Міністерство освіти і науки України

Луцький національний технічний університет

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ГОЛОСОВОГО ПОМІЧНИКА ДЛЯ СТУДЕНТІВ КАФЕДРИ ІПЗ З ВИКОРИСТАННЯМ МОВИ ПРОГРАМУВАННЯ PYTHON

DEVELOPMENT OF A VOICE ASSISTANT FOR STUDENTS OF THE DEPARTMENT OF INDUSTRIAL ENGINEERING USING THE PYTHON PROGRAMMING LANGUAGE

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ІПЗс-21
Демчук Максим Сергійович
_____
(підпис)

Керівник:
асистент
Возняк Анастасія Вадимівна
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«8» червня 2024 р.
к.т.н., доцент
Гарант освітньої програми:
Ліщина Наталія Миколаївна
_____
(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Н. Лінчук
«2» 02 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Демчуку Максиму Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *розробка голосового помічника для студентів кафедри ІПЗ з використанням мови програмування Python*
Керівник роботи: *асистент, Вознюк Анастасія Вадимівна*
затверджені наказом закладу вищої освіти від «*30» грудня 2023 р.* № *477/01-02*

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«5» червня 2024 р.*

3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ, вимоги до розробки об'єкту проектування.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити):
Дослідження та аналіз існуючих рішень, аналіз і вибір засобів розробки голосового помічника, опис вимог асистента, розробка функціоналу помічника, реалізація.

5. Перелік графічного матеріалу: *9 рисунків, 2 таблиці.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Вознюк А. В.*		
Специфікації вимог до розробленої системи	*Вознюк А. В.*		
Розробка об'єкта проєктування	*Вознюк А. В.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту		99%	
Академічна доброчесність	*Ящук А. А.*		

7. Дата видачі завдання *«2» лютого 2024 р.*

КАЛЕНДАРНИЙ ПЛАН

№ З/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	**Примітка**
1	*Огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*07.02.2024 р.*	Виконано
2	*Аналіз проблеми розробки та впровадження об'єкту проектування*	*27.02.2024 р.*	Виконано
3	*Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання*	*12.03.2024 р.*	Виконано
4	*Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних*	*17.04.2024 р.*	Виконано
5	*Практична реалізація об'єкта проектування та розробка бази даних*	*18.05.2024 р.*	Виконано
6	*Тестування та налагодження об'єкта проектування*	*01.06.2024 р.*	Виконано
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*05.06.2024 р.*	Виконано

Здобувач вищої освіти _____ (підпис) *Демчук М. С.* (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (підпис) *Вознюк А. В.* (прізвище, ініціали)

Міністерство освіти і науки України
Луцький національний технічний університет

Рецензія
на кваліфікаційну роботу

Здобувач вищої освіти: Демчук Максим Сергійович

Тема: «Розробка Голосового помічника для студентів кафедри ІПЗ з використанням мови програмування Python»

Коротка характеристика кваліфікаційної роботи: Кваліфікаційна робота бакалавра присвячена розробці голосового помічника на мові програмування Python для виконання повсякденних завдань. Асистент дозволяє користувачам задати запитання та отримати потрібну відповідь, відкрити сторінку в браузері або виконати дію та дає можливість створення та введення необмеженої кількості команд. У роботі використано сучасні технології, такі як , Python та SQLite3.

Самостійні розробки і пропозиції автора: У роботі детально описаний процес розробки голосового помічника та доведено ефективність запропонованих підходів. Розроблено інтерфейс користувача, який забезпечує інтуїтивно зрозуміле керування операціями по запиту та отримання інформації, а також реалізовано функціонал для для отримання актуального розкладу занять.

Практичне значення роботи: Голосовий помічник був успішно реалізований і може бути використаний широкою аудиторією для виконання різного типу завдань. Асистента протестовано в умовах, наближених до реальних, і він показав високу стабільність та зручність у користуванні. Кваліфікаційна робота має значне практичне значення.

Недоліки: В роботі варто було б провести більш глибокий аналіз конкурентних рішень на ринку.

Загальний висновок: Робота виконана на високому науковому рівні, має логічну та послідовну структуру, відповідає встановленим вимогам, може бути представлена до захисту на державній екзаменаційній комісії та заслуговує позитивної оцінки.

Рецензент: к.т.н, доцент, Редько Ольга Іванівна

Рецензент кваліфікаційної роботи _____
(підпис)

«07» червня 2024 р.

Міністерство освіти і науки України
Луцький національний технічний університет

Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

ВІДГУК
КЕРІВНИКА НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Здобувач вищої освіти *Демчук Максим Сергійович*
група *ІПЗс-21*

Тема кваліфікаційної роботи: *Розробка голосового помічника для студентів кафедри ІПЗ з використанням мови програмування python*
Керівник *Вознюк Анастасія Вадимівна*

Актуальність теми: тема дослідження та розробка голосового помічника на мові програмування Python є актуальним завданням у сучасному повсякденні, коли кількість завдань збільшується і тим самим потребує більше часу на поміч прийде асистент який допоможе вирішити рутинні завдання. Незважаючи на існування численних голосових помічників, ще залишається потреба у вирішенні питань зручності користування, збільшення функціоналу, також можливості вдосконалення технології.

Об'єкт дослідження – це голосовий помічник для студентів кафедри ІПЗ з використанням мови програмування Python.

Характеристика теоретичного рівня, наявності самостійних розробок і практичної значимості роботи: студентом був створений ефективний голосовий помічник для виконання повсякденної рутинної роботи, яка дозволяє користувачам зручно і швидко отримати потрібну інформацію. В голосовому помічнику реалізована можливість створення та введення необмеженої кількості команд, що робить його універсальним для різних користувачів. Було розроблено функції для отримання інформації про розклад занять, відкриття популярних сайтів, та відповіді а популярні запитання. Голосовий помічник має сучасний і інтуїтивно зрозумілий інтерфейс, що забезпечує високу продуктивність. Усі розробки були протестовані в умовах, наближених до реальних, і показали високий рівень стабільності та зручності використання.

Зауваження та недоліки: істотних недоліків не виявлено.

Загальний висновок робота виконана на високому рівні та заслуговує оцінки «відмінно» за умови успішного захисту.

Керівник _____ **Вознюк Анастасія Вадимівна**
(підпис) (прізвище, ім'я, по батькові)

«07» червня 2024 р.

АНОТАЦІЯ

Демчук М. С. Розробка голосового помічника для студентів кафедри ІПЗ з використанням мови програмування Python. Рукопис.

Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2024.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз сучасного стану проблеми розробки голосових асистентів і сформульовані завдання на кваліфікаційну роботу. В другому розділі визначено вимоги до розроблюваної системи, а також засоби, методи і алгоритми розробки. У третьому розділі здійснено практичну реалізацію голосового помічника. У висновках узагальнено інформацію, відображену у попередніх частинах. Результати розробки демонструють можливості реалізації голосового асистента в повсякденному житті.

Ключові слова: rhvoice volodymyr, Pyttsx3, speech_recodnition, sqlite3, api, ос.

ABSTRACT

Demchuk M. S. Development of a voice assistant for students of the Department of Industrial Engineering using the Python programming language. Manuscript.

Bachelor's qualifying thesis of the OP "Software Engineering". Lutsk National Technical University. Lutsk, 2024.

The bachelor's qualification work consists of an introduction, three sections, conclusions, a list of used sources and appendices.

In the first chapter, an analysis of the current state of the problem of the development of voice assistants was carried out and tasks for the qualification work were formulated. The second section defines the requirements for the developed system, as well as development tools, methods and algorithms. In the third section, the practical implementation of the voice assistant is carried out. The conclusions summarize the information presented in the previous parts. The development results demonstrate the possibilities of implementing a voice assistant in everyday life.

Keywords: rhvoice volodymyr, Pyttsx3, speech_recodnition, sqlite3, api, os.

ЗМІСТ

ВСТУП

Актуальність теми. Ритм і динаміка людського життя покращується з кожним днем. Люди не мають часу читати довгі статті, дивитися довгі відео, навіть не мають часу писати замітки. Отже, люди частіше користуються голосовими помічниками. Gartner прогнозує, що найближчим часом 30% сеансів перегляду веб-сторінок відбуватимуться без екрана, а 50% усіх запитів будуть голосовими.

Голосові помічники дозволяють зменшити, а іноді й повністю позбутися потреби у практичній візуальній навігації. Але поза повсякденним життям голосові помічники можуть бути корисними в бізнесі.

З розвитком комп'ютерних технологій з'явилося програмне забезпечення під загальною назвою «Голосовий помічник». Ідея віртуального помічника полягала в тому, щоб допомогти кожному знайти потрібну інформацію. Сьогодні це успішно роблять такі віртуальні помічники, як: Google Assistant, розробник Apple Siri та Alexa від Amazon.

Голосові помічники дуже корисні. Вони активні в багатьох сферах діяльності. Голосовий помічник може виконувати багато корисних завдань, таких як: озвучити прогноз погоди, здійснення телефонних дзвінків, запуск програм, пошук інформації, зміна налаштувань, увімкнення музики та багато іншого.

Є кілька способів реалізації голосових помічників. Наприклад, Siri Assistant розроблено на мові програмування Objective C, а Google Assistant – на мові C++.

Метою кваліфікаційної роботи є розробка голосового помічника на мові програмування Python з використанням мови синтезу та виділених бібліотек.

Завдання кваліфікаційної роботи:

– провести аналіз вимог користувачів до голосового помічника;
– проаналізувати існуючі голосові асистенти;

– здійснити вибір методів та алгоритмів розробки голосового помічника;

– розробити інтерфейс, який буде інтуїтивно зрозумілим і зручним для студентів;

– провести тестування та вдосконалення голосового помічника;

– написати докладну документацію до голосового помічника.

Об’єктом дослідження є програмний продукт «Голосовий асистент Роман».

Предметом дослідження є процес розробки та тестування голосового помічника, виправлення недоліків та написання документації.

РОЗДІЛ 1
АНАЛІЗ СУЧАСНИХ ГОЛОСОВИХ АСИСТЕНТІВ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Розглянемо, що таке голосовий помічник. Практично неможливо назвати технологію, яка полегшує наше життя. Існують різні терміни, які стосуються помічників, які можуть виконувати завдання чи послуги для людей, і вони здебільшого сумісні, але не зовсім різні. Нижче наведено деякі основні визначення, подібності та відмінності:

– інтелектуальний особистий помічник – це програмне забезпечення, яке може допомагати людям у виконанні основних завдань, зазвичай природною мовою. Розумний особистий помічник може вийти в Інтернет і знайти відповіді на ваші запитання. Ви можете запускати дії за допомогою тексту або голосу;

– автоматизований персональний помічник. Цей термін є синонімом інтелектуального персонального помічника;

– віртуальні цифрові помічники – це автоматизовані програмні додатки або платформи, які допомагають користувачам розуміти природну мову, письмову чи усну;

– розумний помічник. Цей термін зазвичай відноситься до типу фізичного елемента, який може надавати різноманітні послуги за допомогою розумного динаміка, який слухає певні слова та виконує певні завдання. Echo від Amazon, Home від Google і HomePod від Apple – це всі види розумних помічників.

Голосовий помічник – це цифровий помічник, який використовує розпізнавання мовлення, перетворення тексту в мовлення та обробку природної мови для надання послуг через певні програми. Коли мова заходить про використання голосових помічників, багато пристроїв, якими ми користуємося

щодня, використовують голосові помічники. Вони є на наших смартфонах і домашніх пристроях у наших домівках. Використовується багатьма мобільними додатками та операційними системами. Крім того, їх можна знайти в автомобільних, освітніх, медичних і телекомунікаційних установах, і ними можна керувати голосом.

