Форма № Н-9.02

Міністерство освіти і науки України

Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»

(повне найменування вищого навчального закладу)

Відділення інформаційних технологій, менеджменту, туризму і підготовки іноземних громадян

(назва відділення)

Циклова комісія комп'ютерної інженерії

(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра

(освітньо-професійного ступеня)

на тему: **Розробка програмного забезпечення «Голосовий асистент»**

Виконав: студент IV курсу, групи КІ-418

Спеціальності 123 Комп'ютерна інженерія

(шифр і назва спеціальності)

_____ Руслан КАДИЛО

(ім'я та прізвище)

Керівник Ігор ТХІР

(ім'я та прізвище)

Рецензент _____

(ім'я та прізвище)

Тернопіль – 2024

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
імені ІВАНА ПУЛЮЯ»

Відділення **інформаційних технологій, менеджменту, туризму та підготовки іноземних громадян**
Циклова комісія **комп'ютерної інженерії**
Освітньо-професійний ступінь **фаховий молодший бакалавр**
Освітньо-професійна програма: **Обслуговування комп'ютерних систем і мереж**
Спеціальність: **123 Комп'ютерна інженерія**
Галузь знань: **12 Інформаційні технології**

ЗАТВЕРДЖУЮ
Голова циклової комісії
комп'ютерної інженерії
_____ Андрій ЮЗЬКІВ
"03" квітня 2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кадилові Руслану Романовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи __ Розробка програмного забезпечення «Голосовий асистент»

керівник роботи __Тхір Ігор Любомирович
(прізвище, ім'я, по батькові)

Затверджені наказом ВСП «Тернопільський фаховий коледж ТНТУ імені Івана Пулюя» від 02.04.2024 р №4/9-157.

2. Строк подання студентом роботи: 17 червня 2024 року.

3. Вихідні дані до роботи: блок-схема програми, завдання на проєктування, вибрана мова програмування – Python, тип додатку – Desktop, сумісний з операційною системою – Windows.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Загальний розділ. Розробка технічного та робочого проєкту. Спеціальний розділ. Економічний розділ. Охорона праці, техніка безпеки та екологічні вимоги.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
- блок-схема;
- фрагмент коду (функція);
- структурна схема;
- таблиця техніко-економічних показників.

6. Консультанти розділів роботи

Розділ	Ім'я, прізвище та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний розділ	Богдана МАРТИНЮК викладач		
Охорона праці, техніка безпеки та екологічні вимоги	Володимир ШТОКАЛО викладач		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	04.04	
2	Збір і узагальнення інформації	13.05	
3	Написання першого розділу	20.05	
4	Розробка технічного та робочого проекту	27.05	
5	Написання спеціального розділу	3.06	
6	Розрахунок економічної частини	5.06	
7	Написання розділу охорони праці	7.06	
8	Виконання графічної частини	10.06	
9	Оформлення проекту	12.06	
10	Погодження нормоконтролю	14.06	
11	Попередній захист роботи	17.06	
12	Захист кваліфікаційної роботи		

7. Дата видачі завдання: 04 квітня 2024 року

Студент _____ Руслан КАДИЛО
(підпис) (ім'я та прізвище)

Керівник роботи _____ Ігор ТХІР
(підпис) (ім'я та прізвище)

АНОТАЦІЯ

В кваліфікаційній роботі було поставлено завдання по розробці програмного забезпечення «Голосовий асистент». В ході роботи було сформовано технічне завдання, де визначені основні напрямки та спеціальні вимоги для правильного та ефективного функціонування програмного продукту.

Програма виконує роль голосового асистента, який точно розпізнає голосові команди користувача та швидко їх виконує. Для активації асистента користувач повинен звернутися до нього по імені, після чого асистент активується і почне прослуховувати команду.

У програмі вбудовані системні команди зі заздалегідь визначеною реалізацією. Звичайний користувач також має можливість додавати або видаляти власні команди, вказуючи ключове слово та шлях до виконуваного файлу. Виконувані файли використовують такі бібліотеки**: Speech Recognition, time, pygame, webbrowser, os, sys, random, threading**.

Програма розроблена з урахуванням потреб різних користувачів, забезпечуючи простий та інтуїтивно зрозумілий інтерфейс. Вона призначена для полегшення виконання рутинних задач, таких як відкриття програм, веб-сайтів, запуск мультимедіа та багато іншого. Гнучкість у налаштуванні команд дозволяє користувачам адаптувати асистента до своїх індивідуальних потреб.

Крім основних функцій, програмний асистент може працювати у фоновому режимі, не перешкоджаючи іншим процесам на комп'ютері. Завдяки використанню багатозадачності (threading), асистент може одночасно обробляти кілька команд, що підвищує його ефективність.

Таким чином, даний голосовий асистент є корисним інструментом для автоматизації повсякденних задач, що підвищує продуктивність як досвідчених користувачів, так і новачків.

ЗМІСТ

ВСТУП 7

1 ЗАГАЛЬНИЙ РОЗДІЛ 8

1.1 Аналіз технічного завдання 8
1.2 Технічне завдання 10
1.2.1 Найменування та область застосування 10
1.2.2 Призначення розробки 11
1.2.3 Вимоги до програмного забезпечення 11
1.2.3.1 Вхідні дані 11
1.2.3.2 Вихідна інформація 12
1.2.4 Часові характеристики 12
1.2.5 Вимоги до надійності 13
1.2.6 Умови експлуатації 13
1.2.7 Вимоги до програмної документації 13
1.2.8 Стадії та етапи розробки 14
1.2.9 Порядок контролю та прийому 14
2 РОЗРОБКА ТЕХНІЧНОГО ПРОДУКТУ 15

2.1 Постановка задачі на розробку програмного забезпечення 15
2.2 Опис та обґрунтування вибору методу організації вхідних та вихідних даних 15
2.2.1 Вхідні дані 15
2.2.2 Вихідні дані 16
2.3 Опис методів реалізації функцій програми 17
2.3.1 Функція main 17
2.3.2 Функція main_program 22
2.3.3 Функція command 23
2.3.4 Функція filter_cmd 25
2.3.5 Функція tts 26
2.3.6 Функція read_file 26
2.3.6 Функція respond 27
2.3.7 Функція add_cmd 27
2.3.8 Функція del_cmd 28
2.3.9 Функція cmd_clear 29
2.3.10 Функція hide_widget 30
2.3.11 Функція come_widget 30
2.4 Визначення інформаційних зв’язків програмних компонентів 31
2.5 Написання текстів програми 32
2.6 Тестування та налагодження програми 33

3 СПЕЦІАЛЬНИЙ РОЗДІЛ 37
3.1 Архітектура та дизайн системи 37
3.2 Розпізнавання та обробка мови 40
3.2.1 Алгоритми розпізнавання мовлення 40
3.2.2 Методи обробки природної мови (NLP) 43
3.2.3 Інтеграція з розпізнаваннями мовлення та NLP-сервісами 45
3.2.4 Оптимізація точності та швидкості реакції асистента 48
3.3 Інтерфейс та взаємодія з користувачем 49
3.3.1 Принципи проектування користувацького інтерфейсу 49
3.3.2 Розробка голосового інтерфейсу користувача (VUI) 51
3.3.3 Тестування взаємодії з користувачами 53
3.3.4 Забезпечення доступності та інклюзивності голосового асистента 55
4. ЕКОНОМІЧНИЙ РОЗДІЛ 58
4.1 Визначення стадій технологічного процесу та загальної тривалості розробки та реалізації програмного забезпечення «Голосовий асистент» 58
4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи 59
4.3 Розрахунок витрат на електроенергію 62
4.4 Розрахунок суми амортизаційних відрахувань 63
4.5 Обчислення накладних витрат 63
4.6 Складання кошторису витрат та визначення собівартості НДР 64
4.7 Розрахунок ціни НДР 65
4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень 65
5 ОХОРОНА ПРАЦІ, ТЕХНІКИ БЕЗПЕКИ ТА ЕКОЛОГІЧНІ 67
5.1 Комплекс заходів, спрямованих на боротьбу із шумом 67
5.2 Нервово-емоційна напруженість праці розробника ПЗ 70
5.3 Організація безпечної поведінки працівника в процесі праці 75
ВИСНОВКИ 78
ПЕРЕЛІК ПОСИЛАНЬ 79
ДОДАТКИ 80
Додаток А - Код програми 80

ВСТУП

Час йде, технології не стоять на місці і стрімко розвиваються. Люди завжди хочуть якомога більше спростити собі життя. Автоматизувати легкі рутинні справи чи полегшити складні. Одна з таких технологій – голосовий помічник.

Голосовий помічник — це програма, яка використовує розпізнавання голосу та алгоритми обробки мови, щоб виконувати команди користувачів. Він може відповідати на запитання, виконувати завдання, надсилати повідомлення, відтворювати музику, розповідати жарти та багато іншого.

Голосові помічники дозволяють звести до мінімуму, а іноді і зовсім виключити необхідність використання рук і очей для перегляду контенту в Інтернеті. Вони дуже часто використовуються, наприклад вони можуть перевіряти мову користувача, аналізувати його реакцію та виконувати людські команди. 91% користувачів використовують голосового асистента для отримання відповідей на питання, 89,5% слухають музику, 85,2% перевіряють погоду і 71,4% встановлюють таймер.

Деякі переваги голосового асистента:

- Він зручний та швидкий. Можна спитати голосового асистента про будь-що, просто кажучи йому своє запитання. Голосовий асистент буде надавати вичерпну відповідь або допомагати знаходити її;
- Він інтелектуальний та креативний. Голосовий асистент може не тільки надавати вам факти та дані, але й робити це за допомогою генерації контенту;
- Він безпечний та конфіденційний. Голосовий асистент не зберігає інформацію про ваші запитання та дзвінки і виконує їх на шляху до сервера. Він також не передає ваші дані третьим сторонам без вашого дозволу.

1 ЗАГАЛЬНИЙ РОЗДІЛ

1.1 Аналіз технічного завдання

Голосові асистенти — це інноваційні програмні системи, що використовують технології розпізнавання та синтезу мови для взаємодії з користувачами через голосові команди. Вони значно полегшують наше повсякденне життя, дозволяючи виконувати різноманітні завдання, взаємодіяти з іншими пристроями та отримувати інформацію лише за допомогою голосу.

Основні призначення голосових асистентів охоплюють кілька важливих аспектів:

1. Автоматизація рутинних завдань. Голосові асистенти дозволяють швидко встановлювати будильники, нагадування, створювати списки справ або покупок. Наприклад, можна сказати: "Алекса, додай молоко до списку покупок", і це буде миттєво зроблено.

2. Швидкий доступ до інформації. Ці асистенти можуть надати відповіді на будь-які запитання, знайти інформацію в інтернеті, повідомити про погоду, новини, результати спортивних подій тощо. Наприклад: "Сірі, які новини сьогодні?" або "Гугл Асистент, хто виграв вчорашній матч?"

3. Керування розумним домом. Інтеграція з розумними пристроями дозволяє контролювати освітлення, термостати, системи безпеки та інші елементи розумного будинку через голосові команди. Наприклад: "Окей, Google, вимкни світло в кухні" або "Алекса, встанови термостат на 20 градусів".

4. Організація робочого процесу. Голосові асистенти можуть допомогти з плануванням зустрічей, створенням нагадувань, відправкою повідомлень або електронних листів, що особливо корисно в робочому середовищі. Наприклад: "Сірі, нагадай мені про зустріч о 15:00" або "Гугл Асистент, відправ повідомлення Марії".

5. Розваги та медіа. Голосові асистенти можуть відтворювати музику, запускати подкасти, аудіокниги або відео. Достатньо сказати: "Алекса, включи мою улюблену пісню" або "Гугл Асистент, запусти новий епізод 'Гри престолів'".

6. Навчання та розвиток. Вони можуть бути корисними для дітей і людей похилого віку, надаючи доступ до навчальних програм, інтерактивних ігор та допомагаючи у вивченні нових навичок. Наприклад: "Сірі, розкажи казку" або "Алекса, навчи мене новому слову на іспанській".

Голосові асистенти постійно вдосконалюються, адаптуючись до потреб користувачів і пропонуючи нові функції. Їх здатність до навчання і адаптації робить їх все більш незамінними помічниками у сучасному світі, спрощуючи наше життя та роблячи його більш зручним.

