow.com/questions/56972629/how-to-use-rasa-with-custom-actions (дата звернення : 07.05.2025).

11. Мельник Р. С. Технології штучного інтелекту в сучасному програмному забезпеченні. Львів : Видавництво ЛНУ, 2019. 228 с.

ДОДАТОК А

Код файлу config.yml

```
version: "3.1"
language: uk

pipeline:
- name: WhitespaceTokenizer
- name: LexicalSyntacticFeaturizer
- name: CountVectorsFeaturizer
- name: CountVectorsFeaturizer
  analyzer: char_wb
  min_ngram: 1
  max_ngram: 4
- name: DIETClassifier
  epochs: 100
- name: EntitySynonymMapper
- name: ResponseSelector
  epochs: 100

policies:
- name: MemoizationPolicy
- name: RulePolicy
- name: TEDPolicy
  max_history: 5
  epochs: 100
  constrain_similarities: true
assistant_id: 20250416-125814-ornery-commit
```

ДОДАТОК Б

Код файлу domain.yml

```
version: "3.1"
intents:
  - привітання
  - прощання
  - запитати_традицію
  - запитати_конкретну_традицію
  - подяка
  - не_розумію
entities:
  - назва_традиції
slots:
  tradition_name: # Змінено назву слота
    type: text
    influence_conversation: true
    mappings:
    - type: from_entity
      entity: назва_традиції

responses:
  utter_привітання:
  - text: "Вітаю! Чим я можу вам допомогти дізнатися про українські
традиції?"

  utter_прощання:
  - text: "До побачення! Завжди радий розповісти вам щось цікаве про
Україну."

  utter_запитати_традицію:
  - text: "В Україні є багато чудових традицій! Наприклад, святкування
Великодня, Різдва, Івана Купала, Масляної, щедрування та багато інших.
Яка з них вас цікавить найбільше?"

  utter_розповісти_традицію:
  - text: "З радістю розповім вам про традицію {tradition_name}..." #
Оновлено назву слота

  utter_не_знаю_традицію:
  - text: "Вибачте, я поки що не маю інформації про традицію
‘{tradition_name}’. Можливо, ви мали на увазі щось інше?" # Оновлено
назву слота

  utter_подяка:
  - text: "Будь ласка! Завжди радий допомогти."

  utter_не_розумію:
  - text: "Вибачте, я не зовсім зрозумів ваше питання. Спробуйте
сформулювати його інакше."

actions:
  - action_розповісти_про_традицію
```

ДОДАТОК В

Код файлу nlu.yml

```
version: "3.1"
nlu:
- intent: привітання
  examples: |
    - привіт
    - добрий день
    - вітаю
    - здрастуйте
    - доброго ранку
    - доброго вечора

- intent: прощання
  examples: |
    - до побачення
    - бувай
    - на все добре
    - до зустрічі
    - па-па

- intent: запитати_традицію
  examples: |
    - розкажи про українські традиції
    - які є традиції в Україні?
    - що цікавого в українських звичаях?
    - поділись якоюсь українською традицією
    - яка найцікавіша українська традиція?
    - що святкують в Україні?

- intent: запитати_конкретну_традицію
  examples: |
    - розкажи про [Великдень](назва_традиції)
    - що ти знаєш про [Різдво](назва_традиції)?
    - як святкують [Івана Купала](назва_традиції)?
    - що таке [Масляна](назва_традиції)?
    - розкажи про традицію [щедрування](назва_традиції)
    - що роблять на [Святого Миколая](назва_традиції)?

- intent: подяка
  examples: |
    - дякую
    - дуже дякую
    - спасибі
    - вельми вдячний

- intent: не_розумію
  examples: |
    - що?
    - не зрозумів
    - повтори
    - можете пояснити?
```

ДОДАТОК Г

Код файлу rules.yml

```
version: "3.1"
rules:
- rule: Відповісти на привітання
  steps:
  - intent: привітання
  - action: utter_привітання

- rule: Відповісти на прощання
  steps:
  - intent: прощання
  - action: utter_прощання

- rule: Відповісти на подяку
  steps:
  - intent: подяка
  - action: utter_подяка

- rule: Запитати про конкретну традицію та запустити дію
  steps:
  - intent: запитати_конкретну_традицію
  - action: action_розповісти_про_традицію

- rule: Запитати про загальні традиції
  steps:
  - intent: запитати_традицію
  - action: utter_запитати_традицію

- rule: Відповісти на нерозуміння
  steps:
  - intent: не_розумію
  - action: utter_не_розумію
```

ДОДАТОК Д

Код файлу stories.yml

```
version: "3.1"
stories:
- story: вітання та запит про традицію
  steps:
  - intent: привітання
  - action: utter_привітання
  - intent: запитати_традицію
  - action: utter_запитати_традицію

- story: запит про конкретну традицію
  steps:
  - intent: запитати_конкретну_традицію
    entities:
    - назва_традиції: "Великдень"
  - action: action_розповісти_про_традицію
  - intent: подяка
  - action: utter_подяка

- story: запит про іншу конкретну традицію
  steps:
  - intent: запитати_конкретну_традицію
    entities:
    - назва_традиції: "Різдво"
  - action: action_розповісти_про_традицію
  - intent: прощання
  - action: utter_прощання

- story: подяка після розповіді про Масляну
  steps:
  - intent: запитати_конкретну_традицію
    entities:
    - назва_традиції: "Масляна"
  - action: action_розповісти_про_традицію
  - intent: подяка
  - action: utter_подяка

- story: прощання
  steps:
  - intent: прощання
  - action: utter_прощання

- story: нерозуміння
  steps:
  - intent: не_розумію
  - action: utter_не_розумію
```

ДОДАТОК Е

Код файлу Action.py

```
from typing import Any, Text, Dict, List

from rasa_sdk import Action, Tracker
from rasa_sdk.executor import CollectingDispatcher
from rasa_sdk.events import SlotSet


# Припустимо, у вас є словник або база даних з інформацією про
традиції
traditions = {
    "Великдень": "Великдень - це велике християнське свято, яке
символізує Воскресіння Христове. Люди печуть паски, фарбують
писанки та ходять до церкви.",
    "Різдво": "Різдво в Україні святкують 7 січня. Традиційно
готують 12 пісних страв, співають колядки та ходять у гості.",
    "Івана Купала": "Івана Купала - це стародавнє свято літнього
сонцестояння. Молодь стрибає через вогнище, плете вінки та пускає
їх на воду.",
    "Масляна": "Масляна - це тиждень перед Великим постом, коли
люди готують та їдять млинці, влаштовують гуляння та спалюють
опудало зими.",
    "щедрування": "На Щедрий вечір (13 січня) діти та молодь
ходять по домівках і співають щедрівки, бажаючи господарям
добробуту.",
    "Святого Миколая": "День Святого Миколая святкують 19 грудня.
Дітям, які добре поводилися, Миколай приносить подарунки."
}

class ActionРозповістиПроТрадицію(Action):
    def name(self) -> str:
        return "action_розповісти_про_традицію"

    def run(self, dispatcher: CollectingDispatcher,
            tracker: Tracker,
            domain: dict) -> list:
        tradition_name = tracker.get_slot("tradition_name")

        if tradition_name in traditions:

dispatcher.utter_message(text=traditions[tradition_name])
        else:
            dispatcher.utter_message(text=f"Не знаю традицію
{tradition_name}.")
        return []
```