Розпізнавання мовлення – це здатність машини чи програми отримувати й інтерпретувати диктант або розуміти й виконувати голосові команди. Популярність і використання розпізнавання мовлення зросли з появою розумних помічників, таких як: Alexa, Siri від Apple і Cortana від Microsoft. Системи розпізнавання мовлення дозволяють користувачам взаємодіяти з технологіями, просто кажучи, запитуючи веб-ресурси, створюючи нагадування та виконуючи інші прості завдання без рук.

Програма розпізнавання мовлення на вашому комп'ютері має перетворювати аналогові аудіо сигнали на цифрові. Це відоме як аналого-цифрове перетворення. Щоб комп'ютер розшифрував сигнал, йому потрібна цифрова база даних або словник слів і фраз і простий спосіб порівняти ці дані з сигналом. Мовні моделі зберігаються на диску та завантажуються в пам'ять під час запуску програми. Компілятор переглядає ці збережені шаблони на виході. Це операція, яка називається зіставленням шаблону.

На практиці ефективний словниковий запас механізму розпізнавання мовлення безпосередньо залежить від обсягу оперативної пам'яті комп'ютера, на якому він встановлений. Програма розпізнавання мовлення працює в рази швидше, якщо весь словник можна завантажити в пам'ять. Також важлива швидкість обробки. Це пояснюється тим, що це впливає на швидкість пошуку комп'ютером відповідної пам'яті.

Технологія розпізнавання мовлення виникла в комп'ютерах, але знайшла визнання як у бізнесі, так і в споживчому середовищі, включаючи мобільні телефони та побутові пристрої. Популярність смартфонів відкрила можливість помістити технологію розпізнавання голосу в «кишені» споживачів, а домашні пристрої, такі як Google Home і Amazon Echo, принесли технологію

розпізнавання голосу у вітальні та кухні. Розпізнавання мовлення в поєднанні зі зростаючим Інтернетом речей підняло багато споживчих програмних продуктів на новий рівень.

Оскільки використання технології розпізнавання мовлення зростає та все більше користувачів взаємодіють із нею, компанії, що розробляють програмне забезпечення для розпізнавання мовлення, запроваджують можливості механізму розпізнавання, покращуючи тим самим функціональність і точність своїх продуктів.

Цифрові домашні помічники від відомих компаній, таких як Google, Amazon і Apple, використовують програмне забезпечення для розпізнавання мови для взаємодії з користувачами. Те, як споживачі використовують технологію розпізнавання мовлення, залежить від продукту, але може включати перетворення в текст, налаштування нагадувань, пошук в Інтернеті та відповіді на прості запитання та запити, наприклад ввімкнення музики та обмін інформацією про погоду.

Перевага розпізнавання голосу полягає в тому, що користувачі можуть виконувати багато функцій, звертаючись безпосередньо до Google Home, Amazon Alexa або іншої технології розпізнавання голосу. Використовуючи машинне навчання та складні алгоритми, технологія розпізнавання мовлення може швидко перетворити будь-яку мову на письмовий текст.

Недоліком є те, що хоча точність покращується, усі системи та програми розпізнавання мовлення допускають помилки. Фоновий шум може спричинити помилкові введення, яких можна уникнути, використовуючи систему в тихій кімнаті. Існує також проблема слів, які звучать однаково, але пишуться по-різному та мають різне значення. Цю проблему одного разу можна буде значною мірою подолати завдяки збереженій контекстній інформації. Однак для цього знадобиться більше оперативної пам’яті та швидші процесори, ніж доступно в сучасних персональних комп’ютерах.

У більшості випадків асистента-помічника потрібно «розбудити», вимовивши його ім’я («Hey Siri», «OK Google», «Alexa»), щоб скористатися

його функціями. Більшість віртуальних помічників достатньо розумні, щоб розуміти природну мову, але їм потрібно говорити певні фрази. Наприклад, якщо ви підключите свій Amazon Echo до програми Uber, Alexa зможе замовити таксі, але ви повинні правильно сказати команду Alexa, попросіть Uber замовити таксі».

Зазвичай потрібно поговорити зі своїм віртуальним помічником, так як він чекає певної голосової команди. Однак деякі помічники можуть реагувати на введені команди. Наприклад, на пристроях iPhone з iOS 11 і пізніших версій ви можете вводити запитання та команди в Siri, а не промовляти їх. За бажанням Siri також може відповісти текстом замість голосу. Так само «Google Assistant» може відповідати на команди, які ви вводите голосом або текстом [4].

На своєму смартфоні ви можете використовувати віртуальних помічників для налаштування параметрів і виконання таких завдань, як надсилання текстових повідомлень, здійснення дзвінків і відтворення пісень. Розумні колонки дозволяють керувати іншими розумними пристроями у вашому домі, такими як термостат, освітлення або системи безпеки.

Голосовий помічник – так званий пасивний слуховий апарат, який розпізнає та реагує на команди та привітання (наприклад, «Hey Siri»). Це означає, що ваш пристрій завжди прислухається до того, що відбувається навколо вас. Це може активувати певні функції конфіденційності, як це демонструють смарт-пристрої, які виступають свідками злочинів.

Для роботи голосовий помічник має бути підключений до Інтернету, щоб він міг шукати відповіді в Інтернеті або спілкуватися з іншими розумними пристроями. Однак, оскільки вони є пасивними аудіо пристроями, їх зазвичай потрібно активувати, щоб працювати. Під час голосової взаємодії з віртуальним помічником ви можете промовити до помічника та задати своє запитання без перерви. Наприклад: «Привіт, Siri, як результати Eagle?» Якщо віртуальний помічник не зрозуміє вашу команду або не зможе знайти відповідь, він повідомить вас про це, і ви зможете спробувати ще раз, злегка перефразувавши своє запитання або кажучи голосніше чи повільніше.

Наприклад, у деяких випадках під час використання програми Uber вас можуть попросити надати додаткову інформацію про ваше поточне місцезнаходження чи пункт призначення.

Віртуальних помічників на основі смартфонів, таких як Siri та Google Assistant, також можна активувати, натиснувши та утримуючи кнопку «Додому» на пристрої. Потім ви можете ввести своє запитання чи запит, і Siri та Google дадуть відповідь текстом. Розумні колонки, такі як Amazon Echo, можуть реагувати лише на голосові команди.

Для роботи голосовий помічник має бути підключений до Інтернету, щоб він міг шукати відповіді в Інтернеті або спілкуватися з іншими розумними пристроями. Однак, оскільки вони є пасивними аудіо пристроями, їх зазвичай потрібно активувати, щоб працювати.

Під час голосової взаємодії з віртуальним помічником ви можете зателефонувати помічнику та задати своє запитання без перерви. Наприклад: «Привіт, Siri, як результати Eagle?» Якщо віртуальний помічник не зрозуміє вашу команду або не зможе знайти відповідь, він повідомить вас про це, і ви зможете спробувати ще раз, злегка перефразувавши своє запитання або кажучи голосніше чи повільніше.

Голосові помічники дозволяють зменшити, а іноді й повністю позбутися потреби у практичній візуальній навігації. Але поза повсякденним життям голосові помічники можуть бути корисними в бізнесі та навіть у відділі кадрів.

Розглянемо існуючі реалізації голосового асистенту для персональних комп’ютерів.

Cortrana – це віртуальний помічник, розроблений Microsoft для Windows 10, Windows 10 Mobile, Windows Phone 8.1, Microsoft Band, Surface Headphones, Xbox One і Amazon Alexa (рис. 1.1). Cortana може встановлювати нагадування, розпізнавати природний голос без клавіатури та відповідати на запитання за допомогою тексту з пошукової системи Bing. Зараз Cortana доступна англійською, португальською, французькою, німецькою, італійською,

іспанською, китайською та японською мовами, залежно від програмної платформи, на якій вона використовується [3].

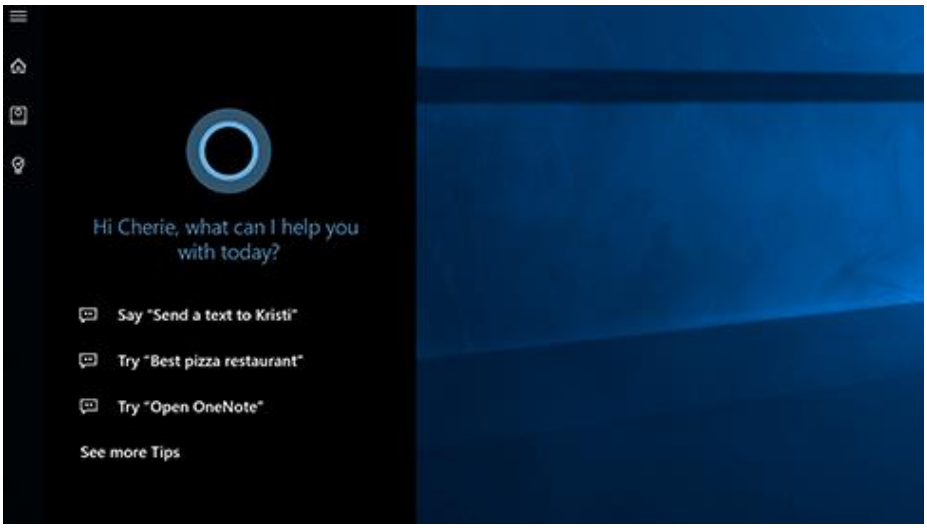

Рисунок 1.1 – Інтерфейс голосового асистенту Cortrana

Переваги:

– можливість розпізнавати голос кількома мовами;

– можливість голосового пошуку через пошукову систему Bing;

– можливість встановлювати нагадування та керувати календарем голосом;

– можливість інтеграції з сервісом Amazon Alexa;

– можливість читати з екрана та вводити запитувану інформацію з голосового зчитувача;

– можливість надсилати електронні листи за допомогою голосового керування.

Недоліки:

– додаток має сувору політику щодо використання продуктів Microsoft, яка не дозволяє використання будь-яких інших пошукових систем, окрім браузерів для продуктів Microsoft;

– він може працювати лише на платформах Microsoft (Windows, XboxOne тощо), що обмежує кількість користувачів;

– бібліотеки, що містять усі команди розпізнавання голосового коду локально на комп’ютері, а при використанні завантажуються в оперативну пам’ять;

– немає можливості ідентифікувати користувача за зображенням обличчя.

Siri – це віртуальний помічник, який інтегрується з операційною системою Apple, iOS, watchOS, macOS, tvOS і audioOS (рис. 1.2). Помічник використовує голосові підказки та інтерфейс користувача для вивчення природної мови, щоб відповідати на запитання, робити пропозиції та виконувати дії, надсилаючи запити до ряду Інтернет-сервісів. Програмне забезпечення автоматично перемикається на особисте використання мови, пошук і налаштування при регулярному використанні [9].

Siri підтримує широкий спектр команд користувача, включаючи здійснення телефонних дзвінків, перегляд ключових подій, керування подіями та нагадуваннями, керування налаштуваннями пристрою, перегляд Інтернету, навігацію по місцях, пошук розважальної інформації та підключення до програм, інтегрованих у iOS.