Історія розвитку голосових асистентів є захоплюючим прикладом технологічного прогресу, що охоплює десятиліття інновацій. Від перших простих систем розпізнавання голосу до сучасних асистентів, керованих штучним інтелектом, цей шлях був позначений значними досягненнями у комп'ютерній науці, обробці мовлення та машинному навчанні.

Сьогодні голосові асистенти, такі як **Siri**, **Alexa**, **Google Assistant** і **Microsoft Cortana**, здатні виконувати складні завдання, інтегруватися з різноманітними пристроями та додатками, а також адаптуватися до індивідуальних потреб користувачів.

Завдяки вдосконаленню технології обробки природної мови (**NLP**) та машинного навчання, голосові асистенти стали більш точними і здатними розуміти контекстні запити. Вони тепер активно використовуються для керування пристроями Інтернету речей (**IoT**), що робить наші будинки "розумнішими" і більш зручними.

У найближчому майбутньому ми можемо очікувати подальшого вдосконалення голосових асистентів, включаючи покращення точності розпізнавання мови, більш глибоку інтеграцію з іншими технологіями та

пристроями, а також розвиток нових функцій, які ще більше спростять наше життя. Еволюція голосових асистентів від простих систем розпізнавання мови до складних, штучно інтелектуальних помічників відображає значний прогрес у технологічній сфері, що впливає на наше повсякденне життя, роблячи його більш зручним та ефективним

1.2 Технічне завдання

1.2.1 Найменування та область застосування

Назва дипломної роботи розробка програмного забезпечення «Голосовий асистент». Дана програма буде використовуватися для полегшення рутинних, щоденних справ. Також вона може стати у нагоді в ситуаціях, коли зайняті руки або немає доступу до таких пристроїв введення інформації як клавіатура чи мишка. Але обов’язково необхідно мати пристрій захоплення звуку.

Об’єктом використання цього програмного продукту є звичайний користувач, який зможе переглядати команди, додавати та видаляти власні, і викликати їх за допомогою голосу. Програмне забезпечення буде надзвичайно легким у користуванні і для експлуатації програми, від користувача вимагається володіти базовими знаннями персонального комп’ютера.

Так як програмне забезпечення буде розроблятись тільки для комп’ютерної версії, вона буде використовуватись виключно в приміщенні, що виключає шкідливий навколишній шум.

Адже, шум це найголовніший шкідливий чинник для даної програми, тому щоб його позбутись, або хоча б звести до мінімуму, програмний продукт матиме високу якість розпізнавання мовлення, та швидку реакцію на команди, для того будуть використані різні технологічно ефективні архітектури розпізнавання звуку.

1.2.2 Призначення розробки

Експлуатаційне призначення даного програмного виробу полягає у полегшенні задач, які виникають при роботі з персональним комп'ютером. Задачі можуть бути як легкі (для звичайних користувачів) так і складні (для системних адміністраторів). Програма міститиме як системні команди так і команди користувача.

1.2.3 Вимоги до програмного забезпечення

Дане програмне забезпечення повинне бути написане на мові програмування **Python**. Мова програмування **Python** — це високорівнева інтерпретована об'єктно-орієнтована мова програмування із суворою динамічною типізацією.

Він був розроблений в 1990 році Гвідо ван Россумом. Структури даних високого рівня, динамічна семантика та динамічні посилання привабливі не лише як засіб об'єднання існуючих компонентів, але й для швидкої розробки додатків. **Python** підтримує модулі та пакети модулів для забезпечення модульності та повторного використання коду.

Інтерпретатор **Python** та стандартна бібліотека доступні як у складеному, так і в оригінальному вигляді на всіх основних платформах. Мова програмування **Python** підтримує кілька парадигм програмування, включаючи об'єктно-орієнтовану, процедурну, аспектно-орієнтовану та функціональну.

1.2.3.1 Вхідні дані

Основне джерело отримання даних користувача — це мікрофон. Так як це голосовий асистент, команди будуть розпізнаватися з мікрофона користувача. Але введення даних з клавіатури та миші теж присутнє у програмі, що дає змогу додавати та видаляти команди.

1.2.3.2 Вихідна інформація

У програмному виробі будуть присутні два види вихідної інформації. Основний вид вихідної інформації буде візуальним. Простіше кажучи, користувач зможе побачити інтерфейс програми. Він потрібен для коректного налаштування команд голосового асистента. Інший вид вихідної інформації буде у вигляді звуку. Асистент зможе повідомляти про стан виконання команд, попереджати про помилки введення даних, або просто спілкуватись із користувачем.

1.2.4 Часові характеристики

Майбутня програма буде створена у середовищі мови програмування **Python**, у неї буде досить швидкий час розпізнавання команди, який напряму залежить від величини запису команди сказаної користувачем. Якщо запис буде надто великим, то програмі потрібно буде достатньо багато часу, щоб розпізнати інформацію введену користувачем.

Для того, щоб уникнути такої ситуації в майбутньому, потрібно буде збільшити вимоги до апаратних засобів, або зменшити число записів. Також програмний продукт не повинен володіти обмеженими тимчасовими характеристиками. Програмне забезпечення буде оптимізоване так, щоб зменшити потужність і збільшити ефективність, тобто коли програма буде працювати у фоновому режимі, вона не буде записувати годинами, поки не розпізнає звернення.

Однак, як уже згадувалося, програма буде написана мовою програмування **Python** і запускатиметься в операційній системі **Windows**, а тривалість безперервної роботи програмного продукту буде напряму залежити від безперервної роботи самої операційної системи **Windows**.

1.2.5 Вимоги до надійності

Найголовнішою вимогою для надійної роботи програмного забезпечення є те, що всі файли, що використовуються в даному програмному продукті, повинні знаходитися в одному каталозі і їх редагування, тобто зміна імені та розширення, суворо заборонено.

У файлі **data_base** з розширенням **py**, у якій зберігатиметься інформація про системні команди та налаштування програвача звуку. Для цього у програмі повинен бути організований блок, який опитує про наявність цього файлу на жорсткому диску. У випадку, коли файл не буде знайдено, то програма повинна видати користувачу інформацію про помилку.

1.2.6 Умови експлуатації

Для того, щоб програмний продукт правильно і швидко функціонував, необхідно поставити такі вимоги до апаратного і програмного забезпечення:

- Операційна система: Windows 7, 8, 10;
- Процесор: Ryzen 5 5600G , Intel Core i3 10100f і вище;
- Оперативна пам’ять: 16 GB ОП і вище;
- Вільне місце на жорсткому диску: 2 GB доступного місця.

1.2.7 Вимоги до програмної документації

Разом із готовою програмою постачається така технічна документація:

- Інструкція запуску програми;
- Загальні відомості про можливості програми;
- Інструкція з вимогами до апаратних засобів;
- Інструкція по експлуатації ПЗ.

1.2.8 Стадії та етапи розробки

Розробка даного програмного забезпечення повинна пройти такі стадії та етапи:

- Формування вимог до поставленої задачі;
- Проектування та створення блок-схем, необхідних для подальшої
- роботи;
- Складання загального алгоритму роботи програми;
- Розробка і написання тексту програми в середовищі програмування
- Тестування програми на різному апаратному і програмному
- забезпеченню;
- Усунення помилок при роботі ПЗ;
- Створення інструкції з експлуатації;
- Створення технічної документації до даного програмного продукту.

1.2.9 Порядок контролю та прийому

Тестування програмного продукту буде проводитись на базі персонального комп’ютера такої конфігурації апаратного та програмного забезпечення:

- Операційна система: Windows 10;
- Процесор: Intel Core i3 12100f;
- Оперативна пам’ять: 16 GB ОП.

2 РОЗРОБКА ТЕХНІЧНОГО ПРОДУКТУ

2.1 Постановка задачі на розробку програмного забезпечення

Програма повинна виконувати функції голосового помічника. Вона повинна чітко розпізнавати голосові команди користувача, та достатньо швидко виконувати їх. Перед тим як асистент розпізнаватиме команду, користувачу необхідно звернутись до нього за його іменем. Після того як асистент розпізнає звернення, він запуститься на прослуховування команди.

У програмі будуть присутні системні команди, які наперед будуть записані в програму та їхня реалізація. Також користувач зможе додати або видалити власні команди, вказавши ключове слово та шлях до виконуваного файлу. Виконуваний файл повинен використовувати такі бібліотеки як: **Speech Recognition, time, pygame, webbrowser, os, sys, random, threading**.

2.2 Опис та обґрунтування вибору методу організації вхідних та вихідних даних

2.2.1 Вхідні дані

Як вже загадувалось раніше, основні вхідні дані користувача відображатимуться у вигляді звуку. Тобто для спілкування з асистентом потрібно використовувати пристрої введення саме звукової інформації.

Після запису голосу користувача, програма почне розпізнавати слова, і у разі відповідності перетворить звуковий тип інформації у текстовий. Далі вона буде виконувати перевірку, тобто шукати серед тексту команду, яка буде записана в системі.

Якщо команду не знайдено, то програма далі продовжить прослуховування, якщо знайдено — асистент виконає команду, повідомить

про це користувача і далі продовжить прослуховування (звісно, якщо це не була команда завершення програми).

У програмі також буде реалізована структура, яка дозволяє здійснювати запис даних у файл. Така реалізація повністю задовольняє вимоги до курсової роботи. Також за допомогою цієї структури здійснюється запис у файл, зчитування з файлу та швидкий пошук у ньому.

2.2.2 Вихідні дані

Вихідні дані у програмному продукті поділятимуться на три типи:

- Чат/консоль спілкування з асистентом. В програмному забезпеченні буде в наявності інтерфейс. Інтерфейс поділятиметься на дві вкладки: Головна і Меню команд. Чат спілкування з асистентом відображатиметься на головному екрані. В ньому асистент буде повідомляти про всілякі дії. У чаті відображатимуться відповіді на питання задані користувачем. Також у ньому буде виводитись список команд самого асистента та результат певних команд;
- Звукові сповіщення. Асистенту який вміє розуміти голос, не вистачатиме його ж голосу. Тому в програмному продукті асистент матиме змогу відтворювати аудіозаписи на яких наперед записані його відповіді і коли користувач звернеться до асистента то він відповість: «Слухаю». Також після виконання певної команди асистент сповістить користувача про успішне або не успішне виконання команди;
- Текстові файли. Програма матиме функціонал роботи з текстовими файлами. Ця можливість дозволить створювати, редагувати та видаляти файли. Одна з команд асистента це розпізнавати, що сказав користувач та записувати його у текстовий файл. Після чого користувач зможе переглянути у файлі, те що він сказав. Таким чином можна дуже спростити рутинні задачі по диктуванні тексту.

2.3 Опис методів реалізації функцій програми

Програмне забезпечення буде складатись з двох файлів, одинадцяти функцій та одинадцяти бібліотек, без яких програма не зможе працювати належним чином. Також важливо, що програмне забезпечення повинно працювати у двох потоках.

Потоки виконуватимуться паралельно, тобто незалежно один від одного. Наприклад, якщо користувач попрощається з асистентом, то інтерфейс продовжить свою роботу і навпаки. У першому (головному) потоці буде виконуватись імпортування потрібних бібліотек(перед запуском програми). Також головний потік буде спрямований на виконування функцій інтерфейсу. Тобто там буде створено об’єкти типу: Текст, Поле вводу та зображення, ще там будуть присутні кнопки та їхні функції.

А другий потік буде спрямований саме на прослуховування мікрофона. Тобто у другому потоці буде виконуватись системна програма, там вже будуть створюватись функції з командами асистента, функції відтворення сповіщень, функції фільтрації тексту та багато іншого.

2.3.1 Функція main

При запуску програми одразу запускається функція **main**, яка моментально створює блоки, кнопки, текст, після чого запускатиметься інтерфейс. Вікно інтерфейсу (див. рис.2.1) буде мати розширення: 450 пікселів в ширину та 700 пікселів у висоту. Спочатку створюватиметься фон програми, після чого буде перенесений у код. Інтерфейс матиме зображення фону, зображення асистента, під зображенням самого асистента буде розміщено інформаційний текст, який під час роботи програми буде змінюватись залежно від його стану. Тобто коли асистент відпочиває, прослуховує, або виконує команду в тому блоці буде відображатись інформація (див. рис. 2.2).

Далі під текстом будуть розміщуватись дві кнопки поблизу одне до одного. Перша кнопка називатиметься «Головна», друга — «Меню команд». Функціонал кнопки «Головна», полягає у відображенні головної сторінки програми. Тобто щоб, коли користувач перемкнеться на другу вкладку, мав змогу повернутись назад у головне меню.