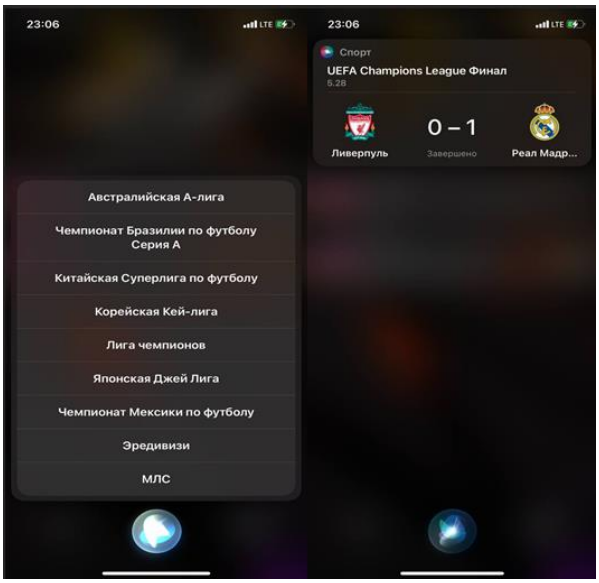

Рисунок 1.2 – Приклад пошуку голосовим запитом

З випуском iOS 10 у 2016 році Apple відкрила обмежений доступ до Siri, включаючи сторонні розробники та програми для обміну повідомленнями, а також додатки для платежів, обміну та Інтернету. З випуском iOS 11 Apple оновила голос Siri для більш чіткого, більш людського голосу, почала підтримувати нові запитання та мовні переклади, а також нові дії третіх сторін.

Siri – віртуальний помічник, що входить до складу операційної системи Apple, iOS, watchOS, macOS, tvOS і audioOS. Асистент використовує голосові запити та користувальницький інтерфейс для вивчення природної мови для відповіді на запитання, надання рекомендацій та виконання дій шляхом делегування запитів до набору Інтернет-послуг. Програмне забезпечення адаптується до індивідуальних користувацьких мов, пошуків і уподобань з постійним використанням [9].

Переваги:

– розпізнавання голосу;

– виконати голосовий пошук, після чого вивести результати;

– встановлюйте нагадування та керуйте календарем за допомогою голосу;

– здатність викликати;

– можливість спілкуватися з користувачем голосом або словами;

– налаштувати пристрої.

Недоліки:

– програма доступна лише для пристроїв однієї компанії (Apple), що обмежує кількість користувачів;

– бібліотеки з усіма командами розпізнавання голосового коду локально доступні на комп’ютері, а при використанні завантажуються в оперативну пам’ять;

– неможливо ідентифікувати користувача за зображенням обличчя.

Google Assistant – це віртуальний помічник на основі штучного інтелекту, розроблений Google, доступний переважно на мобільних пристроях і пристроях

розумного дому (рис. 1.3). На відміну від попереднього віртуального помічника компанії Google Assistant, Google Assistant здатний вести двосторонні розмови [4].

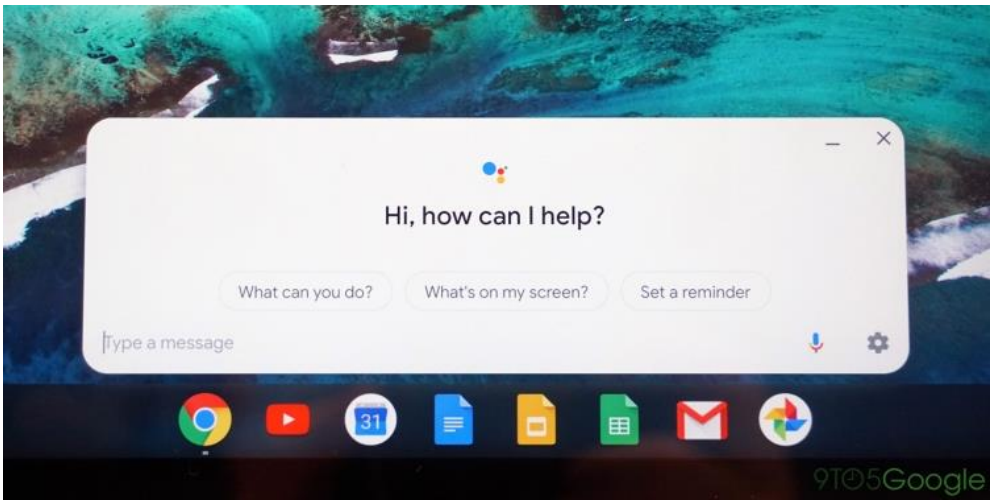

Рисунок 1.3 – Інтерфейс голосового асистенту Google assistant

Користувачі взаємодіють з Google Assistant переважно своїм природним голосом, хоча також підтримується введення з клавіатури. У тій же якості та стилі, що й Google Assistant, Assistant може здійснювати пошук в Інтернеті, планувати події та будильники, змінювати параметри апаратного забезпечення на пристрої користувача та відображати інформацію з підтвердження облікового запису Google користувача. Google також оголосив, що помічник зможе ідентифікувати об’єкти та збирати візуальну інформацію через камеру пристрою, а також допомагатиме купувати товари та надсилати гроші, а також музику.

Переваги:

– здатність розпізнавати голос;

– можливість голосового пошуку та виведення результатів;

– можливість встановлювати нагадування та керувати календарем голосом;

– налаштувати обладнання;

– можливість підтримки онлайн-покупок;

– можливість розпізнавати обличчя або зображення з камери.

Недоліки:

– програма доступна лише у браузері для персональних комп’ютерів, тому на даний момент потребує додаткової підтримки браузера;

– він може використовувати лише одну пошукову систему.

У проекті були вивчені найпопулярніші програми, що позиціонуються як голосові помічники або віртуальні помічники для персональних комп’ютерів, які можуть виконувати певні дії за допомогою голосу користувача. Представлені програми з конкретною реалізацією для їх ОС. Кожна з представлених функцій має ряд плюсів мінусів, після аналізу яких можна визначити оптимальні вимоги до функціональності розробленого додатку для задоволення конкретних потреб користувача.

Розглянемо основні недоліки готового рішення:

– неможливо розширити розроблений функціонал, оскільки кожна з представлених програм була розроблена із закритим вихідним кодом, що унеможливлює безперебійну розробку з використанням коду з відкритим кодом. Практика неможлива;

– так як голосовий помічник не може бути перенесений на іншу платформу, так як кожна програма розроблялася для своєї ОС;

– неможливо забезпечити конфіденційність даних, які використовуються під час використання голосового помічника (пошукові записи, особисті записи тощо), тому що захист обмежений лише ОС, і доступу до неї, до того ж, ніхто не отримує. доступ до програми за зображенням обличчя користувача;

– майже всі голосові помічники для ОС використовують для розпізнавання локальні бібліотеки голосових кодів, які можуть займати багато місця в ОС користувача та споживати велику кількість оперативної пам’яті та

обчислювальних ресурсів, тому що для встановлення необхідних правил ідентифікації, майже вся бібліотека завантажується в оперативну пам’ять;

– оскільки додаток Google Assistant є функціональним розширенням браузера Google Chrome, його можна вважати кросплатформним, але він також покладається безпосередньо на нього та використовує пошукові запити лише через пошук Google (інші голосові помічники також використовують лише свої дослідження обладнання), що може бути проблемою зі статистичною надійністю даних.

На основі проаналізованих недоліків можна виділити основні вимоги до продуктивності програми:

– голосовий помічник, який не залежить від платформи і не використовує локальну бібліотеку для розпізнавання голосу;

– можливість програми записувати голосову команду та відображати результат;

– розроблений додаток має бути легко розширюваним за допомогою технології «Open source».

Щоб усунути недоліки, описані в існуючих аналогічних додатках, необхідно розробити додаток із зазначеними вище вимогами.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Метою цієї кваліфікаційної роботи є створення голосового помічника з використанням мови програмування Python, залучаючи інструменти для синтезу мови та спеціалізовані бібліотеки.

Для реалізації поставленої мети були сформульовані наступні завдання:

– проаналізувати існуючі голосові асистенти, а саме розглянути принципи роботи сучасних голосових помічників, визначити їх переваги та недоліки;

– провести аналіз вимог користувачів до голосового помічника, визначити, які функції потрібні студентам і які запити вони найчастіше будуть використовувати. Визначити платформи, з якими повинен бути сумісний голосовий помічник (наприклад, операційні системи, пристрої). Розробити діаграму варіантів використання для голосового помічника;

– здійснити вибір методів та алгоритмів розробки голосового помічника. Обрати бібліотеки та фреймворки, які будуть використовуватись під час розробки та дадуть змогу здійснювати розпізнавання мови, обробку запитів та генерацію відповідей;

– розробити інтерфейс, який буде інтуїтивно зрозумілим і зручним для студентів. Здійснити реалізацію функції відповіді на запитання студентів, включаючи доступ до бази знань або пошукової системи. Додати можливість виконувати команди з відкриття сайтів та додатків, забезпечивши правильне розпізнавання команд. Підключити базу даних для збереження команд голосового помічника;

– провести тестування та вдосконалення голосового помічника на різних пристроях, щоб забезпечити правильне розпізнавання мови. Виконати тестування помічника на різних сценаріях використання;

– написати докладну документацію до голосового помічника, включаючи інструкції з використання, встановлення та налаштування.

Висновки до розділу 1

Голосовий помічник – це віртуальний помічник, якому ви можете віддавати голосові команди для різних вказівок, наприклад, змінити час будильника, відкрити потрібний додаток, увімкнути навігатор або отримати відповідь на запитання зараз.

За допомогою однієї команди користувач може знайти своє місцезнаходження на карті, пограти в улюблену гру, встановити час і дату, чи дізнатися розклад занять а також отримати відповідь на поширені запитання.

В даному розділі було проведено аналіз сучасного стану проблеми, а саме опис принципу роботи сучасних голосових помічників, їх плюсів та мінусів. Відповідно до мети кваліфікаційної роботи було здійснено постановку завдань, які потрібно виконати під час роботи над кваліфікаційною роботою.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Основними вимогами до розроблюваного програмного забезпечення є правильне розпізнавання голосу користувача та відтворення відповідей на запитання. Саме мова програмування Python чудово підходить як середовище розробки для голосових помічників, оскільки вона містить в собі різні потужні бібліотеки та API для легкого створення функціоналу розпізнавання та відтворення мовлення.

До прикладу можна взяти наступні модулі:

– pyttsx3. Цей модуль використовується для перетворення тексту в мову в програмі, яка працює в автономному режимі. Щоб встановити цей модуль, введіть у терміналі наведену нижче команду import pyttsx3 [8];

– speech recognition. Для створення голосового асистента одним з найважливіших моментів є те, щоб асистент розпізнавав ваш голос (тобто те, що ви хочете сказати/запитати);

– sqlite – це бібліотека C, яка забезпечує легку дискову базу даних, що не вимагає окремого серверного процесу і дозволяє отримувати доступ до бази даних за допомогою нестандартного варіанту мови запитів SQL. Деякі програми можуть використовувати SQLite для внутрішнього зберігання даних. Також можна створити прототип програми з використанням SQLite, а потім перенести код на більшу базу даних, таку як PostgreSQL або Oracle [10].

Для більш детального вияву потреби студентів щодо запитів голосовому асистенту потрібно провести опитування, що вияснить найбільш популярні запитання студентів кафедри ІПЗ і сформувати відповіді на дані запитання які будуть зрозумілими для користувачів та детально розписаними.

Також не менш важливим буде вибір зрозумілого та приємного на слух голосу для голосового помічника, для цього необхідно обрати та встановити голосовий модуль та відповідно по бажанню обрати голос жінки чи чоловіка.

Ще одним ключовим моментом є надання інформації щодо розкладу занять, що є най частішим запитом будь якого студента ну і також не варто забувати про розклад дзвінків.