Друга кнопка під назвою «Меню команд» дає змогу перемкнутись на меню, у якому відбувається конфігурація команд. Меню конфігурації команд представлятиме собою два поля вводу та три кнопки (див. рис.2.3). Над полями вводу відображатиметься текст, який пояснює користувачу, які саме дані потрібно ввести у поле. Під полями вводу будуть розташовані три кнопки.

Перша кнопка матиме назву «Додати команду». Дана кнопка матиме функцію запису власної команди користувача. Для цього йому потрібно буде ввести ключове слово, за яким потрібно звертатись до асистента та шлях до виконуваного файлу. Якщо ввести невірну інформацію, то програма про це сповістить (див. рис.2.4). Друга кнопка «Видалити команду», виконуватиме такий ж самий тип дії як і попередня, але вона не додаватиме, а видалятиме команду зі списку.

Остання кнопка «Очистити всі команди», вона буде виконувати функціонал очищення всіх команд зі списку. Дана кнопка полегшить конфігурування команд, щоб не видаляти багато команд по черзі, варто скористатись цією. Перед очищенням всіх команд програма відкриє діалогове вікно, у якому користувач може дати згоду на виконання команди або ж навпаки відмовитись (див. рис.2.5).

Останній об’єкт інтерфейсу це велике текстове поле вводу. У ньому відображатиметься інформація яку надаватиме асистент. Тобто це може бути, результат виконання команди, відповідь користувачеві на питання або ж список команд асистента (див. рис.2.6).

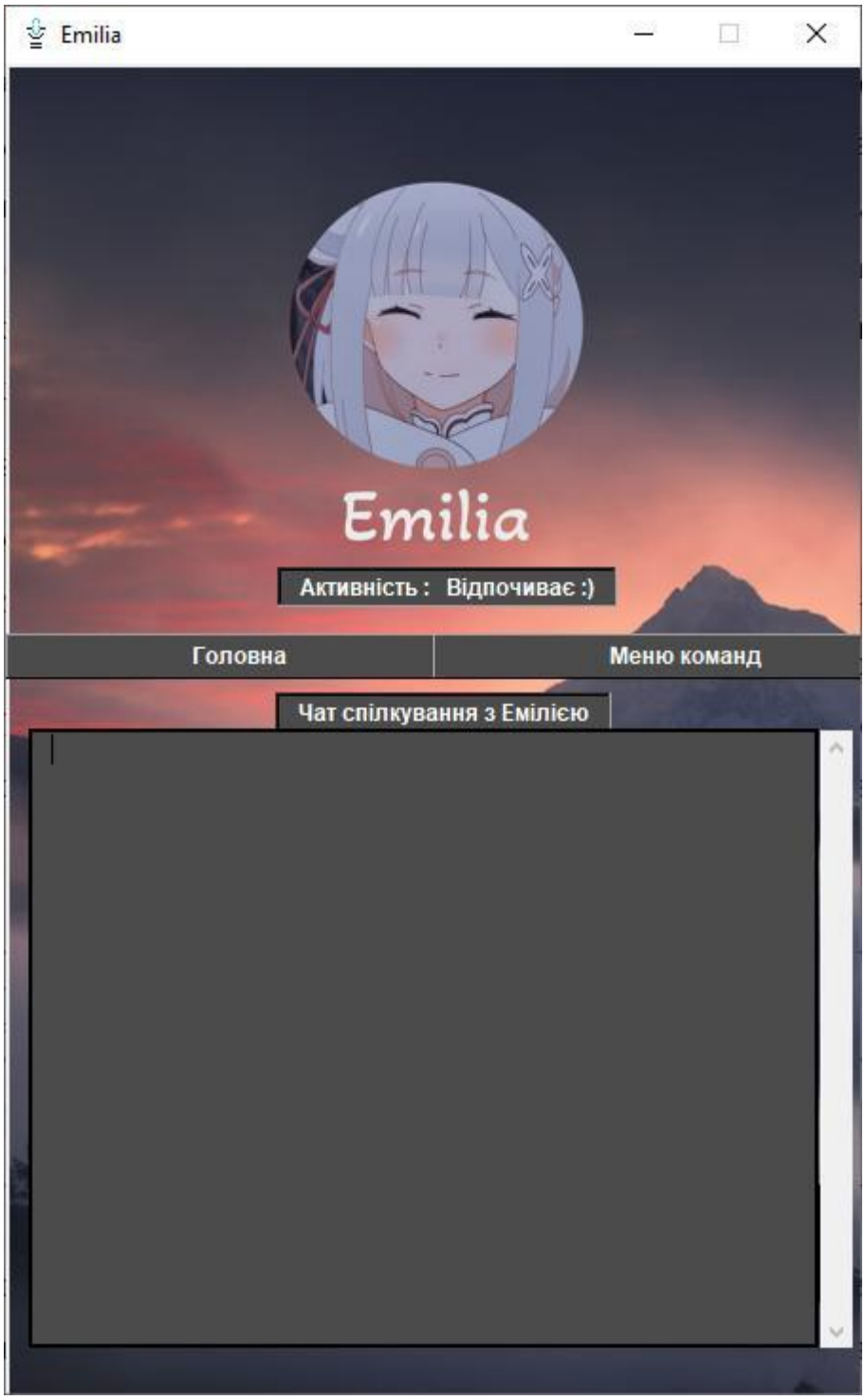

Рисунок 2.1 – «Інтерфейс програми на головній сторінці»

Рисунок 2.2 – «Інформаційний текст про стан роботи асистента»

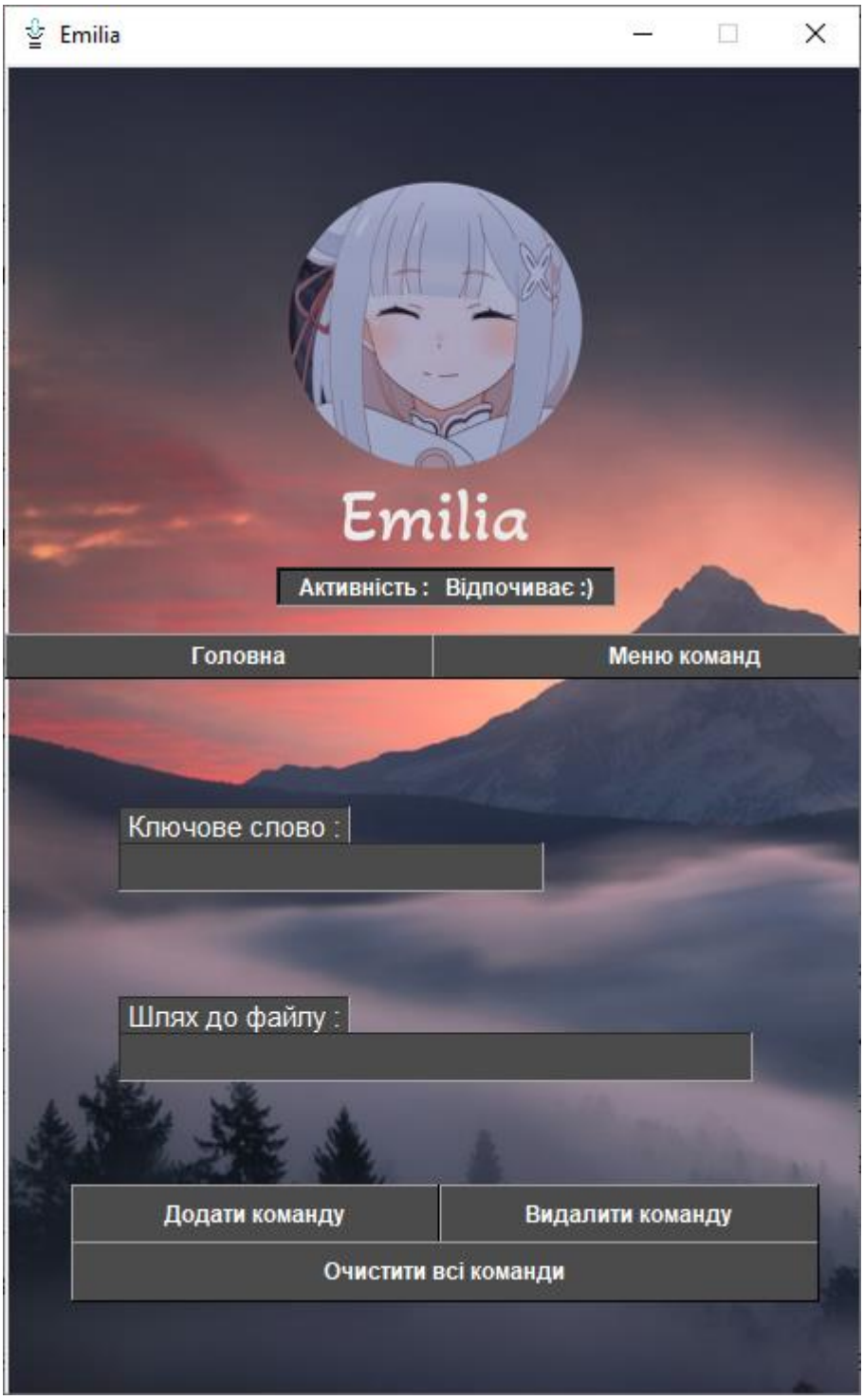

Рисунок 2.3 – «Інтерфейс програми на вкладці **Меню команд**»

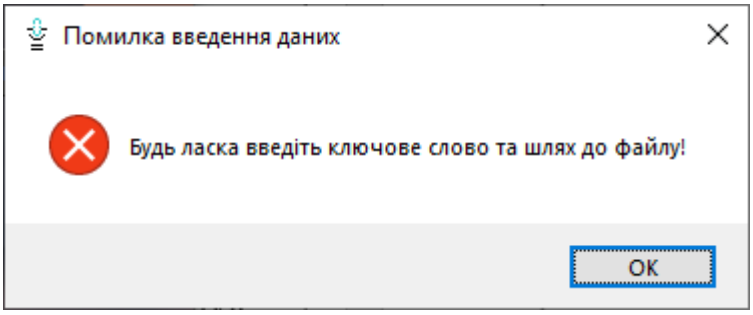

Рисунок 2.4 – «Вікно помилки при введенні невірних даних»

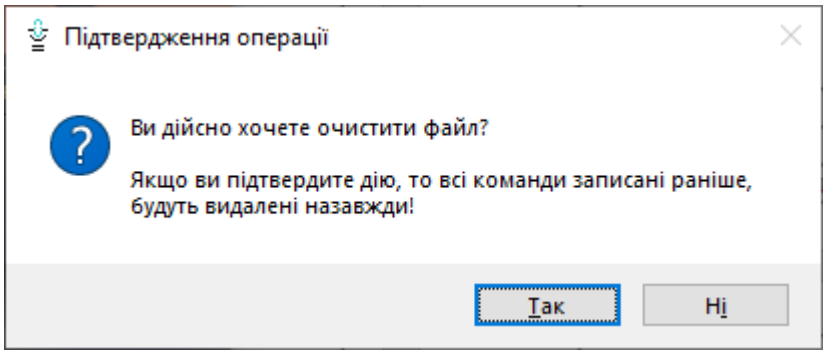

Рисунок 2.5 – «Діалогове вікно при очищенні всіх команд»

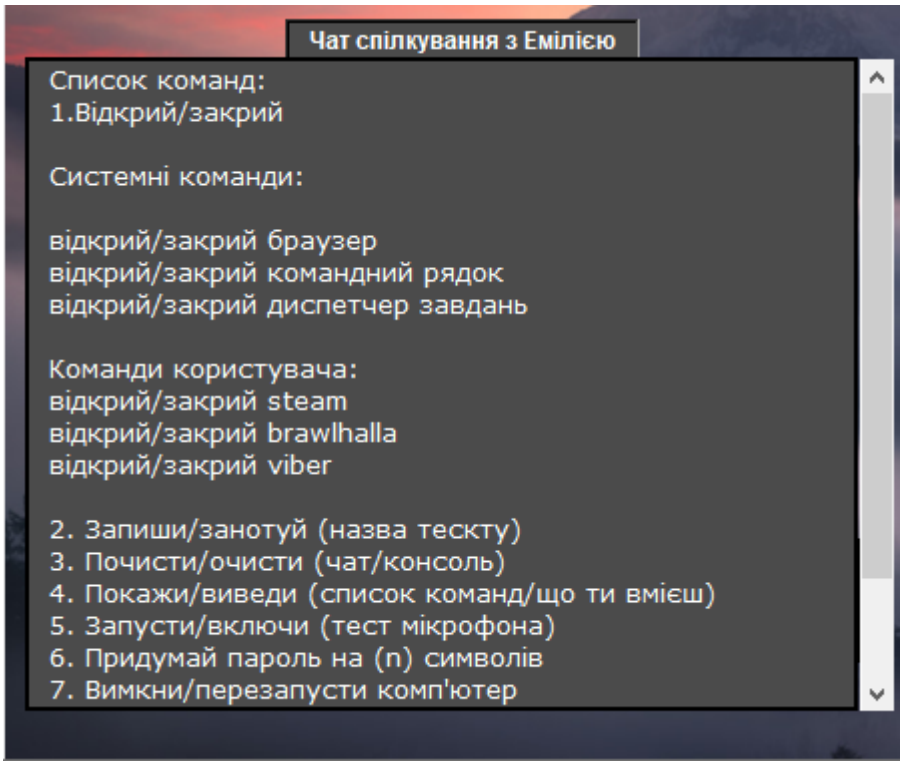

Рисунок 2.6 – «Чат спілкування з асистентом»

2.3.2 Функція main_program

Головна функція, в якій будуть виконуватись основні операції програми. На початку функції оголошуються змінні та об’єкт під назвою «**Recognizer**», це буде екземпляр класу, який дозволить виконувати різні дії з мікрофоном. Далі запускатиметься вічний цикл **while True:** для того , щоб прослуховувати мікрофон користувача на постійній основі, поки користувач не закриє програму.