Можливість за допомогою команди відкривати в браузері сайт університету чи інші програмні засоби для зручного та швидкого перегляду чи прослуховування різного типу інформації. Нижче зображено діаграму варіантів використання UML (рис. 2.1).

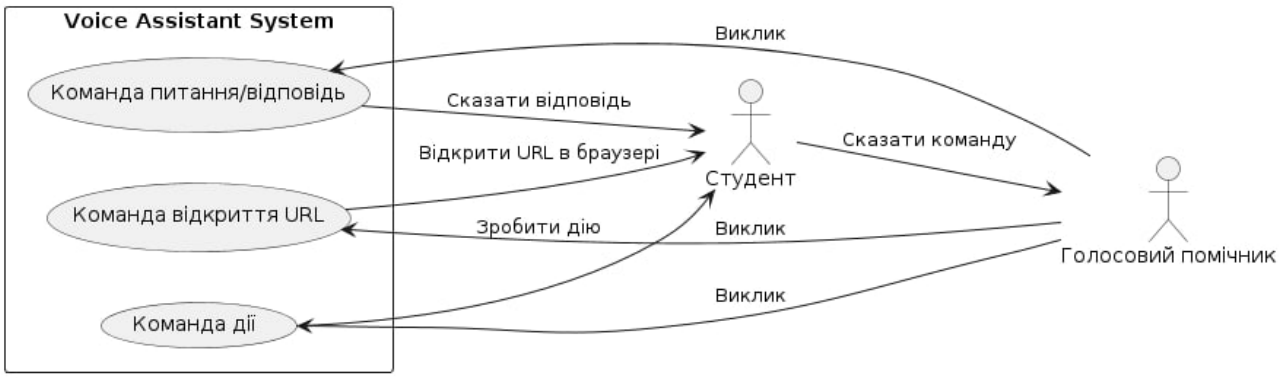

Рисунок 2.1 – Діаграма варіантів використання

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Існують різні способи реалізації персонального голосового помічника. У цьому проекті буде використано мову програмування Python.

Python – це об’єктно-орієнтована мова програмування високої чіткості зі строгою динамічною типізацією. Розроблена Гвідо ван Россумом у 1990-х роках. Висока структура даних у поєднанні з динамічним зв’язуванням робить його привабливим для швидкої розробки додатків і як спосіб поєднання існуючих компонентів. Python підтримує модулі та пакети модулів, які

заохочують модульність і повторне використання коду. Інтерпретатор та стандартні бібліотеки Python доступні у скомпільованій та вихідній формі на всіх основних платформах. Мова програмування Python підтримує низку моделей програмування, зокрема: об’єктно-орієнтоване, процедурне, функціональне та аспектно-орієнтоване [12].

Для створення голосового помічника необхідно встановити сторонні бібліотеки за допомогою системи управління пакетами pip.

pip – це система керування пакетами для встановлення та керування програмними пакетами, написаними мовою Python. Багато пакетів можна знайти в індексі пакетів Python. Починаючи з Python 2.7.9 і Python 3.4, пакет pip включено за замовчуванням [7].

Для додавання та збереження команд потрібна СУБД. SQLite3 – досить легкий і поставляється в комплекті з Python.

SQLite 3 – це легка реляційна система баз даних. Він встановлений у формі бібліотеки, де використовуються багато стандартів SQL-92. Вихідний код SQLite поширюється як публічне надбання, тобто його можна використовувати без обмежень і безкоштовно для будь-яких цілей [10].

Бібліотека, яка взаємодіє з аудіо драйверами для керування мікрофоном. PyAudio – це набір прив’язок Python для PortAudio, кросплатформної бібліотеки C++.

Бібліотека Speech recognition розпізнавання мовлення надається для використання мікрофона та допоміжних систем розпізнавання. Це залежить від PyAudio під час використанні мікрофона.

Бібліотека виявлення, яка підтримує широкий спектр механізмів і API, як онлайн, так і офлайн.

Підтримка системи розпізнавання мови/API:

– CMU Sphinx (works offline);
– Google Speech Recognition;
– Google Cloud Speech API;

– Wit.ai;
– Microsoft Bing Voice Recognition;
– Houndify API;
– IBM Speech to Text;
– Snowboy Hotword Detection.

Для перетворення заданого тексту в голос за допомогою синтезованого голосу RHVoice Volodymyr на Python зручно використати бібліотеку Pyttsx3. На відміну від інших бібліотек вона працює в автономному режимі і сумісна як з Python 2 так і з Python 3 [8].

Для прикладу можна створити функцію speak і передати аргумент text. Далі використати фабричну функцію pyttsx3.init(), щоб отримати посилання на pyttsx3.Engine – екземпляр. Під час будування двигун ініціалізує pyttsx3.driver.DriverProxy об'єкт відповідальний за завантаження реалізації драйвера вимовного двигуна із pyttsx3.drivers модуля. Після побудови програми використовується об'єкт механізму для реєстрації і відміни реєстрації зворотніх виклків подій; виготовляти і зупиняти цикли подій.

Завод двигунів pyttsx3.init (ім'я_драйвера: рядок, відладка: логічне значення) pyttsx3.engine. Отримує посилання на примірник механізму, котрий буде використовувати даний драйвер. Якщо запрошений драйвер уже використовує другим примірником движка, вертає цей двигун. В інакшому випадку створиться новий двигун. Параметри та дії описані в таблиці 1.1.

Таблиця 1.1 – Параметри та дії

Параметри	driverName – ім'я pyttsx3.drivers модуля для завантаження і використання. На даний момент за замовчуванням використовується кращий доступний драйвер для платформи sapi5 – SAPI5 використовується для Windows, nsss – NSSpeechSynthesizer для Mac OS X, espeak – eSpeak на будь-якій іншій платформі
Піднімає	ImportError – коли запрошений драйвер не знайдено. RuntimeError – коли драйвер не може ініціалізуватися

Інтерфейс Engine класс pyttsx3.engine.Engine представляє додаткам доступ до синтезу мовлення.

– getProperty(name, string) – поточне значення властивості механізму;

– параметри name – ім’я властивості для запиту;

– повертає значення якості під час цього виклику;

– наступні імена властивостей допустимі всім драйверів;

– rate – цілочисленна швидкість мови в словах за хвилину. За замовчуванням 200 слів за хвилину;

– voice – строковий ідентифікатор активної застави;

– voices – список pyttsx3.voice.Voice об’єктів дескриптора;

– volume – об’єм з плаваючою комою в діапазоні від 0,0 до 1,0 включно. Типово 1.0.

SetProperty(name , value) – ставить у чергу команду для встановлення якості двигуна. Нове значення властивості впливає всі репліки, поставлені у чергу після цієї команди.

RunAndWait( ) – блокує під час обробки всіх поточних команд у черзі. Викликає зворотні дзвінки для повідомлень механізму відповідним чином. Повертає, коли всі команди, які стояли у черзі перед цим викликом, очищаються від черги.

Say (text: Unicode, name: string) – ставить у чергу команду вимовити висловлювання. Йдеться відповідно до властивостей, заданих перед цією командою в черзі.

Параметри:

– «text» текст для вимовлення;

– «name» ім'я, яке буде пов’язане з висловлюванням. Включено до сповіщень про це висловлювання;

– «stop()» зупиняє поточне висловлювання та очищає чергу команд.

Розмовний текст опилано в лістингу 2.1.

Лістинг 2.1 – Розмовний текст

```
import pyttsx3

engine = pyttsx3.init()
engine.say('Hello world!')
engine.say('Some text.')
engine.runAndWait()
```

Кінець лістингу 2.1

PyAudio потрібно тоді і тільки тоді, коли хочеться використовувати мікрофонний вхід (Microphone). Потрібна PyAudio версії 0.2.11+, тому що в попередніх версіях були відомі помилки управління пам’яттю при записі з мікрофонів у певних ситуаціях.

Якщо він не встановлений, все в бібліотеці, як і раніше, працюватиме, за винятком того, що спроба створити екземпляр об’єкта Microphone викличе AttributeError . У Windows встановіть PyAudio за допомогою Pip: виконайте pip install pyaudio у терміналі. У дистрибутивах Linux, похідних від Debian (наприклад, Ubuntu та Mint), встановлюється PyAudio за допомогою APT: потрібно виконати sudo apt-get install python-pyaudio python3-pyaudio у терміналі [8].

Якщо версія в репозиторіях занадто стара, потрібно встановити останню версію за допомогою Pip: виконайте sudo apt-get install portaudio19-dev python-all-dev python3-all-dev && sudo pip install pyaudio (замініть pip на pip3, якщо використовуєте Python 3).

У OS X PortAudio всановлюється за допомогою Homebrew: brew install portaudio. Після чого PyAudio встановлюється за допомогою Pip: pip install pyaudio. В інших системах на основі POSIX установлюються пакети portaudio19-dev та python-all-dev (або python3-all-dev при використанні Python 3) (або їх найближчі еквіваленти) за допомогою диспетчера пакетів, а потім встановіть PyAudio за допомогою Pip: pip install pyaudio.

Пакети PyAudio для найпоширеніших 64-розрядних версій Python для Windows і Linux включені для зручності в каталог Third-Party/ в корені

репозиторію. Для установки просто запускається командою pip install wheel, а потім pip install ./first-party/WHEEL_FILENAME у кореневому каталозі репозиторію.

Версії, доступні в більшості репозиторіїв пакетів, застаріли і не працюватимуть із пов’язаними мовними даними. Рекомендується використовувати комплектні пакети коліс або збирання з вихідного коду.

Клієнтська бібліотека Google API для Python (для користувачів Google Cloud Speech API). Клієнтська бібліотека Google API для Python потрібна лише тоді, коли ви хочете використовувати Google Cloud Speech API (cognir_instance.recognize_google_cloud). Якщо він не встановлений, все в бібліотеці, як і раніше, працюватиме, за винятком того, що виклик cognir_instance.recognize_google_cloud викликає RequestError [11].

Згідно з офіційними інструкціями по установці, рекомендований спосіб встановлення – за допомогою Pip: виконайте pip install google-api-python-client (замініть pip на pip3, якщо використовуєте Python 3).

Крім того, можна виконати інсталяцію повністю в автономному режимі з вихідних архівів в каталозі Third-Party/Source для клієнтської бібліотеки Google API для Python та її залежностей [6]. У Python 2 і тільки в Python 2, якщо не встановити бібліотеку Monotonic for Python 2 деякі функції будуть працювати повільніше, ніж могли б в іншому випадку (хоча все як і раніше буде працювати правильно). У Python 3 функціональність цієї бібліотеки вбудована у стандартну бібліотеку Python, що робить її непотрібною.

Це пов’язано з тим, що монотонний час необхідний для правильної обробки закінчення терміну дії кешу перед змін системного часу та інших проблем, пов’язаних з часом. Якщо функціональність монотонного часу недоступна, такі речі, як запити маркерів доступу, кешуватися не будуть. Для встановлення використовується pip потрібно виконати pip install monotonic у терміналі.

Розпізнавач намагається розпізнати мову, навіть коли користувач перестав говорити. Для цього потрібно збільшити значення властивості cognir_instance.energy_threshold.

По суті, це те, наскільки чутливим є розпізнавання до того, коли воно має починатися. Вищі значення означають, що він буде менш чутливим, що корисно, якщо ви знаходитесь у галасливій кімнаті. Це значення залежить від вашого мікрофона або аудіоданих. Немає універсального значення для всіх, але хороші значення зазвичай знаходяться в діапазоні від 50 до 4000.

Також перевірка налаштування гучності мікрофона. Якщо він дуже чутливий, мікрофон може вловлювати багато навколишнього шуму. Якщо він занадто нечутливий, мікрофон може сприймати мову просто шум.

Recognizer неспроможний розпізнати мову відразу після того, як почав слухати вперше. Властивість recognize_instance.energy_threshold, ймовірно, встановлено на значення, яке занадто велике для початку, а потім автоматично знижується за допомогою динамічного налаштування порога енергії. Перш ніж він буде на хорошому рівні, енергетичний поріг настільки високий, що просто вважається фоновим шумом.

Рішення полягає в тому, щоб зменшити цей поріг або заздалегідь викликати recognize_copy. Recognizer не розуміє конкретну мову/діалект. Потрібно встановити мову розпізнавання на мову/діалект. Наприклад, якщо мова/діалект – англійська, краще використовувати як мову «en-GB», а не «en-US». Recognizer зависає на recognize_instance.listen; зокрема, коли він викликає Microphone.MicrophoneStream.read.

Зазвичай це відбувається, коли використовується плата Raspberry Pi, яка сама не має можливості введення звуку. Це призводить до того, що мікрофон за замовчуванням PyAudio, що використовується, просто блокується, коли намагатися його прочитати. Якщо використовується Raspberry Pi, знадобиться звукова карта USB (або мікрофон USB).

Після цього потрібно змінити всі екземпляри Microphone() на Microphone(device_index=MICROPHONE_INDEX), де MICROPHONE_INDEX є

апаратно-залежним індексом мікрофона. Не вказавши device_index буде використовувати мікрофон на замовчуванням.

База даних SQLite 3 зберігання команд в базі є доволі зручним рішенням, Sqlite – це бібліотека C, яка надає легку дискову базу даних, яка не вимагає окремого серверного процесу та дозволяє отримати доступ до бази даних за допомогою нестандартного варіанту мови запитів SQL. Деякі програми можуть використовувати SQLite для внутрішнього зберігання даних. Також можна створити прототип програми за допомогою SQLite, а потім перенести код до більшої бази даних, наприклад PostgreSQL або Oracle [10].

Щоб використовувати модуль, почніть зі створення об’єкта Connection, який представляє базу даних. Тут дані будуть збережені у файлі example.db: import sqlite3; con = sqlite3.connect(example.db). Спеціальне ім’я шляху memory можна надати для створення тимчасової бази даних в RAM. Після встановлення з’єднання створити об’єкт Cursor і викликати його метод execute() для виконання команд SQL (лістинг 2.2).

Лістинг 2.2 – Створення об’єкту Cursor

```
cur = con.cursor()
# Створити таблицю
cur.execute('''CREATE TABLE stocks (date text, trans text, symbol text, qty real,
price real)''')
# Вставити рядок даних
cur.execute("INSERT INTO stocks VALUES ('2006-01-05','BUY','RHAT',100,35.14)")
# Зберегти зміни
con.commit()
# Також можна закрити з'єднання, якщо ми закінчимо з ним.
# Просто переконавшись, що будь-які зміни внесено, інакше вони будуть втрачені.
con.close()
```

Кінець лістингу 2.2

Збережені дані є постійними їх можна перезавантажити в наступному сеансі навіть після перезапуску інтерпретатора Python (лістинг 2.3).

Лістинг 2.3 – Імпорт sqlite3

```
import sqlite3

con = sqlite3.connect('example.db')
cur = con.cursor()
```

Кінець лістингу 2.3

Щоб отримати дані після виконання оператора SELECT, або розглядайте курсор як ітератор, викликайте метод курсора fetchone(), щоб отримати один відповідний рядок, або виклик fetchall(), щоб отримати список відповідних рядків. for row in cur.execute('SELECT * FROM stocks ORDER BY price'): print (row). Буде виведено:

('2006–01–05', 'BUY', 'RHAT', 100, 35.14)

('2006–03–28', 'BUY', 'IBM', 1000, 45.0)

('2006–04–06', 'SELL', 'IBM', 500, 53.0)

('2006–04–05', 'BUY', 'MSFT', 1000, 72.0)

Щоб вставити змінну в рядок запиту, потрібно скористатись заповнювачем у рядку та замінити фактичні значення в запиті, надавши їх як кортеж значень до другого аргументу методу execute(). Інструкція SQL може використовувати один із двох видів заповнювачів: знаки питання (стиль qmark) або іменовані заповнювачі (іменований стиль).

Для стилю qmark параметри мають бути послідовністю. Для названого стилю це може бути екземпляр послідовності або dict. Довжина послідовності повинна відповідати кількості заповнювачів, інакше виникає помилка ProgrammingError. Якщо надано dict, він повинен містити ключі для всіх іменованих параметрів. Будь-які додаткові елементи ігноруються. Приклад зображений на лістингу 2.4.

Лістинг 2.4 – Приклад обох стилів

```
con = sqlite3.connect(":memory:")
cur = con.cursor()
cur.execute("CREATE TABLE lang (ім'я, first_appeared)")
```

```
cur.execute("INSERT INTO VALUE lang (?, ?)", ("C", 1972))

# Стиль qmark, що використовується з executemany():
lang_list = [
    ("Fortran", 1957),
    ("Python", 1991),
    ("Go", 2009),
]
cur.executemany("вставити в значення lang (?, ?)", lang_list)
cur.execute("select * from lang where first_appeared=:year", {"year": 1972})
print(cur.fetchall())
con.close()
```

Кінець лістингу 2.4

Веб сторінка SQLite – документація описує синтаксис і доступні типи даних для підтримуваного діалекту SQL.

sqlite3.paramstyle – рядкова константа, яка вказує тип форматування маркера параметра, очікуваного модулем sqlite3. Вимагається DB-API. Жорстко закодований на qmark. Модуль sqlite3 підтримує стилі параметрів як qmark, так і числових DB-API, оскільки саме це підтримує базова бібліотека SQLite. Однак DB-API не дозволяє декілька значень для атрибута paramstyle.

sqlite3.version – номер версії цього модуля у вигляді рядка. Це не версія бібліотеки SQLite. sqlite3.version_info Номер версії бібліотеки SQLite під час виконання у вигляді рядка. sqlite3.sqlite_version_info – номер версії.

Висновки до розділу 2

Розділ описує собою основні вимоги які потрібно застосувати до розробки голосового помічника, такі як встановлення модулів, бібліотек, збирання інформації щодо запитів голосовому помічнику, поставлено завдання по вибору та інтеграції голосу для асистента а також розробка оптимального та зручного дизайну інтерфейсу користувача. Також після поставлених завдань було розглянуто засоби та методи вирішення поставленого завдання та описано саме встановлення всіх потрібних бібліотек їх налаштування та оптимізація.

РОЗДІЛ 3
РОЗРОБКА ГОЛОСОВОГО ПОМІЧНИКА ДЛЯ СТУДЕНТІВ КАФЕДРИ ІПЗ

3.1 Практична реалізація об’єкта проектування

Для асистента потрібно розробити зручний та інтуїтивно зрозумілий дизайн користувацького інтерфейсу який забезпечити швидкий доступ до основних функцій, отож схема інтерфейсу включає в собі (рис. 3.1):

- назву та логотип;
- вікно з відображенням основних функцій помічника;
- поле яке відображає текстові запити та відповіді;
- кнопки зменшення та збільшення гучності відповіді;
- кнопка активації прослуховування запитів асистентом.

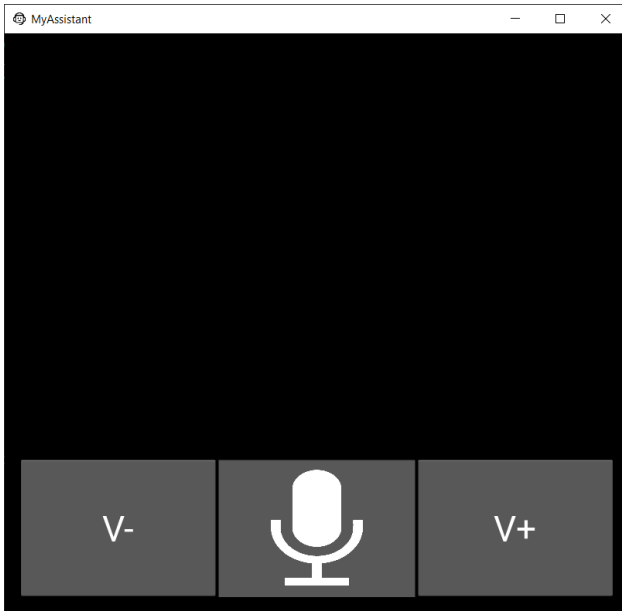

Рисунок 3.1 – Інтерфейс асистента

Для цього проекту буде використовуватись мова програмування Python з версією 3.8. Створення програми буде відповідати документу PEP 8 який описує правила написання коду на мові програмування Python. Також будуть застосовані в проекті бібліотеки та модулі.

Особисті:

– settings – налаштування основних параметрів помічника;

– add_cmd – добавлення власних команд;

– speakengine – відтворення тексту в голос;

– api_ package – пакет з файлами які взаємодіють з різними API.

Вбудовані:

– random – генерація випадкових чисел, букв, послідовностей тощо;

– time – для роботи з часом;

– webbrowser – для відкриття посилань в python;

– math – містить в собі математичні функції та константи;

– json – дозволяє кодувати та декодувати дані в зручному форматі;

– os – бібліотека функцій для роботи з операційною системою;

– sys – прості функцію за допомогою яких можна взаємодіяти напряму з інтерпретатором.

Встановлені:

– pyttsx3 – для відтворення та налаштування голосу;

– pyAudio – для роботи з мікрофоном;

– speech_recognition – для доступу до мікрофона та розпізнавання. В разі використання мікрофону є залежною від PyAudio;

– fuzzywuzzy – для перевірки схожості розпізнаного голосу з командою;

– datetime – для отримання даних точного часу;

– kivy – фреймворк для створення графічного інтерфейсу.

Для подальшого оголошення команд знадобиться встановлення синтезованого голосу RHVoice Volodymyr [1].

Для написання програми знадобиться використати текстовий редактор. Під даний проект було обрано написання коду в програмі Visual studio Code, головними плюсами даного продукту є простота та швидкість роботи під час виконання технічного завдання.

Для початку створення застосовується комбінація клавіш Win+R далі вводимо команду cmd, після чого відкриється вікно командного рядку. Ввівши команду mkdir voice_assistant створиться папка проекту. Далі перейшовши в папку проекту за допомогою команди cd voice_assistant потрібно створити ізольоване середовище розробки. При використанні python з версії 3.6 можна прописати команду python -m venv virt.

Створено каталог virt який містить три папки:

– bin – файли, які взаємодіють з віртуальним середовищем;

– include – C заголовки, компілюючи пакети Python;

– lib – копія версії Python разом з папкою site-packages, в якій встановлена кожна залежність.

Щоб використовувати ресурси середовища, знадобиться перейти в папку bin та активувати їх виконавши команду cd virt/scripts && activate. Після вдалої активації віртуального середовища розпочинається встановлення бібліотек, для їх виготовлення виконується така команда pip install [назва бібліотеки].

Тепер залишилось встановити український синтезований голос. Для цього досить непоганим вибором буде обрати RHVoice Volodymyr. На наступному етапі потрібно створити основну програму main.py та відкрити в Visual studio Code [5]. Далі відкрити папку самого проекту в текстовому редакторі File>Open folder як показано на рисунку 3.2.