Після цього буде конструкція **try:** для перевірки умови на виявлення помилок. В ній перевірятимуться дві умови: перша — якщо програма не змогла розпізнати текст сказаний користувачем **except UnknownValueError:**, тоді у консоль виводиться **print('шум ...')**. Після чого вона буде пропускати ітерацію командою **continue**. Друга, це якщо виникли технічні неполадки у самого сервісу **Google Speech Recognition**, помилка виглядатиме так: **except RequestError as e:**. При виникненні цієї помилки в консолі буде виведено про це повідомлення **print("Could not request results from Google Speech Recognition service; {0}".format(e))**.

Якщо ж помилок не виявлено то програма буде починати прослуховувати мікрофон. Кожна ітерація програмного забезпечення буде відправляти сказаний текст у функцію **respond**. Це буде відбуватись до тих пір поки програму не буде закрито або коли програма розпізнає звертальне слово. Буде починатись перевірка яка включає в себе розпізнання слова звернення, серед слів сказаних користувачем.

Коли асистент розпізнає слово звернення (його ім’я), то буде пропускатись ітерація і наступний текст сказаний користувачем буде розпізнаватись як команда.

Якщо у словах користувача буде розпізнано ключове слово команди, то асистент виконає її і сповістить про це користувача, якщо ж ні, то асистент повідомить користувача, що дана команда відсутня і цикл далі продовжить очікувати на слово звернення.

2.3.3 Функція command

Функція **command**, одна з ключових функцій програми. Саме ця функція отримує список з інформацією, який тип дії та яке ключове слово списав користувач. Вона прийматиме два параметри: **str text** і **bool resp**. У параметр **text** буде передаватись текст який розпізнав асистент, а у параметр **resp** буде передаватись булеве значення **True**. Далі будуть оголошені глобальні змінні, такі як **loop_listen**, **test_or_no** і **name_fale**, за допомогою команди **global**.

Після цього буде перевірятися умова на ймовірність виникнення похибки. Вона повинна бути присутня, та як вона перевіряє чи активне вікно програми. Якщо цього не зробити то програма буде закриватись сама по собі. Далі буде написана перша умова на перевірку команди. Ця умова буде перевіряти, якщо користувач попрощався з асистентом то програма завершиться.

Але якщо вікно з інтерфейсом залишиться активним, то в ньому у чаті з асистентом буде написано «Щоб покликати Емілію, перезапустіть будь ласка програму!», командою **field.insert(text)**. Така структурна схема дозволить користуватись інтерфейсом, навіть якщо голосовий асистент було відключено. Якщо жодна з умов не виконалась, то текст користувача відправиться фільтруватись у функцію **filter_cmd**.

Отриманий відфільтрований текст, буде повертатися у вигляді списку з двома елементами, перший елемент це тип дії який повинен виконати асистент, а другий - це ключове слово, якому відповідає команда. У змінній **resp**, яка була оголошена раніше необхідно буде змінити **True** на **False**, щоб після виконання команди асистентом, програма не очікувала на ще одну команду, а очікувала на слово звернення.

Програмний продукт матиме два типи команд, це звичайні команди та термінові. Як раніше було сказано, голосова інформація записана мікрофоном відправлятиметься у фільтр, де серед великого об'єму тексту він знаходить ключове слово та тип дії. Дана структурна схема займає трохи часу, тому

існуватиме поділ команд, де термінові команди не будуть фільтруватись. Тобто так як, у цьому не має потреби, термінові команди будуть виконуватись набагато швидше. Така реалізація сприяє підвищенню оптимізації програми, що є дуже ефективним рішенням.

Далі у програмі будуть розташовані чотири основні умови для розпізнавання команди користувача, три з яких перевіряють термінові команди першими:

Перша умова. Вона буде виконуватись якщо користувач викликав команду 'скасуй сеанс'. У цьому випадку програма скасує раніше запланований сеанс на комп'ютерні. Буде виконана команда os.

Друга умова. Так як функція приймає список з двома елементами, необхідно перевірити розмір списку. Якщо розмір не відповідає нормі, то ітерація пропускається.

Третя умова. Дана умова виконує перевірку на виявлення команди яка керує сеансами комп'ютера. Наприклад: "Вимкни комп'ютер", "перезапустити комп'ютер" або "вийди з користувача".

Четверта умова. Якщо жодна з вищих умов не виконалась, то буде виконана остання, яка вже визначає яку конкретно команду викликав користувач та виконає її.

Після виконання будь-якої команди голосовий асистент про це повідомить своїм голосом. Також після дії відповідних команд в чат з асистентом буде виведено відповідну інформацію. В кінці цієї функції буде розташовуватись те що вона повертає, а саме повертати вона буде змінну **resp**, у якої раніше було змінено значення.

Отже, голосовий асистент виконавши команду розпізнану команду, знову продовжить прослуховування мікрофона та буде очікувати ключове слово.

2.3.4 Функція filter_cmd

Функція **filter_cmd** теж буде одною з ключових функції програмного продукту. Так як вона виконуватиме роль фільтрації тексту. Саме ця функція наддасть змогу вияснити ключове слово та тип дії команди, яку оголосив користувач. Вона викликатиметься у функції **command**, приймаючи один параметр **str text**. Функція починатись буде, з викликання функції **read_file()**, яка поверне список команд зчитаних з текстового файлу і запишеться у змінну **contain_file**.

Далі потрібно буде створити три зміні **AS_KEY_WORDS, AS_KEY_PATH** і **lst_sort**, яким буде призначено три пусті списки. Це потрібно для того щоб, зчитані дані з текстового файлу переформатувати такий тип даний який потрібен фільтру. Для цього, буде існувати три цикли **for**. Перший з них розділить кожен текстовий рядок, на два підрядки і збереже їх у списку **lst_sort**.

В другому циклі списки **AS_KEY_WORDS** і **AS_KEY_PATH** будуть призначати собі кожен по рівній кількості підрядків. Щоб не займати оперативну пам’ять намарно, список **lst_sort** потрібно буде очистити. Головний цикл даної функції це **for lst in data_base.AS_TYPE_DO:**, він буде проходитись по кортежі зі списками, у яких записані типи дії команди та буде виявляти їх. У ньому існуватиме наступний цикл, який вже проходитиме по тому списку і шукати відповідний тип дії. Програмне забезпечення дозволятиме вимовляти команди не точно.

Наприклад у програмі буде присутня команда **покажи що ти вмієш**, навіть якщо користувач скаже **чуєш, цей покажи мені що ти вмієш якби тойво**, фільтр чітко розпізнає які з цих слів є важливими і лишні видалить. Завдяки тому методу фільтр буде проходитиме по всім спискам з командами та типами дії і шукати відповідне їм. В кінці функція повертатиме список **lst_sort**, який у свою чергу матиме два елемента, тип дії та ключове слово команди.

2.3.5 Функція tts

Функція **tts** буде приймати один аргумент **str text**, який є рядком. Вона буде генерувати випадковий звуковий файл з відповідної категорії сповіщень, який зберігатиметься у папці **sounds**. На початку функції буде створено дві змінні, перша **rand_int**, друга **time_int**. Першій буде присвоєно число **1**, другій число **2**. Змінній **lst_sound**, буде присвоєно словник, у якому зберігатимуться дані для відповідної категорії звуків. Тобто у даному словнику для кожної категорії звуків, буде присвоєно кількість різних відповідей та час їх програвання.

Перший елемент **lst_sound** це кількість варіацій відповідей асистента і щоб вони програвались у випадковому порядку, буде використано функцію **randint(1 , lst_sound[0])**, яка рандомно вибиратиме число від **1** до **lst_sound[0]**. Змінній **time_int** буде присвоєно **lst_sound[1]**, у якому зберігатиметься час програвання звуку. Далі буде розташована сама команда відтворювання аудіо-файлів — **mixer.Sound(f"{sys.path[-1]}/{text} {rand_int}.mp3").play()**. **sys.path[-1] –** вказує на каталог з аудіо-файлами у системі, **text –** категорія звуку, **rand_int** – випадковий аудіо-файл з певної категорії, а функція починатиме відтворювати звук. Після старту відтворення звукового файлу, система засинає на **time_int** секунд, за допомогою команди **sleep(time_int)**.

2.3.6 Функція read_file

Дана функція не прийматиме жодного аргументу. Їй це і не потрібно буде, так як вона буде існувати лише для отримання з текстового файлу з командами користувача дані.

Все починатиметься з перевірки на помилку, вона потрібна щоб вияснити чи присутній файл у каталозі системи. Тобто далі йде функція, яка

відкриває файл у режимі **читання**, якщо виникла помилка **FileNotFoundError**, це означає що файлу не знайдено.

Тоді файл буде відкрито у режимі **створення/перезапису**, де файл буде створено у каталозі системи та записано туди рядок **command#path** командою **file.write(text)**. Це потрібно для того щоб система розуміла чи точно цей файл було відкрито. У будь-якому випадку файл буде закрито командою **file.close()**. Функція в кінці повертатиме список команд користувача, командою **return contain_file**.

2.3.6 Функція respond

Функція **respond**, буде отримувати дві змінні **str text** і **bool resp**. У змінну **str text**, буде передано текст, який сказав користувач, а у змінну **bool resp**, буде передано значення **False**. Дана функція перевірятиме текст на наявність слова звернення до асистента.

Буде створена умова **if text in data_base.AS_ALIAS:**, тобто якщо текст є у списку слів звернення до асистента, то виконається команда **tts(“listening”)**, яка відтворить відповідь голосового асистента. А значення **resp**, буде змінено на **True**. Якщо ж умова не виконалась, то змінній **resp**, буде призначено **False**. Останній рядок функції буде повертати змінну **resp** зі зміненим булевим значенням.

2.3.7 Функція add_cmd

Дана функція, та наступні будуть спрямовані на конфігурації інтерфейсу. Завдяки функції **add_cmd**, користувач зможе додати у загальний список команд, який зберігається у текстовому файлі під назвою **open_close_cmd**, власну команду, вказавши шлях до виконуваного файлу та ключове слово по якому асистент повинен зрозуміти, що від нього хоче користувач.

Для того, щоб додати команду у список інших команд, користувач повинен ввести ключове слово та шлях до виконуваного файлу. Якщо користувач введе дані не в два поля а одне, або взагалі не введе інформацію, то програма припиниться. Щоб такої ситуації не виникало перед тим як додати команду, буде виконуватись умова.

Якщо хоча б один з полів не заповнено, то виконається дана команда **showerror(title='Помилка введення даних',message='Будь ласка введіть ключове слово або шлях до файлу!')**, завдяки якій, на екрані вискочить вікно системної помилки, у якому повідомляється причина її виникнення. Якщо ж дані введено вірно, то перед тим як додати команду, потрібно перевірити чи є він у наявності за допомогою команди **try:**. Якщо файл знайдено то він відкриється у режимі **дозапису**, якщо не знайдено то у режимі **створення/перезапису**.