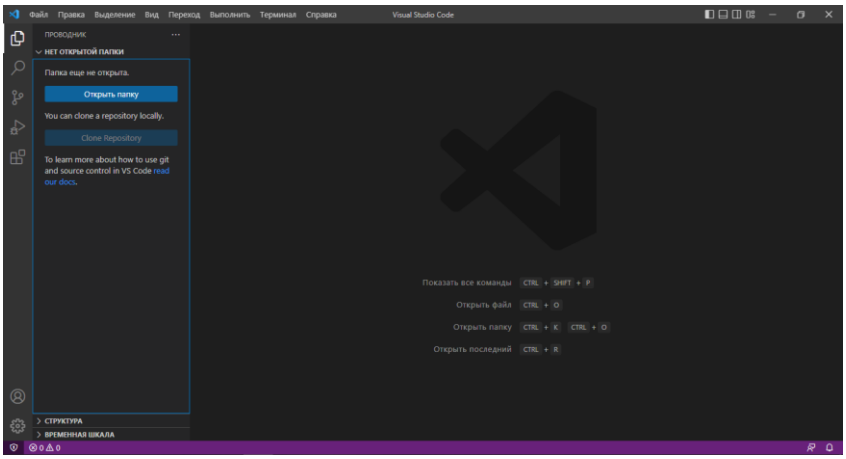

Рисунок 3.2 – Основне вікно редактора Visual studio Code

Для відтворення тексту в голос буде використовується бібліотека Pyttsx3, код функції зображено в лістингу 3.1.

Лістинг 3.1 – Створення та налаштування функції speak()

```
import pyttsx3
speak_engine=pyttsx3.init()
voices=speak_engine.getProperty('voices')
VOICE="HKEY_LOCAL_MACHINE\\SOFTWARE\\Microsoft\\Speech\\Voices\\TokenEnums\\RHVoice
\\Volodymyr"
convers_speed=150
def speak(self):
      speak_engine.setProperty("rate", convers_speed)
      speak_engine.say( self.text )
      speak_engine.runAndWait()
                                    speak_engine.stop()
```

Кінець лістингу 3.1

Import pyttsx3 – команда для підключення бібліотеки pyttsx3, convers_speed – команда відповідає за швидкість відтворення слів у хвилину (150 слів у хвилину) [8]. VOICE – вказує на синтезований голос який потрібно використовувати. В даному випадку буде використовується голос RHVoice Volodymyr. speak(text) – функція, що при виклику відтворює text.

__name__ рівняється __main__ лише в тому випадку якщо main.py запущена як основна програма, а не як модуль. Після запуску команди python main.py буде відтворюватись «Доброго дня».

Для отримання часу потрібно скористатись функцією now_time(). Якщо хвилина буде менша за 10, то поверне значення без нуля. Наприклад «Зараз тринадцять п’ять». Тому потрібно добавити нуль для коректного відтворення. Синтезований голос відтворить рядок «Зараз 13:05» як «Зараз тринадцять нуль п’ять». Код функції описано у лістингу 3.2.

Лістинг 3.2 – Отримання часу string

```
@staticmethod
    def now_time() -> str:
```

```
                                    now = f"Зараз
  {datetime.datetime.now().hour}:{datetime.datetime.now().minute}"
        return now
```

Кінець лістингу 3.2

На цьому етапі голосовий помічник може розмовляти. Для того, щоб переданий текст був відтворений потрібно викликати функцію speak(). Далі буде використовуватися мікрофон та розпізнавання голосу для розмови з помічником.

Для цього використовується бібліотека speech_recognition. Ця бібліотека містить в собі такий клас Microphone() – який дозволить використати мікрофон та Recognizer() – За допомогою якого можна здійснювати записування голосу та розпізнати його [7]. Виклик функції наведено в лістингу 3.3.

Лістинг 3.3 – Виклик функції speak()

```
def voice_to_txt(voice):
    text = r.recognize_google(voice, language=”uk-UK”).lower()
    return text
r = sr.Recognizer()
micro = sr.Microphone()
class Assistant:
    def cmd_call(self):
        try:
            print(self)
            with micro as sourse:
                print("Розпізнано: " + self.text)
            self.recognize_cmd()
        except sr.UnknownValueError:
            speak("Голос не розпізнано!")
```

Кінець лістингу 3.3

Отож, створюємо класс Assistant. Після цього буде увімкнено мікрофон та помічник буде слухати далі буде виконуватись виключення sr.UnknownValueError якщо голос не було розпізнано. Також буде виконуватися розпізнавання голосу і повернення змінної text в вигляді тексту в нижньому регістрі.

Для розробки наступного функціоналу, а саме можливості отримання розкладу занять та іспитів потрібно звернутись до сайту університету та за допомогою командного рядка в розділі мережа сформувати URL посилання розкладу занять [2]. Даний процес зображено нижче (рис. 3.3).

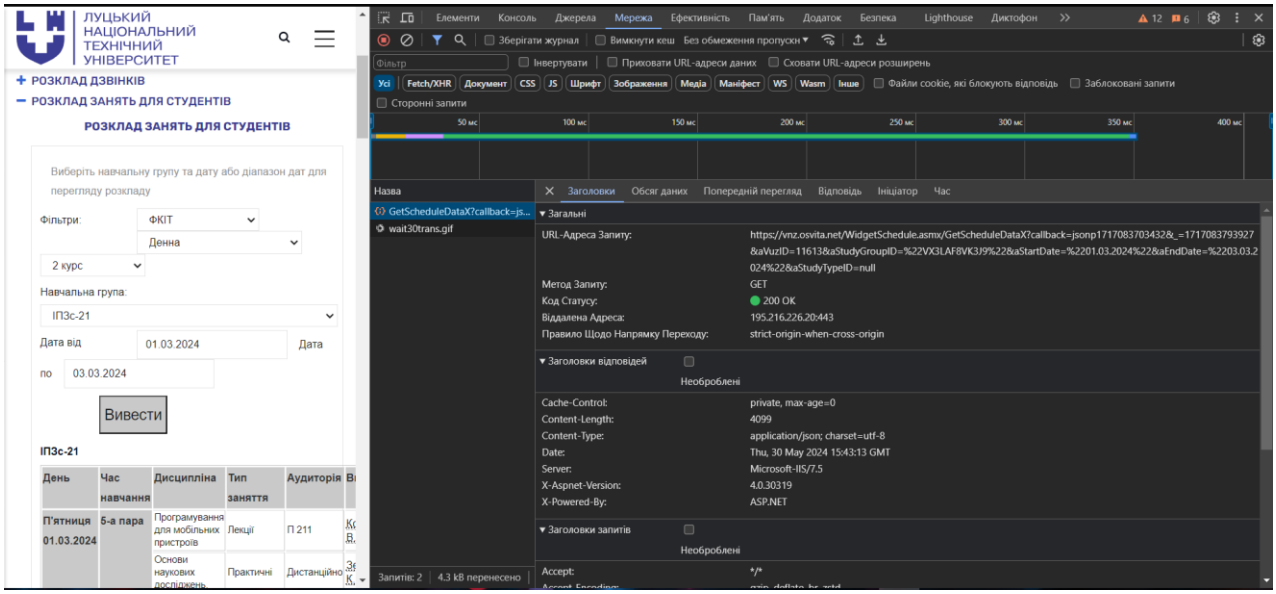

Рисунок 3.3 – Процес отримання URL посилання

Для отримання цього посилання потрібно на сторінці в розділі «Розклад занять для студентів» обрати свій факультет, форму навчання, курс, групу, та проставити дати. Далі повертаємось до написання функціоналу в якому будуть використовуватись функції get_date, яка генерує дату форматом «dd.mm.yyyy», де n це кількість днів від поточної дати, extract_json_from_jsonp функція яка витягує JSON з JSONP рядка, що в свою чергу дозволяє отримати звіт з даними розкладу. Код функції описаний у лістингу 3.4.

Лістинг 3.4 – Запис функцій get_date та extract_json_from_jsonp

```
def get_date(n=0):
    date = datetime.now() + timedelta(days=n)
    yyyy = date.year
    mm = date.month
    dd = date.day

    if dd < 10:
```

```
        dd = '0' + str(dd)
    if mm < 10:
        mm = '0' + str(mm)

    formatted_date = f"{dd}.{mm}.{yyyy}"
    return formatted_date

def extract_json_from_jsonp(jsonp_string):
    match = re.search(r'\((.*?)\)', jsonp_string)
    if match:
        json_data = match.group(1)
        schedule_data = json.loads(json_data)
        print(schedule_data)
        return schedule_data
    else:
        print("Error: JSONP format not recognized.")
```

Кінець лістингу 3.4

Наступна функція get_schedule має в собі два значення startDate та endDate за допомогою яких задається період для розкладу, далі формується URL і параметри HTTP запиту, виконується GET запит на сервер за допомогою requests.get далі додається заголовок User-Agent, щоб отримати запит від браузера в разі успіху (статус-код 200), витягується JSONP рядок з відповіді та передає його в extract_json_from_jsonp та повертаються розпарсені JSON-дані, але в разі того якщо запит не вдалий друкується повідомлення про помилку з відповідним статус кодом і голосовий асистент промовляє, що немає даних схоже пар немає. Код функції описано у лістингу 3.5.

Лістинг 3.5 – Написання функції get_schedule()

```
def get_schedule(startDate=get_date(), endDate=get_date(7)):
    url = "https://vnz.osvita.net/WidgetSchedule.asmx/GetScheduleDataX"
    params = {
        'callback': 'jsonp1717010024814',
        '_': '1717010054860',
        'aVuzID': '11613',
        'aStudyGroupID': '"VX3LAF8VK3J9"',
        'aStartDate': '"01.03.2024"',
        'aEndDate': '"01.03.2024"',
        'aStudyTypeID': 'null'
    }
```

```
    headers = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3'
    }
response = requests.get(url, params=params, headers=headers)
    print(response.url)  # Додатково можна роздрукувати остаточний URL для
перевірки
    print(response.status_code)
    if response.status_code == 200:
        jsonp_string = response.text
        print(jsonp_string)
        return extract_json_from_jsonp(jsonp_string)
    else:
        print(f"Error: Unable to fetch data. Status code: {response.status_code}")
```

Кінець лістингу 3.5

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Під час розробки голосового помічника було проведено ручне тестування в якому тестувались кнопки гучності та увімкнення мікрофону та правильність відповіді на команди і тестування різних голосів якими може розмовляти голосовий асистент.

Тестування та налагодження є невід’ємною частиною будь-якого проекту, воно включає в собі виявлення дефектів які потрібно виправити у системі, перевірку відповідності, що дозволяє зрозуміти чи система відповідає вимогам і підтвердження якості, коли система працює відповідно до очікувань користувача, після чого проводиться аналіз помилок та їх виправлення.

В даному проекті під час розробки було виявлено декілька помилок та розроблено спосіб їх виправлення. Під час розробки голосового асистента були виявлені наступні помилки:

– швидкість вимови була надто нестабільною, через що голосовий помічник неправильно вимовляв деякі слова;

– неправильна фільтрація голосових команд через дану помилку голосовий помічник не міг зрозуміти слова та те, що від нього хоче користувач.

Для виправлення помилки швидкості було написано метод який включає в собі налаштування швидкості та гучності мовлення, цей метод містить в собі conversation_speed команду, що встановлює швидкість вимови відповідей голосового асистента далі voice встановлює голос для синтезу, а наступна команда volume дає змогу встановлювати гучність мовлення.

Після чого метод speak_engine.say передає озвучування тексту, а далі викликаються методи runAndWait, щоб зачекати закінчення озвучування відповіді і stop який зупиняє двигун системи синтезу мови. Даний фрагмент коду зображено в лістингу 3.9.