Незалежно від того, у якому режиму було відкрито файл блок **finally:** запише команду у файл командою **file.write(f"{in_word}#{in_path}\n")**, закриє файл командою **file.close()** і очистить поля вводу командами **key_path.delete(0,END)** та **key_word.delete(0,END)**.

Також буде викликано інформаційне вікно у системі, яке сповістить користувача, що додавання команди пройшло успішно, команда **showinfo(title='Інформація про файл',message='Команду успішно додано у файл!')**.

2.3.8 Функція del_cmd

Функція **del_cmd,** теж буде виконувати операції з текстовим файлом з командами, але її функціонал буде повністю протилежний дія минулої функція, так як вона буде не додавати нову команду до списку існуючих, а навпаки видаляти команди зі списку.

Знову ж таки, потрібно перевірити чи ввів дані у поля користувач, якщо дані не було введено то буде викликано вікно системної помилки, за

допомогою команди **showerror (title = 'Помилка введення даних', message = 'Будь ласка введіть ключове слово або шлях до файлу!')**.

На відміну від функції додавання команди **add_cmd**, функція видалення, не потребує повної інформації про файл. Тобто команду, яку раніше було додано у список, можна буде видалити вказавши не два параметри, а лише один з них, назва команди, або шлях до файлу.

Далі буде виконуватись блок **try:**, для перевірки на наявність текстового файлу командами у системі. Якщо файл знайдено, то він відкриється у режимі **читання**, якщо ні — у режимі **створення/перезапису**.

Після чого, відбуватиметься фільтрація команд користувача, які було зчитано з текстового файлу. Далі шляхом перебору кожного елементу в циклі, буде виконано пошук потрібної команди у списку. Пошук буде виконуватись по двох критеріях: шлях до виконуваного файлу та ключе слово (обидва аргументи повинні бути коректними, інакше буде викликано вікно системної помилки).

Якщо співпадіння знайдено, то команду буде видалено, а асистент сповістить про це користувача командою **showinfo(title='Інформація про файл',message='Команду успішно видалено з файлу!')**, якщо ж ні — то програма виведе на екран вікно системної помилки **showerror (title='Інформація про файл',message='Список з командами пустий!')**.

Незалежно від того, у якому режимі відкриватиметься файл та чи була виявлена системна помилка у роботі, програма у блоці **finally:**, завершить роботу з текстовим файлом. Також буде очищено, поля вводу, щоб користувач не робив це вручну.

2.3.9 Функція cmd_clear

Призначення даної функції полягає у тому, що вона повністю очищує текстовий файл з командами користувача. Так як важливо врахувати різні події, перед тим як очистити усі файли, буде викликано діалогове вікно

командою **askyesno(title='Підтвердження операції', message='Ви дійсно хочете очистити файл?', detail='Якщо ви підтвердите дію, то всі команди записані раніше, будуть видалені назавжди!')**.

Тип дії, яку вибрав користувач (так\ні) буде записано у змінну **yes**, у вигляді булевого значення **True** або **False**. Якщо користувач вибрав **Так**, то текстовий файл відкриється у режимі **перезапису**, усі команди які були у файлі буде безслідно видалено (видалені команди неможливо буде відновити, для того і створено діалогове вікно, щоб попередити користувача про це). Після чого туди запишеться команда, яка повинна бути присутня на початку файлу, тоді текстовий файл буде закрито, якщо ж ні, то операція буде відхилена.

2.3.10 Функція hide_widget

Дана функція викликається при натисканні кнопки **Меню команд**. Вона приховає усі віджети з головного меню та проявить віджети з меню команд, що створить ефект переходу користувача з одного вікна в інше. Якщо користувач завчасно знаходився у вкладці **Меню команд**, то нічого не відбудеться.

2.3.11 Функція come_widget

Дана функція викликається при натисканні кнопки **Головна**. Вона приховає усі віджети з **Меню команд** та проявить віджети з головного меню, що створить ефект переходу користувача з одного вікна в інше. У випадку, якщо користувач завчасно знаходився у вкладці **Головна**, то нічого не відбудеться.

2.4 Визначення інформаційних зв’язків програмних компонентів

Інформаційні зв’язки програмного продукту, будуть побудовані по функціональній специфікації за рівнями зверху – вниз. На верхньому рівні виділені групи функцій, на наступних рівнях вони деталізуються у нові компоненти.

Отже головною функцією програми є функція **main_program()**, з якої викликаються функції **command** та **respond**. На рисунку 2.7 приведена ієрархічна структура інформаційних зв’язків програмних компонентів.

При завантаженні програмне забезпечення, починає процес прослуховування мікрофона, тоді виконується функція **command**. Після того, як голосовий ввід було записано, він перетворюється у звичайний текст, після цього, здійснюється зв’язок з іншими функціями у програмі, які взаємодіють зі самою структурою програмного продукту, а та сама структура в свою чергу, взаємодіє із файлом довільного доступу, під назвою **open_close_cmd.txt**.

Даний текстовий файл, містить у собі інформацію про особисті команди користувача, які можна конфігурувати у вікні асистента, додавати, видаляти та змінювати. На початку текстового файлу, у першому рядку писатиме **command#path**. Що візуально відображає, яким способом буде записана команда користувача. Тобто спочатку йде ключове слово, по якому потрібно звернутись до асистента, знак # і після нього вже шлях до виконуваного файлу.

Після зчитування та аналізу команд з файлу open_close_cmd.txt, програмне забезпечення порівнює введену користувачем команду з записаними у файлі командами. Якщо знайдено співпадіння, асистент виконує відповідну дію, запуск файлу або програми за вказаним шляхом. У випадку, якщо команда не знайдена, асистент повертає відповідне повідомлення про помилку або просить користувача повторити запит.

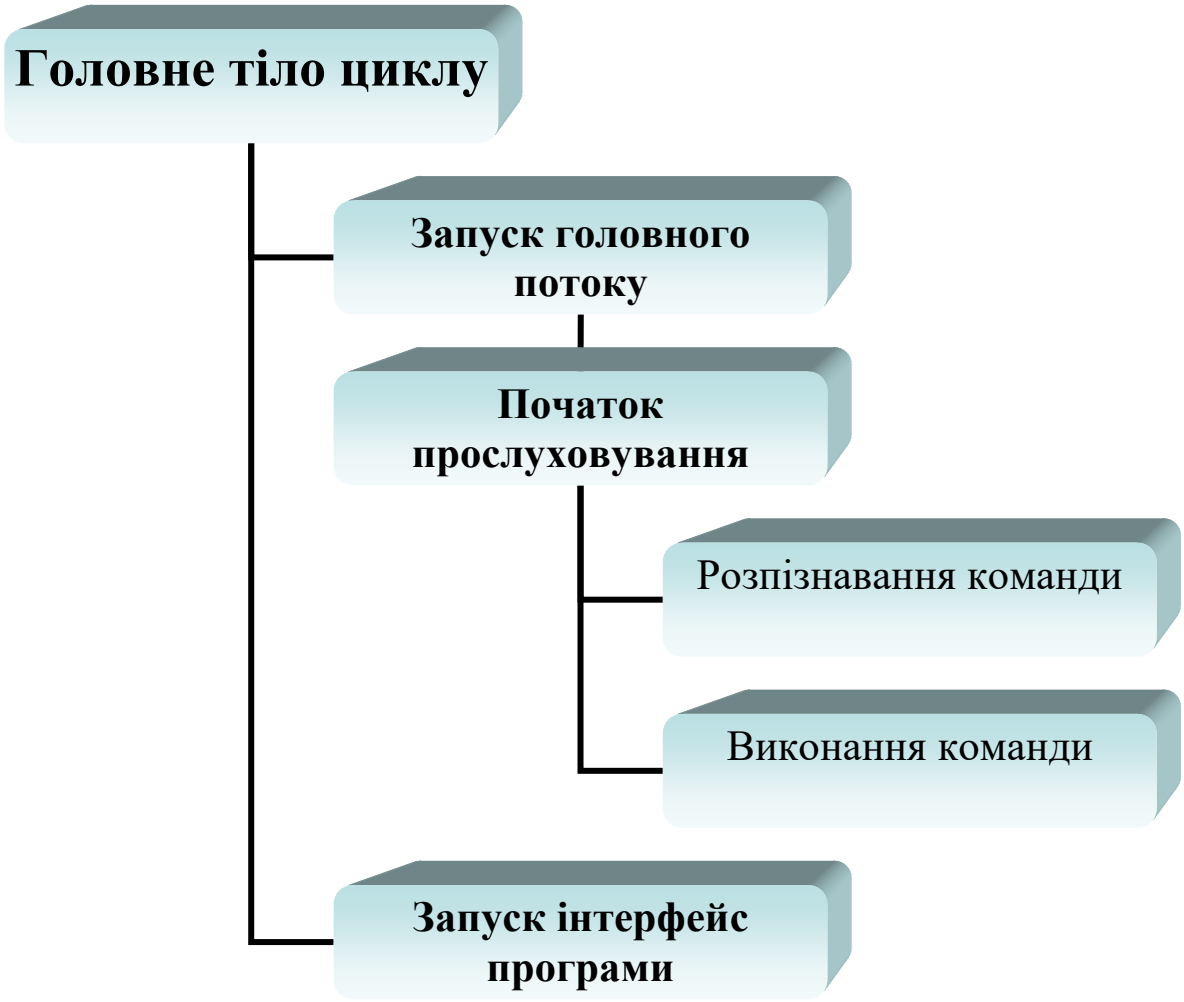

Рисунок 2.7 – «Ієрархічна структура інформаційних зв’язків програмних компонентів»

2.5 Написання текстів програми

Після того, як усі алгоритми функцій програмного продукту і загальні алгоритми роботи були складені, можна приступати до написання самого тексту програми.

Так як програмне забезпечення повинно бути написано на мові програмування **Python**, то середовищем програмування обиратиметься **Visual Studio Code**, яке дає змогу програмісту створювати та використовувати потрібні для роботи компоненти.

Мова **Python** - це потужна мова програмування, яка проста у вивченні. Вона має ефективні структури даних високого рівня та простий, але ефективний підхід до об’єктно-орієнтованого програмування. **Python** має елегантний синтаксис і динамічну типізацію, що разом з його інтерпретованим характером роблять його ідеальною мовою для створення сценаріїв і швидкої розробки додатків у багатьох сферах на більшості платформ .

Python містить у собі обширну стандартну бібліотеку, яка містить багато корисних модулів, таких як **os, sys, math, datetime, random, re, json, csv, sqlite3, unittest, xml, http, socket, email, smtplib, tkinter, numpy, pandas, matplotlib, scikit-learn, tensorflow**, та багато інших.

При розробці програми спочатку було реалізовано основну схему зчитування аудіо даних з мікрофона, а потім послідовно описано основні функції відповідно до заздалегідь складених алгоритмів.

Вони були описані послідовно відповідно до алгоритму. Після чого, програмний продукт був перевірена на наявність помилок, та був виправлений, якщо такі були знайдені.

2.6 Тестування та налагодження програми

Тестування програмного забезпечення — це процес перевірки відповідності заявлених вимог до продукту та реально реалізованої функціональності, який здійснюється шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, обраних певним чином.

Тестування може оцінювати відповідність вимогам, правильність відповіді для всіх можливих вхідних даних, виконання функцій за прийнятний час, практичність, сумісність із програмним забезпеченням та операційними системами, відповідність задачам замовника. Тестування можна проводити, як тільки створено виконуваний код (навіть частково завершений).

Процес розробки зазвичай передбачає, коли та як буде відбуватися тестування. Готове програмне забезпечення, було протестовано на базі персонального комп'ютера, з такою конфігурацією апаратного та програмного забезпечення:

- Операційна система: Windows 10;
- Процесор: Intel Core i3 12100f;
- Оперативна пам'ять: 16 GB ОП.

При завантаженні програми запускається: перевірка на наявність звукозаписуючого пристрою, перевірка на справність пристрою, перевірка з'єднання з мережею (вона потрібна для розпізнавання мовлення, так як модель мови зберігається не у системних файлах, а у хмарному сховищі), початок прослуховування мікрофона та вікно з інтерфейсом голосового асистента (див. рис.2.8).

Після того, як голосовий ввід було записано, він перетворюється у звичайний текст, після цього, здійснюється зв'язок з іншими функціями у програмі, які взаємодіють зі самою структурою програмного продукту, а та сама структура в свою чергу, взаємодіє із файлом довільного доступу, під назвою **open_close_cmd.txt**.