Лістинг 3.9 – Фрагмент коду з усуненням помилки мовлення

```
def __call__(self, text):
        self.speak_engine.setProperty('rate', CONVERSATION_SPEED)
        self.speak_engine.setProperty("voice", self.voice)
        self.speak_engine.setProperty("volume", self.volume)
        self.speak_engine.say(text)
        self.speak_engine.runAndWait()
        self.speak_engine.stop()
```

Кінець лістингу 3.9

Наступна помилка яку потрібно виправити це некоректне фільтрування мови, щоб усунути дану несправність потрібно написати код в якому буде реалізовано підключення до бази даних, сам функціонал фільтрації голосу та перевірки команд в таблицях (лістинг 3.10).

Лістинг 3.10 – Фрагмент коду з усуненням помилки мовлення

```
 def recognize_command(self):

        sql = sqlite3.connect("commands.db")
        self.filtered_voice = self.filtering()

        for cmd in sql.execute("SELECT * FROM questions"):
if fuzz.ratio(cmd[0], self.filtered_voice) > SIMILARITY:
                MyApp.set_label_text(cmd[1], "left")
                speak(cmd[1])
                sql.close()
```

```
    return 0
```

Кінець лістингу 3.10

В цьому коді виконується підключення до бази даних за допомогою команди commands.db а фільтрація голосу відбувається за допомогою filtered_voice, яка отримує відфільтровану голосову команду через метод self.filtered_voice, далі здійснюється перевірка у таблиці бази даних questions та перебираються всі команди і якщо схожість fuzz.ratio між командою з таблиці та self.filtered_voice перевищує значення similarity то викликається метод MyApp.set_label_text який встановлює текст відповіді в інтерфейсі користувача а відповідь на команду виконується за допомогою функції speak та при завершенні виконання закривається з’єднання з базою даних і функція повертає нуль.

3.3 Розробка бази даних порядок розділів

Для того, щоб помічник працював і виконував поставлені на його задачі потрібно створити файл add_cmd.py далі імпортувати всі потрібні бібліотеки для підключення бази даних SQLite, створення таблиць з командами, внесення та виведення команд до бази даних а також створення графічного інтерфейсу програми для додавання команд в таблиці [10]. Імпортування бібліотек наведено в лістингу 3.6.

Лістинг 3.6 – Імпортування потрібних бібліотек

```
import sqlite3
import sys

from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.gridlayout import GridLayout
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
from kivy.core.window import Window
from kivy.uix.textinput import TextInput
```

```
db = sqlite3.connect('commands.db')
sql = db.cursor()

sql.execute("""CREATE TABLE IF NOT EXISTS questions (
command TEXT,
answer TEXT
)""")

sql.execute("""CREATE TABLE IF NOT EXISTS URL (
command TEXT,
url TEXT
)""")

sql.execute("""CREATE TABLE IF NOT EXISTS execute_cmd (
command TEXT,
link TEXT
)""")
```

Кінець лістингу 3.6

Цей код не тільки імпортує бібліотеки але й створює таблиці в базі даних такі як questions яка зберігає в собі команди та відповіді на них, URL таблиця, яка містить в собі запит та url посилання на інтернет ресурси, і execute_cmd в якій прописано команди, що має виконувати голосовий помічник наприклад, закрити діалог і вимкнутись або проговорити розклад занять. Структура таблиць бази даних зображено нижче (рис. 3.4).

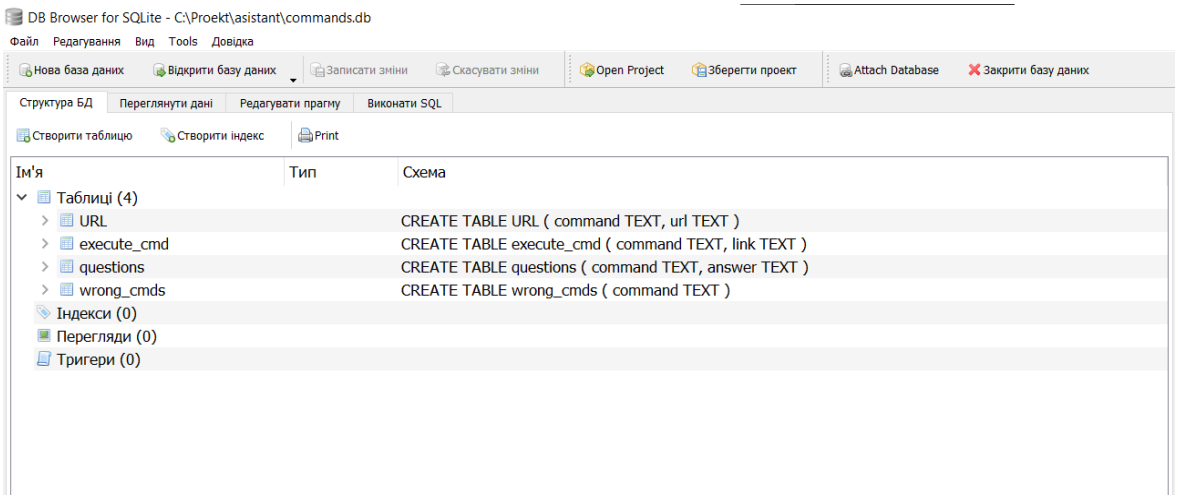

Рисунок 3.4 – Структура бази даних

Наступним етапом потрібно використати функцію add_cmd(instance), яка додає в таблицю команду та відповідь та delete(), яка в свою чергу видаляє команди з таблиці, код функції описано у лістингу 3.7.

Лістинг 3.7 – Додавання функцій add_cmd(instance) та delete()

```
def add_cmd(instance):
    print(instance.text)
    cmd = myApp.text_cmd.text
    answer = myApp.text_response.text
    sql.execute("INSERT INTO " + instance.text + " VALUES (?,?)", (cmd, answer))
    db.commit()
    myApp.lbl.text = f"Додано команду: {myApp.text_cmd.text}"
    myApp.text_cmd.text = ""
    myApp.text_response.text = ""

def delete():
    table = input("Select a table: ")
    elem = input("Select a question: ")
    sql.execute(f"DELETE from {table} where command = '{elem}'")
    print("The questions has been deleted!")
    db.commit()
```

Кінець лістингу 3.7

Після створення функцій внесення та вилучення команд з таблиць бази даних потрібно розробити графічний інтерфейс програми для цього буде використано функцію AddCommandApp(App). Приклад використання даної функції наведено в лістингу 3.8.

Лістинг 3.8 – Додавання функцій AddCommandApp(App)

```
class AddCommandApp(App):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.bl = BoxLayout(orientation="vertical", padding=20)
        self.gl = GridLayout(cols=3, spacing=3)
        self.text_cmd = TextInput(size_hint=(1, .5))
        self.text_response = TextInput(size_hint=(1, .5))
        self.lbl_cmd = Label(text="Команда", font_size=20, halign="left",
valign="center", size_hint=(1, .5),
                             text_size=(Window.size[0] - 40, Window.size[1] / 20))

        self.lbl_responce = Label(text="Відповідь",
```

```
                                font_size=20,
                                halign="left",
                                valign="center",
                                size_hint=(1, .5),
                                text_size=(Window.size[0] - 40, Window.size[1] /
20))
        self.lbl = Label(text="Добавте команду",
                         font_size=20,
                         halign="left",
                         valign="center",
                         size_hint=(1, .5),
                         text_size=(Window.size[0] - 40, Window.size[1] / 20))
        self.lbl1 = Label(text="Виберіть таблицю:",
                          font_size=20,
                          halign="left",
                          valign="center",
                          size_hint=(1, .5),
                              text_size=(Window.size[0] - 40, Window.size[1] / 20))
```

Кінець лістингу 3.8

Клас AddCommandApp(App) дозволяє створити додаток за допомогою якого можна записувати нові команди та відповіді на них до бази даних та при потребі вилучити їх. Сам графічний інтерфейс складається з текстових полів та кнопок вибору таблиці в яку буде записано команду та дію. Вигляд цього інтерфейсу зображено на рисунку 3.5.

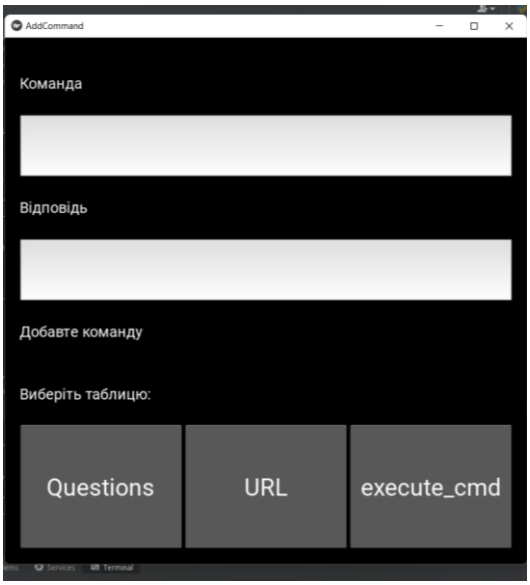

Рисунок 3.5 – Вікно програми яка додає команди в базу даних

3.4 Розробка документації для програмного продукту

Для встановлення та використання голосового асистента потрібно в залежності від операційної системи обрати версію Python, в даному проекті використовувалась версія 3.8. Беручи до уваги дану версію можна сформувати наступні вимоги до операційних систем. Дані вимоги описані в таблиці 3.1.

Таблиця 3.1 – Вимоги до ПЗ

Операційна система	Windows	macOS	Linux
Версія	Windows 10 Windows Server 2016 або новіше	macOS 10.9 (Mavericks) або новіше	Розповсюдження, що підтримує glibc 2.17 або новіше Ubuntu 16.04 LTS, Fedora 28, Debian 9 і новіше, тощо)
Пам’ять	Мінімум 512 МБ оперативної пам’яті (рекомендується 1 ГБ)		
Місце на диску	Мінімум 100 МБ для встановлення Python (додаткове місце для встановлення бібліотек та залежностей через pip)		

Щоб асистент міг працювати коректно, необхідно імпортувати потрібні бібліотеки такі як, sqlite, speech recognition, pyttsx3, після встановлення даних модулів голосовий помічник може розмовляти, розпізнавати голос та брати інформацію з створеної бази даних. Для того, щоб змінити голос асистента потрібно знайти і інтегрувати синтезований голос, а також налаштувати швидкість мовлення так, що слова були зрозумілими.

Для того, щоб скористатися всіма можливостями голосового помічника необхідно відкрити командний рядок та написати команду python main.py, ця команда запускає програму та з’являється вікно голосового помічника у якому відображається поле з командами та відповідями, а також знизу є три кнопки які відповідають за збільшення та зменшення гучності голосу та кнопка мікрофону після натискання якої можна задати запитання асистенту.

Для внесення змін в базу даних та додавання нових команд необхідно скористатися командою python cmd add pay після введення якої відкривається вікно з рядками для введення запитання чи дії, яку повинен виконати асистент та другий рядок внесення відповіді на дану команду, після додавання всіх

даних в вибрані поля потрібно вибрати одну із трьох таблиць, які відображаються в даному вікні в формі кнопок, щоб голосовий помічник зміг дати звичайну відповідь на запитання потрібно натиснути кнопку questions.

Якщо потрібно по запиту користувача відкрити посилання потрібно звернутися до таблиці URL, і наостанок для того, щоб виконувалась певна дія асистентом до прикладу це може бути розповісти розклад занять чи вимкнути програму, потрібно натиснути кнопку execute_cmd.