Відповідно до цього вікна, користувач може зробити такі дії:

- Переглянути активність голосового асистента;
- Перемкнутись між вкладками;
- Додати власну команду у список команд (перейшовши на вкладку **Меню команд**);
- Видалити команду зі списку (перейшовши на вкладку **Меню команд**);
- Очистити усі команди зі списку (перейшовши на вкладку **Меню команд**);
- Отримувати результати команд у чаті.

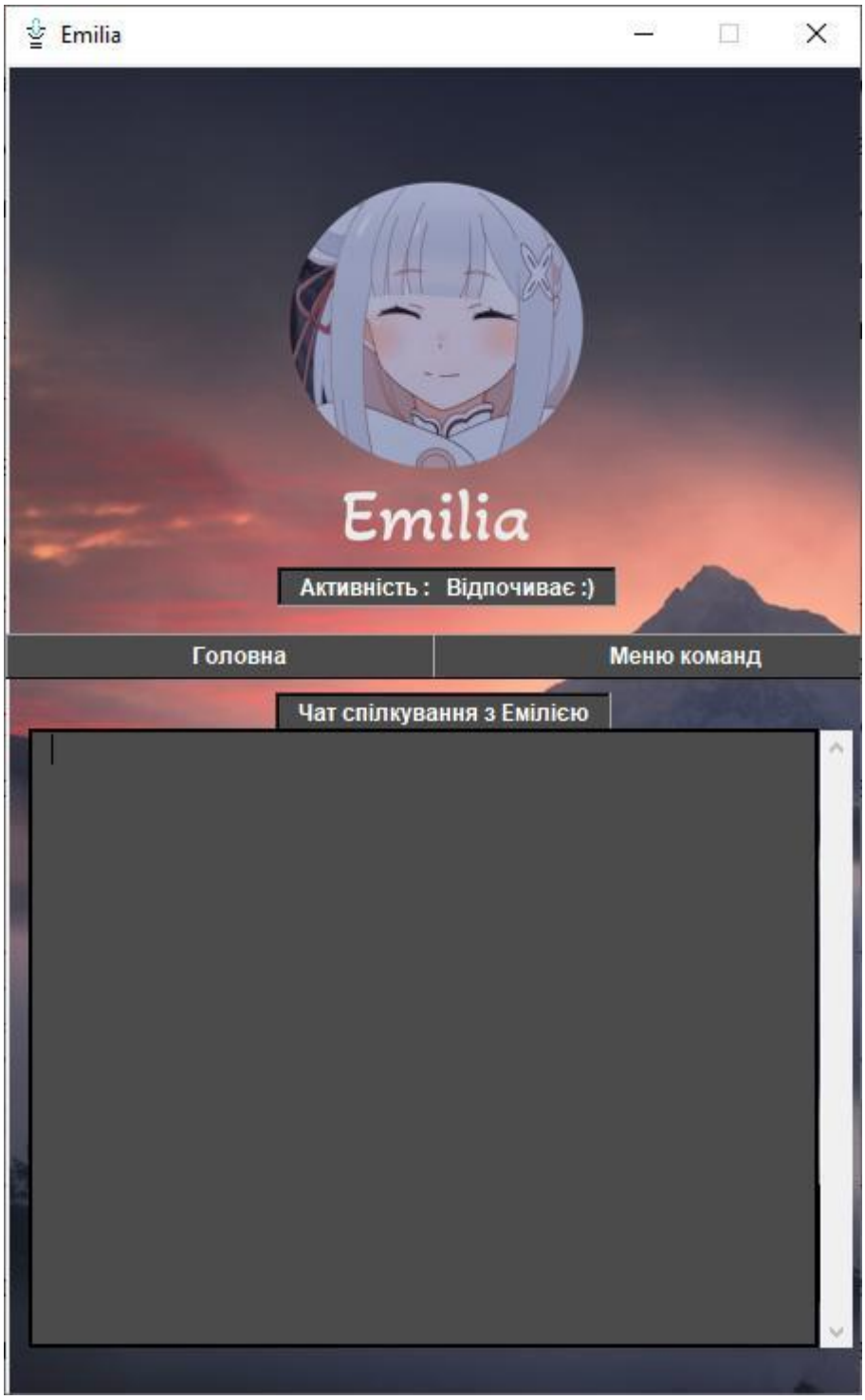

Рисунок 2.8 – «Інтерфейс програми на головній сторінці»

Якщо ж розглядати основний функціонал голосового асистента, то користувач після запуску, може одразу почати з ним спілкуватись. Для цього користувач повинен звернутись до асистента по імені **Емілія** або **Емма**, саме

так вирішив назвати її розробник. Після того як вона відповіла «**Слухаю вас»**. Користувач може дати одну з цих команд асистентові:

1. Відкрий/закрий (браузер/командний рядок/диспетчер завдань/ «команда користувача»);
2. Запиши/занотуй (назва текстового документу);
3. Почисти/очисти (чат/консоль);
4. Покажи/виведи (список команд/що ти вмієш);
5. Запусти/включи (тест мікрофона);
6. Придумай пароль на (n) символів;
7. Вимкни/перезавантаж комп’ютер;
8. Вийди з користувача;
9. Скасуй/сеанс (скасовує заплановане відключення ПК);
10. Бувай/папа (вимкнення асистента);

Після виконання всіх необхідних функцій, користувач для виходу з програми повинен відключити асистента, попрощавшись з нею командою під номером 10 та закрити вікно асистента. Так як асистент та її інтерфейс працюють у різних потоках, користувач може закрити вікно, і далі спілкуватись з асистентом у фоновому режимі або навпаки, відключити асистента та продовжити налаштування власних команд.

На основі проведеного тестування, можна зробити висновок, що програма «Голосовий асистент» є робочою, оскільки тестувалась в середовищі, що є еквівалентним реальному середовищу, в якому повинна функціонувати програма.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

3.1 Архітектура та дизайн системи

Для розробки програмного забезпечення завжди потрібно прикладати максимальні зусилля, щоб розроблений продукт, був надійний та універсальний. Щоб розробити програмний продукт, потрібно вибрати мову програмування, це ключовий момент усієї роботи, так як мов програмування велика кількість.

Кожна мова програмування відрізняється чимось, кожна розроблена під певні задачі, які вона повинна виконувати. Також є певні напрямки, на які поділені різні мови програмування. Існує два основних напрямки **frontend** та **backend**.

Frontend розробка:

- Охоплює частину веб-додатку саме ту, з якою користувач може напряму взаємодіяти та бачити. Дана частина містить інтерфейс користувача, відображення функціональних елементів, обробку запитів та інше;
- **UX** (користувацький досвід), дане поняття включає навігацію веб-додатку, відповідає за функціонал веб-додатку та все решта що відноситься до верстки: розташування елементів, текст та зображення. Дизайнер цього напрямку, несе велику відповідальність за привабливість та зацікавленість веб-застосунку.
- **UI** (інтерфейс користувача), може здаватись шо цей напрям ідентичний до **UX**, але це не так. **UX —** вивчає саму зацікавленість користувача, визначає характер користувача, та старається догодити кожному. У свою ж чергу **UI** відповідає за самий вигляд сайту, стиль сайту, тобто перше шо бачить користувач відвідавши веб-застосунок. Все з чим взаємодіє даний напрям це: вікна веб-додатку, блоки, форми, кнопки, іконки, зображення, текст, відеофайли та багато інших елементів інтерфейсу.

- Мови програмування для роботи з **frontend**: **HTML**, **CSS**, **JavaScript** та інші.

Отже, **frontend** — це лице додатку, те що користувач бачить та з чим взаємодіє, але що ж відбувається всередині «під капотом».

Backend розробка:

- Розробка та підтримка серверної логіки. Розробники цього напряму несуть велику відповідальність за реалізацію функціональності, яка відбувається на сервері. Це може включати обробку запитів, авторизацію, оброку даних та інші операції бізнес-логіки.
- Робота з базами даних: **backend**-розробники взаємодіють з реляційними (мова SQL) або нереляційними базами даних для зберігання та обробки інформації.
- Створення та підтримка API. Також дані розробники впроваджують **API** (Application Programming Interfaces), які визначають правила взаємодії між клієнтською та серверною частинами програми. Це дозволяє іншим компонентам взаємодіяти з сервером.
- Тестування та налагодження. бекенд-розробники перевіряють, чи серверна частина працює належним чином, виявляють та виправляють помилки.
- Оптимізація продуктивності. Оптимізація також важлива задача, яка відповідає за продуктивність системи для забезпечення швидкого виконання запитів.
- Мови програмування для роботи з **backend**: **Java, Python, Node.js, PHP** та інші.

Отже, для розробки програмного забезпечення потрібно вибрати одну з мов програмування яка відноситься до **backend**. Для розробки програми «голосовий асистент», була вибрана мова програмування **Python**. Мова програмування **Python** — це інтерпретована об'єктно-орієнтована мова програмування високого рівня зі строгою динамічною типізацією.

Високорівневі структури даних, а також динамічна семантика та динамічне зв’язування роблять його неймовірним для швидкої розробки додатків і як засіб об’єднання існуючих компонентів.

Python підтримує модулі та пакети модулів, що сприяє модульності та повторному використанню коду. Інтерпретатор **Python** і стандартна бібліотека доступні у скомпільованому та вихідному коді на всіх основних платформах. Мова програмування **Pytho**n — також підтримує кілька парадигм програмування, включаючи: процедурне, об’єктно-орієнтоване, аспектно-орієнтоване та функціональне. До основних переваг мови програмування **Python**, можна віднести наступне:

- Стандартний дистрибутив має велику кількість корисних модулів (включаючи модулі для розробки графічного інтерфейсу);
- Зручний для вирішення математичних задач (має інструменти для роботи з комплексними числами, може працювати з цілими числами будь-якого розміру, може використовуватися як потужний калькулятор у діалоговому режимі);
- Можливість використовувати **Python** в діалоговому режимі (дуже корисно для експериментів і вирішення простих задач);
- Переносимість програми (характеристика більшості інтерпретованих мов);
- Стандартний дистрибутив має просте, але в той же час досить потужне середовище розробки під назвою IDLE, яка написана на мові **Python**;
- Чистий синтаксис (блоки мають бути виділені відступами).

Програмний продукт містить у собі стандартні модулі, та багто інших, кожен з яких, повинен бути присутній для виконання певних задач.

Список модулів: **os, sys, math, datetime, random, re, json, csv, sqlite3, unittest, xml, http, socket, email, smtplib, tkinter, numpy, pandas, matplotlib, scikit-learn, tensorflow**, та багато інших.

Головну роль у розпізнаванні мовлення, відіграє бібліотека **speech recognition**. Саме вона містить у собі важливі властивості, функції та моделі для розпізнавання слів. Для розробки програмного забезпечення було вибрано українську модель мовлення розширеного типу, тобто вона ще враховує фразеологізми, окличні слова та інше. Дана модель займає 3ГБ пам’яті, але ця модель є зручною так як вона зберігається у хмарному сховищі.

3.2 Розпізнавання та обробка мови

3.2.1 Алгоритми розпізнавання мовлення

Розпізнавання та обробка мови – це важлива складова частина сучасних голосових асистентів, яка включає декілька етапів:

- збір аудіосигналу;
- його аналіз;
- очищення шуму;
- розпізнавання слів та інтерпретацію команд.

Основна мета цієї технології – перетворення мовлення в текст, який може бути оброблений комп'ютером для виконання відповідних дій.

Існують наступні алгоритми розпізнавання мови:

- збір аудіо сигналу;
- фонетичний аналіз;
- акустичне моделювання;
- лінгвістичне моделювання;
- моделі на основі нейронних мереж;
- трансформерні моделі;
- постобробка.

Збір аудіо сигналу. На першому етапі система розпізнавання мовлення приймає аудіосигнал з мікрофона та перетворює його у цифровий формат. Це передбачає оцифрування аналогового сигналу, що здійснюється за допомогою аналогово-цифрового перетворювача (АЦП).

Аудіосигнал, записаний мікрофоном, зазвичай містить багато шумів, які потрібно фільтрувати. Для цього використовуються фільтри низької і високої частоти, що видаляють непотрібні компоненти сигналу. Оцифрований сигнал представляється у вигляді дискретної послідовності чисел, які відповідають амплітудам звукових хвиль у різні моменти часу. Щоб покращити якість звуку, застосовуються методи нормалізації та попередньої обробки, такі як усунення фонових шумів та ехо. Цей етап є критично важливим, оскільки якість оцифрованого сигналу безпосередньо впливає на точність подальшого розпізнавання мовлення.

Фонетичний аналіз. Фонетичний аналіз полягає у розділенні аудіосигналу на менші звукові одиниці – фонеми. Фонема – це найменша звукова одиниця, яка може змінити значення слова. Для кожної мови існує певний набір фонем, ідентифікація яких є основною задачею цього етапу.

Для фонетичного аналізу використовуються методи спектрального аналізу, такі як швидке перетворення Фур'є (**FFT**), яке перетворює часовий сигнал у частотну область. Спектрограмми, що отримуються при цьому, дозволяють візуально ідентифікувати фонеми. Крім того, застосовуються мел-частотні кепстральні коефіцієнти (**MFCC**), які забезпечують кращу відповідність людському слуховому сприйняттю, виділяючи особливості, важливі для розпізнавання мовлення.

Акустичне моделювання. Акустичні моделі використовують ймовірнісні методи для визначення ймовірності того, що певний звуковий сигнал відповідає певній фонемі. Одним з найбільш поширених підходів є використання прихованих марковських моделей (HMM). **HMM –** представляє мовлення як послідовність прихованих станів, кожен з яких відповідає певній

фонемі або їх комбінації. **HMM** дозволяє враховувати різні варіації вимови, акценти та шуми.

Для кожної фонеми створюється модель, яка описує ймовірність переходів між різними станами та ймовірність спостереження певних акустичних характеристик у цих станах. Навчання **HMM** здійснюється на основі великої кількості аудіозаписів з транскрипціями, що дозволяє системі розпізнавати нові зразки мовлення з високою точністю.

Лінгвістичне моделювання. Лінгвістичні моделі використовуються для контекстного аналізу та покращення точності розпізнавання мовлення. Вони включають синтаксичний та семантичний аналіз, який допомагає розуміти значення слів у контексті фрази. Одним з основних підходів є використання n-грам моделей, які враховують ймовірність появи слова в залежності від попередніх n слів.

Крім того, застосовуються більш складні моделі на основі нейронних мереж, такі як рекурентні нейронні мережі (**RNN**) і трансформери, які здатні враховувати довготривалі залежності у тексті. Ці моделі дозволяють покращити розпізнавання мовлення, зменшуючи кількість помилок при розпізнаванні складних або багатозначних слів.

Моделі на основі нейронних мереж. Моделі на основі нейронних мереж, сучасні системи розпізнавання мовлення часто використовують глибокі нейронні мережі (**DNN**) та рекурентні нейронні мережі (**RNN**), такі як **LSTM** (довготривала короткочасна пам'ять). Ці моделі можуть обробляти великі обсяги даних і забезпечувати високу точність розпізнавання.

Глибокі нейронні мережі здатні навчатися складним паттернам у великих наборах даних, що дозволяє їм розпізнавати мовлення з високою точністю навіть у складних умовах. Рекурентні нейронні мережі, особливо **LSTM** і **GRU** (гейтовані рекурентні блоки) представленні на рисунку 3.1, здатні враховувати попередній контекст, що особливо важливо для обробки мовлення.

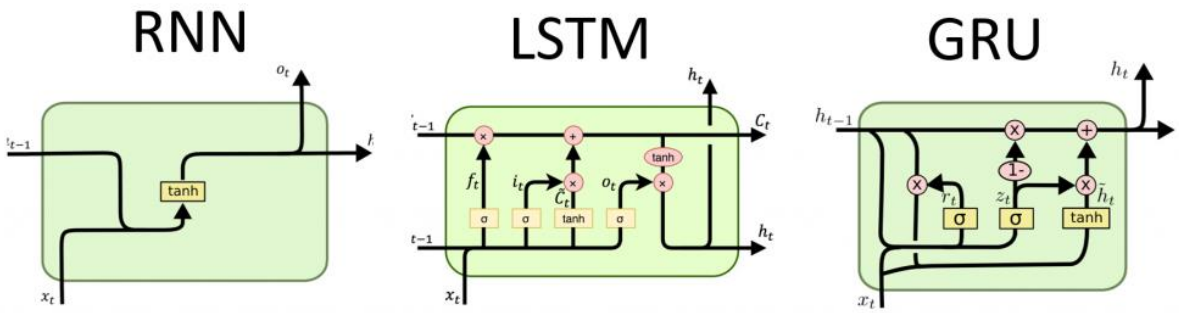

Рисунок 3.1 – «Принципи роботи RNN, LSTM, GRU»

Трансформерні моделі. Трансформерні моделі, новітні моделі, такі як трансформери (наприклад, **BERT**, **GPT**), можуть враховувати контекст на рівні речень та текстів, що дозволяє їм розуміти складні структури мовлення і надавати точні результати. Вони здатні навчатися на великих корпусах текстів і мають високу здатність до узагальнення.

Трансформерні моделі використовують механізм уваги, який дозволяє моделі зосереджуватися на важливих частинах вхідного тексту, що забезпечує більш точне розпізнавання та розуміння мовлення. Ці моделі є основою сучасних систем обробки природної мови (**NLP**) і забезпечують високу якість розпізнавання мовлення у різних умовах.

Постобробка. Після того, як мовлення перетворено у текст, здійснюється постобробка, яка включає корекцію помилок, аналіз синтаксису та семантики. Постобробка може включати виправлення орфографічних помилок, нормалізацію тексту та аналіз семантики для кращого розуміння контексту. Це дозволяє підвищити точність і зрозумілість розпізнаного тексту, що є особливо важливим для практичних застосувань.

3.2.2 Методи обробки природної мови (NLP)

Природна обробка мови (**NLP**) є критично важливою для створення голосових асистентів, оскільки вона дозволяє системам розуміти, інтерпретувати та реагувати на людську мову.

У цьому розділі розглянемо методи обробки природної мови, які використовуються для забезпечення високоякісного функціонування голосового асистента, включаючи розуміння мови, генерацію відповідей та навчання з часом (див. рис.3.2).

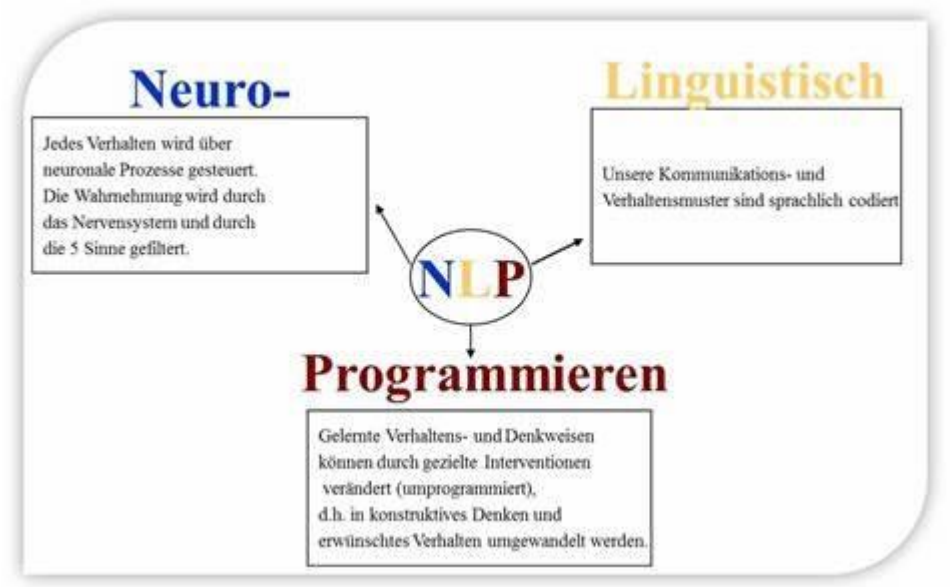

Рисунок 3.2 – «Розшифрування абревіатури NLP»

Розуміння мови є одним з основних завдань **NLP**. Воно включає в себе кілька етапів: токенізацію, лематизацію, стемінг, аналіз частин мови, розпізнавання сутностей, синтаксичний та семантичний аналіз. Токенізація розбиває текст на окремі слова або фрази. Лематизація та стемінг зводять слова до їх основної форми, що полегшує аналіз. Аналіз частин мови визначає граматичні категорії слів, що допомагає зрозуміти структуру речення. Розпізнавання сутностей виділяє важливі елементи в тексті, такі як імена чи дати.

Синтаксичний аналіз визначає граматичні зв'язки між словами, а семантичний аналіз допомагає зрозуміти значення слів і фраз у контексті. Генерація відповідей є наступним етапом після розуміння мови. Використовуються шаблонні відповіді для стандартних запитів, але для більш складних завдань застосовуються нейронні мережі. Моделі на основі

трансформерів, такі як **GPT**, можуть генерувати природні та контекстно залежні відповіді.

Моделі секвенційних послідовностей, такі як **LSTM**, перетворюють вхідний текст на вихідний. Розуміння контексту діалогу забезпечується за допомогою механізмів уваги, що дозволяє враховувати попередні частини розмови.

Навчання з часом є важливою складовою розвитку голосового асистента. Зворотний зв'язок від користувачів допомагає системі вчитися на помилках і покращувати точність відповідей. Алгоритми машинного навчання використовуються для безперервного аналізу даних і вдосконалення моделей. Онлайн навчання дозволяє моделям адаптуватися до нових умов і запитів в режимі реального часу. Метод A/B тестування застосовується для експериментального порівняння різних підходів до обробки запитів і вибору найбільш ефективних.

Інтеграція **NLP** у голосовий асистент дозволяє не тільки розуміти та генерувати текст, але й вчитися на основі взаємодії з користувачами, адаптуватися до їхніх потреб та надавати все більш релевантні й точні відповіді. Це робить асистента не тільки інструментом для виконання завдань, але й справжнім помічником, здатним розуміти та підтримувати контекст діалогу.

3.2.3 Інтеграція з розпізнаваннями мовлення та NLP-сервісами

Інтеграція з розпізнавачами мовлення та NLP-сервісами є ключовим елементом для створення високоякісних голосових асистентів. Вона дозволяє використовувати потужні зовнішні сервіси для точного розпізнавання мовлення, розуміння контексту і генерації відповідей. У сучасному світі існує багато сервісів, які можуть значно покращити функціональність голосових асистентів.

Google Speech Recognition. Одним із провідних сервісів є **Google Speech-to-Text**. Цей сервіс підтримує понад 120 мов і діалектів, що робить його ідеальним для використання у міжнародних проектах. Google використовує передові алгоритми машинного навчання для точного розпізнавання мовлення, навіть у складних умовах, таких як фоновий шум або невнятна мова.

Крім того, цей сервіс надає можливість налаштування параметрів для оптимізації точності та швидкості розпізнавання, що дозволяє адаптувати його під конкретні потреби користувачів.

Також найголовніше те, що саме цей сервіс було використано у розробці програмного забезпечення, так як він надзвичайно зручний для використання, та швидкий для роботи.

IBM Watson. Даний сервіс є ще одним потужним інструментом для розпізнавання мовлення та обробки природної мови. **Watson** пропонує широкий спектр можливостей, включаючи розпізнавання мовлення, аналіз тексту, виявлення сутностей та настроїв, а також генерацію природної мови. Однією з ключових переваг **Watson** є його здатність обробляти великі обсяги даних у реальному часі, що робить його ідеальним для інтеграції у голосових асистентів. Крім того, **Watson** підтримує інтеграцію з іншими сервісами **IBM**, що дозволяє створювати комплексні рішення для взаємодії з користувачами.

Microsoft Azure Cognitive Services. Сервіс від **Microsoft** також пропонує широкий спектр інструментів для розпізнавання мовлення та обробки природної мови. **Azure Speech Service** забезпечує високу точність розпізнавання мовлення та підтримує різні мови. Крім того, **Azure Language Understanding** (**LUIS**) дозволяє створювати моделі для розуміння намірів користувачів і виявлення сутностей у тексті. **Azure** також пропонує можливості для аналізу настроїв і тональності тексту, що дозволяє створювати більш інтерактивні та персоналізовані відповіді. Використання цього сервісу може значно покращити користувацький досвід та оптимізувати бізнес-процеси.

Amazon Web Services (**AWS**) надає сервіс **Amazon Transcribe** для розпізнавання мовлення та **Amazon Comprehend** для обробки природної мови. **Amazon Transcribe** забезпечує точне розпізнавання мовлення з підтримкою багатьох мов, а також можливість обробки аудіофайлів у реальному часі. **Amazon Comprehend** пропонує аналіз тексту, включаючи виявлення сутностей, аналіз настроїв та класифікацію тексту. Інтеграція з іншими сервісами **AWS** дозволяє створювати масштабовані рішення для обробки мовлення та тексту.