Висновки до розділу 3

В даному розділі було описано практичну реалізацію голосового помічника, що містило в собі опис процесу написання коду програми, та демонстрацію застосування мови програмування python, пояснювалось реалізації окремих алгоритмів та методів написання коду. Після опису етапів створення голосового асистента було проведено тестування та налагодження системи в якому були виявлені помилки які стосувалися коректної роботи голосового модуля після чого були вдало виправлені та налагоджені.

Було розроблено модулі add_cmd.py, settings.py. add_cmd.py – модуль був розробленим для додавання власних команд. settings.py – модуль для заміни параметрів голосового помічника: імені, голосу та мови розпізнавання. Також розглянуті питання щодо використання технологій таких як: бібліотеки для використання мікрофону, розпізнавання голосу та розробки графічного інтерфейсу користувача.

Також описувалася реалізація розробки бази даних для помічника, спосіб виведення інформації з бази даних внесення змін та додавання нового функціоналу.

Після чого було описано детальну документацію, яка пояснює як правильно користуватися голосовим асистентом та вносити зміни до функціоналу.

ВИСНОВКИ

Голосові помічники можна використовувати різними способами. Вони були створені, щоб люди не витрачали свій зайвий час на прості повсякденні завдання. Функції голосового помічника можуть бути досить широкими.

Отже створений голосовий помічник може виконувати наступні задачі:

– переходити за посиланням відкривши в браузері сторінку;

– відповідати на певні поставлені питання;

– розповісти який час;

– за допомогою команди «Бувай» – голосовий помічник попрощається з вами та завершить роботу.

Успішно виконані всі поставлені завдання на кваліфікаційну роботу бакалавра:

– були проаналізовані існуючі голосові асистенти такі як Cortrana, Siri та Google Assistant. Розглянуті їх принципи роботи та визначені існуючі переваги та недоліки;

– проведений аналіз вимог користувачів до голосового помічника, та визначені вимоги до розробленого асистента. Голосовий помічник має записувати голосову команду та відображати результат. Також розроблений додаток має бути легко розширюваним за допомогою принципу Open source.

– визначено функції які потрібні студентам і які запити вони найчастіше будуть використовувати. Розроблено діаграму варіантів використання для голосового помічника;

– здійснено вибір методів та алгоритмів розробки голосового помічника. Голосовий помічник був розроблений на мові програмування Pyton. Додатково були обрані такі бібліотеки та фреймворки такі як: pyttsx3, sqlite3, pyaudio, kivy, speech recognition;

– розроблений інтерфейс асистента, який є інтуїтивно зрозумілим і зручним для студентів. Реалізовано функції відповіді на запитання студентів,

включаючи доступ до бази знань або пошукової системи. Забезпечено правильне розпізнавання команд. Додано можливість виконувати команди з відкриття сайтів та додатків. Підключена база даних SQLite для збереження команд голосового помічника;

– проведене ручне тестування асистента та тестування роботи асистента з різними голосовими модулями. Здійснено вдосконалення голосового помічника на різних пристроях, щоб забезпечити правильне розпізнавання мови. Виконано тестування помічника на різних сценаріях використання;

– написана докладна документація, включаючи інструкції з використання, встановлення та налаштування асистента. Сформульовані системні вимоги до пристрою, для успішного встановлення та запуску голосового помічника.

Дана робота спрямована на вирішення проблеми зручності повсякденному життю користувачів так як для невеличких завдань які пов’язані з повідомленням користувача про якусь дію відповідь на запитання чи відкриття інтернет-ресурсу голосовий асистент справляється на відмінно. Результат роботи можуть бути використані як основа для подальшого розвитку та вдосконалення комп’ютеризованих рішень у в більшості сфер так як дану технологію можна розвивати та додавати все більше нового функціоналу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Каталог готових голосів – RHVoice Lab. Про проект – RHVoice Lab. URL: https://rhvoice.com/voices/ (дата звернення: 05.03.2024).

2. Розклад занять та іспитів ЛНТУ | ЛНТУ | Луцький національний технічний університет. Головна | ЛНТУ | Луцький національний технічний університет. URL: https://lntu.edu.ua/uk/studentu–0/navchannya/rozklad–zanyat–ta–ispytiv–lntu (дата звернення: 7.03.2024).

3. Cortana та конфіденційність – Підтримка від Microsoft. Microsoft Support. URL: https://support.microsoft.com/uk–ua/topic/cortana–та–конфіденційність–47e5856e–3680–d930–22e1–71ec6cdde231 (дата звернення: 08.03.2024).

4. Google Assistant, your own personal Google. Assistant. URL: https://assistant.google.com/ (дата звернення: 09.03.2024).

5. Microsoft. Visual Studio Code – Code Editing. Redefined. Visual Studio Code – Code Editing. Redefined. URL: https://code.visualstudio.com (дата звернення: 05.04.2024).

6. News API. News API – Search News and Blog Articles on the Web. URL: https://newsapi.org/ (дата звернення: 06.04.2024).

7. PyPI The Python Package Index. PyPI. URL: https://pypi.org/ (дата звернення: 09.04.2024).

8. Pyttsx3 – Text–to–speech x–platform – pyttsx3 2.6 documentation. pyttsx3 – Text–to–speech x–platform – pyttsx3 2.6 documentation. URL: https://pyttsx3.readthedocs.io/en/latest/ (дата звернення: 15.04.2024).

9. Siri. Apple. URL: https://www.apple.com/siri/ (дата звернення: 20.04.2024).

10. SQLite Home Page. SQLite Home Page. URL: https://www.sqlite.org/index.html (дата звернення: 04.05.2024).

11. Weather API – OpenWeatherMap. Current weather and forecast – OpenWeatherMap. URL: https://openweathermap.org/api (дата звернення: 12.05.2024).

12. Welcome to Python.org. Python.org. URL: https://www.python.org (дата звернення: 15.05.2024).

ДОДАТКИ

Додаток А

Лістинг програми main.py

```
from speak_engine import SpeakEngine
from Services import get_subjects
from config import *

from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.gridlayout import GridLayout
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label

from speech_recognition import Microphone, Recognizer, UnknownValueError,
RequestError
from fuzzywuzzy import fuzz
import sqlite3
import webbrowser as wb
import json
import requests
import datetime
import sys


speak = SpeakEngine(VOICE_PATH)


class Assistant:

    def __init__(self):
        self.filtered_voice = None
        self.__recognized = ""
        self.micro = Microphone()
        self.rec = Recognizer()
        self.audio: Recognizer = self.rec

    def on_micro(self):
        with self.micro as source:
            print("start listening...")
            self.rec.adjust_for_ambient_noise(source)
            self.audio = self.rec.listen(source, phrase_time_limit=4)
            print("End listening!")
            return self.audio

    def call_assistant(self, _):
        """Call when clicked button. _ – instance, call microphone power and
writing to self.audio recognize"""
        self.on_micro()
        try:
```

```
            self.__recognized = self.rec.recognize_google(self.audio, language="uk-
UK").lower()
            MyApp.set_label_text(self.__recognized, "right")

        except UnknownValueError:
            self.__recognized = UNKNOWN_VALUE_ERROR

        except RequestError:
            self.__recognized = "RequestError"

        finally:
            self.recognize_command()

    def filtering(self):
        if self.__recognized == UNKNOWN_VALUE_ERROR:
            speak(UNKNOWN_VALUE_ERROR)
            return 0

        if self.__recognized == "RequestError":
            MyApp.set_label_text("Network error", "left")
            return 0

        words = self.__recognized.split()

        for elem in words:
            if elem in IGNORE_WORDS:
                self.__recognized = self.__recognized.replace(elem, "")

        if self.__recognized.startswith(" "):
            del self.__recognized[0]
        if self.__recognized.endswith(" "):
            del self.__recognized[-1]

        return self.__recognized

    def recognize_command(self):

        sql = sqlite3.connect("commands.db")
        self.filtered_voice = self.filtering()

        for cmd in sql.execute("SELECT * FROM questions"):
            if fuzz.ratio(cmd[0], self.filtered_voice) > SIMILARITY:
                MyApp.set_label_text(cmd[1], "left")
                speak(cmd[1])
                sql.close()
                return 0

        for cmd in sql.execute("SELECT * FROM URL"):
            if fuzz.ratio(cmd[0], self.filtered_voice) > SIMILARITY:
                MyApp.set_label_text(cmd[1], "left")
```

```
                wb.open(cmd[1])
                sql.close()
                return 0

        for cmd in sql.execute("SELECT * FROM execute_cmd"):
            services = {
                "get_usd": Service().get_usd,
                "now_time": Service().now_time,
                "get_schedule": Service().get_schedule,
                "exit": Service().exit
            }
            if fuzz.ratio(cmd[0], self.filtered_voice) > SIMILARITY:
                for key, value in services.items():
                    if cmd[1] == key:
                        speak(value())
                        sql.close()
                        break
                return 0

    @staticmethod
    def volume_plus(_):
        speak.volume += 0.1
        print(speak.volume)
        speak("Добре")

    @staticmethod
    def volume_minus(_):
        speak.volume -= 0.1
        print(speak.volume)
        speak("Добре")


class Service:

    @staticmethod
    def get_schedule() -> str:

        schedule_data = get_subjects.get_schedule()
        if not schedule_data:
            return "Не вдалося отримати розклад."

        disciplines = ""
        for item in schedule_data['d']:
            disciplines += f"{item['study_time']} {item['discipline']} "

        if disciplines:
            return "Сьогодні: " + disciplines
        else:
            return "Сьогодні немає пар."
```

```
    @staticmethod
    def now_time() -> str:
        return f"Зараз
{datetime.datetime.now().hour}:{datetime.datetime.now().minute}"

    @staticmethod
    def get_usd() -> str:
        content =
requests.get("https://api.privatbank.ua/p24api/pubinfo?json&exchange&coursid=5").co
ntent
        data = json.loads(content)
        for elem in data:
            if elem["ccy"] == "USD":
                price = elem["buy"]
                price = price.split(".")
                return f"зараз по {price[0]} гривень {price[1][:2]} копійок"

    @staticmethod
    def exit():
        speak(EXITING)
        sys.exit()


assistant = Assistant()


class WindowAssistantApp(App):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.bl = BoxLayout(orientation="vertical", padding=20)
        self.gl = GridLayout(cols=3, spacing=3)

        self.labels = [Label(text="", halign="left", valign="bottom",
text_size=(400 - 40, 15),
                             size_hint=(1, .3), font_size=16) for _ in range(10)]

        self.b1 = Button(text="V-",
                         on_press=assistant.volume_minus,
                         border=(10, 10, 10, 10),
                         size_hint=(1, .4),
                         font_size=46)
        self.b2 = Button(on_press=assistant.call_assistant,
                         background_normal="images/microphone.jpg",
                         border=(10, 10, 10, 10),
                         size_hint=(1, .4))
        self.b3 = Button(text="V+",
                         on_press=assistant.volume_plus,
                         border=(10, 10, 10, 10),
                         size_hint=(1, .4),
                         font_size=46)
```

```
    def set_label_text(self, text, align="left"):
        for i in range(len(self.labels) - 1):
            self.labels[i].text, self.labels[i].halign = self.labels[i + 1].text,
self.labels[i + 1].halign

        self.labels[-1].text, self.labels[-1].halign = text, align

    def build(self):
        self.title = APP_TITLE
        self.icon = APP_ICON

        for elem in self.labels:
            self.bl.add_widget(elem)

        self.gl.add_widget(self.b1)
        self.gl.add_widget(self.b2)
        self.gl.add_widget(self.b3)

        self.bl.add_widget(self.gl)

        return self.bl


if __name__ == '__main__':
    MyApp = WindowAssistantApp()
    MyApp.run()
```