Baidu Speech Recognition є провідним сервісом для розпізнавання мовлення в Китаї. Цей сервіс використовує передові алгоритми глибокого навчання для забезпечення високої точності розпізнавання мовлення китайською мовою. **Baidu** також надає можливості для обробки природної мови, включаючи аналіз тексту і виявлення сутностей. Це робить його ідеальним вибором для проектів, орієнтованих на китайськомовну аудиторію.

Wit.ai. Даний сервіс, який належить **Facebook**, є безкоштовним сервісом для розпізнавання мовлення та обробки природної мови. **Wit.ai** дозволяє створювати інтелектуальні боти та голосові асистенти, що можуть розуміти та відповідати на запити користувачів. Сервіс підтримує інтеграцію з багатьма платформами, такими як **Messenger**, **Slack**, і **IoT**-пристрої, що робить його універсальним інструментом для розробників.

Nuance Communications. Цей сервіс є відомим постачальником технологій розпізнавання мовлення та обробки природної мови. **Nuance Dragon** є одним з найпопулярніших продуктів для розпізнавання мовлення, який забезпечує високу точність і швидкість. **Nuance** також пропонує рішення для аналізу тексту та взаємодії з користувачами в різних галузях, таких як охорона здоров'я та фінанси.

Houndify. Розроблений компанією **SoundHound**, є потужним інструментом для створення голосових асистентів. **Houndify** використовує технологію **Deep Meaning Understanding**, що дозволяє розпізнавати складні

та багатозначні запити користувачів. Сервіс підтримує широкий спектр мов і забезпечує інтеграцію з різними платформами та пристроями.

Kaldi є відкритим програмним забезпеченням для розпізнавання мовлення, що забезпечує високу гнучкість і точність. **Kaldi** підтримує інтеграцію з різними мовними моделями та інструментами для обробки природної мови, що дозволяє створювати індивідуальні рішення для конкретних потреб. Відкрита архітектура **Kaldi** дозволяє налаштовувати його під специфічні вимоги проекту, що робить його популярним серед дослідників та розробників.

Інтеграція з цими сервісами дозволяє вашому голосовому асистенту забезпечити високу точність розпізнавання мовлення, ефективну обробку тексту та персоналізовані відповіді на запити користувачів. Це підвищує якість взаємодії та робить асистента кориснішим і зручнішим у використанні. Використання передових технологій і сервісів дозволяє створювати потужні рішення для різних галузей і забезпечувати найвищий рівень сервісу для кінцевих користувачів.

3.2.4 Оптимізація точності та швидкості реакції асистента

Для забезпечення високої точності розпізнавання мовлення та швидкої відповіді асистента використовуються різноманітні методи та техніки. Перш за все, це використання потужних алгоритмів машинного навчання, які навчаються на великому обсязі даних для розпізнавання мовлення. Такі алгоритми, як правило, використовуються для перетворення мовлення в текстову форму, що може бути подальше оброблена асистентом.

Для підвищення точності розпізнавання використовуються також моделі глибокого навчання, які дозволяють асистентові "вивчати" особливості мовлення та підганяти свою роботу під конкретного користувача. Це дозволяє досягти високої точності навіть у складних умовах, наприклад, при розпізнаванні акцентів чи різних мовленнєвих особливостей.

Щоб забезпечити швидку відповідь асистента, використовуються техніки оптимізації швидкості обробки запитів. Це може включати в себе використання спеціалізованих апаратних рішень, таких як графічні процесори, для прискорення обчислень. Також важливою є оптимізація програмного забезпечення для швидкого доступу до даних та ефективної обробки запитів.

Загалом, поєднання потужних алгоритмів машинного навчання, глибокого навчання та оптимізації програмного та апаратного забезпечення дозволяє забезпечити високу точність розпізнавання мовлення та швидку відповідь асистента, що робить його більш ефективним та корисним для користувачів.

Одним з ключових методів для забезпечення високої точності розпізнавання мовлення є використання різноманітних нейронних мереж, таких як рекурентні нейронні мережі (**RNN**) або трансформери, які дозволяють моделі асистента враховувати контекст мовлення. Для підвищення точності можуть використовуватися алгоритми адаптації до голосу користувача, які навчаються розпізнавати індивідуальні особливості голосу та мовлення кожного користувача. Додатковою технікою для забезпечення точності є використання акустичних моделей, які дозволяють асистентові краще розпізнавати звуки та інтонацію мовлення.

3.3 Інтерфейс та взаємодія з користувачем

3.3.1 Принципи проектування користувацького інтерфейсу

Проектування користувацького інтерфейсу для голосового асистента базується на кількох ключових принципах, які сприяють створенню ефективних і зручних для користувачів систем.

Простота і зрозумілість є основними аспектами будь-якого користувацького інтерфейсу, а для голосового асистента вони набувають ще

більшого значення. Користувачі повинні легко розуміти, як взаємодіяти з асистентом, навіть без технічних знань. Інтерфейс має бути інтуїтивно зрозумілим, команди - логічними та послідовними, а відповіді асистента - чіткими та лаконічними.

Послідовність у всьому є ще одним важливим принципом. Усі елементи інтерфейсу повинні працювати однаково, щоб користувачі могли передбачати, як система відреагує на їхні дії. Це створює відчуття надійності та стабільності, що особливо важливо для нових користувачів, які лише знайомляться з функціональністю голосового асистента.

Зворотний зв'язок грає критичну роль у взаємодії користувача з голосовим асистентом. Користувачі повинні отримувати негайну та чітку відповідь на свої запити, щоб розуміти, що їхні команди були почуті і правильно інтерпретовані. Голосовий асистент повинен надавати зворотний зв'язок у зрозумілій формі, без зайвої технічної термінології.

Доступність і інклюзивність є важливими для забезпечення того, щоб голосовий асистент міг бути використаний якомога ширшою аудиторією. Це включає підтримку різних мов і діалектів, а також можливість роботи з користувачами з обмеженими можливостями. Важливо враховувати різні акценти, дефекти мови та інші фактори, що можуть впливати на взаємодію з системою.

Контекстність відповідає за те, щоб голосовий асистент розумів, у якому контексті використовуються команди. Це дозволяє системі надавати більш релевантні та точні відповіді. Наприклад, якщо користувач запитує про погоду, асистент повинен розуміти, що він має на увазі поточне місце розташування користувача або конкретне місто, яке користувач часто згадує.

Зручність використання також включає можливість персоналізації. Голосовий асистент повинен мати можливість адаптуватися до уподобань і потреб користувача, запам'ятовувати його попередні запити і надавати відповідну інформацію на основі цих даних. Персоналізація сприяє більш глибокій інтеграції асистента в повсякденне життя користувача.

Безпека та конфіденційність є критично важливими аспектами, оскільки голосові асистенти часто мають доступ до чутливої інформації. Користувачі повинні бути впевнені, що їхні дані захищені і використовуються лише для надання необхідних послуг. Прозорість у питаннях збору та використання даних допомагає створити довіру до системи.

Емоційний зв'язок з користувачем також відіграє важливу роль. Голосовий асистент повинен не лише виконувати функціональні завдання, а й підтримувати позитивний емоційний стан користувача. Це може включати використання дружнього тону, гумору або навіть емпатії в відповідях асистента.

Врахування цих принципів при розробці користувацького інтерфейсу для голосового асистента допомагає створити систему, яка є не лише функціональною, але й приємною у використанні. Це сприяє більш глибокій інтеграції асистента в повсякденне життя користувачів, забезпечуючи їм комфорт і задоволення від взаємодії з технологією.

3.3.2 Розробка голосового інтерфейсу користувача (VUI)

Розробка голосового інтерфейсу користувача (VUI) є комплексним процесом, який включає кілька основних етапів: визначення сценаріїв взаємодії, дизайн діалогів, а також врахування різних акцентів і діалектів. Кожен з цих етапів критично важливий для створення ефективної і зручної системи, що забезпечує природну і приємну взаємодію між користувачем і асистентом.

Першим і дуже важливим кроком у розробці VUI є визначення сценаріїв взаємодії. Це процес опису того, як користувачі будуть взаємодіяти з голосовим асистентом у різних контекстах. Для цього необхідно провести дослідження користувачів, щоб зрозуміти їхні потреби та звички. Це може включати опитування, інтерв'ю та аналіз поведінки користувачів. Наступним кроком є визначення основних задач, які користувачі хочуть вирішувати за

допомогою голосового асистента, наприклад, пошук інформації, керування розкладом або відтворення музики. Після цього створюються конкретні сценарії використання, які детально описують кроки користувача та асистента. Наприклад, сценарій замовлення їжі може включати запит користувача, уточнення деталей асистентом, підтвердження замовлення та його завершення.

Другим етапом є дизайн діалогів, який є ключовим для створення природної і ефективної взаємодії. На цьому етапі розробляються діалогові моделі, які визначають структуру діалогу для кожного сценарію взаємодії. Це включає розробку початкових запитів, відповідей асистента, уточнень та завершення діалогу. Написання скриптів є наступним кроком, де створюються текстові скрипти для всіх можливих варіантів діалогу. Важливо забезпечити, щоб відповіді асистента були короткими, зрозумілими та природними, використовуючи природні паузи, інтонації і вирази. Крім того, необхідно розробити механізми обробки помилок, щоб користувач міг легко зрозуміти, що асистент не розпізнав його запит або він не був зрозумілий. Це включає надання підказок або повторних запитів для уточнення.

Врахування різних акцентів та діалектів є надзвичайно важливим для забезпечення доступності голосового асистента для широкої аудиторії. Це досягається шляхом збору великої кількості мовних даних від користувачів з різними акцентами і діалектами. Це допомагає тренувати моделі розпізнавання мови на більш широкому спектрі мовних варіацій. Використання сучасних методів машинного навчання і нейронних мереж для створення мовних моделей, здатних розпізнавати і правильно інтерпретувати різні акценти та діалекти, є ключовим аспектом. Регулярне тестування системи з користувачами, які мають різні акценти та діалекти, дозволяє вносити необхідні корективи на основі отриманих результатів, налаштовуючи чутливість розпізнавання мови та адаптуючи моделі.

Після розробки всіх компонентів VUI важливо інтегрувати їх у єдину систему і провести ретельне тестування. Інтеграція включає об'єднання

мовних моделей, сценаріїв взаємодії та діалогових скриптів у єдину систему, забезпечуючи сумісність всіх компонентів і їх коректну роботу разом. Функціональне тестування передбачає перевірку всіх функцій голосового асистента на різних сценаріях взаємодії, включаючи тестування всіх можливих діалогів, обробки помилок та реакції на різні акценти і діалекти. Юзабіліті-тестування проводиться з реальними користувачами для оцінки зручності використання системи, збору зворотного зв'язку і внесення необхідних покращень на основі отриманих даних. Крім того, важливо провести тестування на безпеку, перевіряючи систему на відповідність вимогам безпеки і конфіденційності, забезпечуючи захист даних користувачів і надійну роботу асистента.

У висновку, процес розробки голосового інтерфейсу користувача (VUI) є багатогранним і включає кілька ключових етапів: визначення сценаріїв взаємодії, дизайн діалогів, врахування різних акцентів та діалектів, інтеграцію та тестування. Врахування всіх цих аспектів дозволяє створити систему, яка не лише виконує свої функції, але й забезпечує позитивний досвід користувача, сприяючи глибокій інтеграції технології в повсякденне життя.

3.3.3 Тестування взаємодії з користувачами

Тестування взаємодії з користувачами є критично важливим етапом у розробці голосового асистента, оскільки дозволяє оцінити його ефективність, зручність та точність у реальних умовах. Основні методи тестування, які використовуються для перевірки взаємодії користувачів з голосовим асистентом, включають різноманітні підходи, кожен з яких має свої переваги.

Тестування передбачає залучення реальних користувачів для оцінки зручності використання голосового асистента. Користувачам пропонується виконати певні завдання або сценарії взаємодії з асистентом. Під час тестування збирається зворотний зв'язок, який допомагає ідентифікувати проблемні місця, непорозуміння та труднощі, що виникають у процесі

використання. Юзабіліті-тестування можна проводити як у лабораторних умовах, так і у природному середовищі користувачів. А/Б тестування використовується для порівняння двох або більше варіантів інтерфейсу або функціональності.

Користувачам пропонуються різні версії системи, і на основі зібраних даних аналізується, який варіант є більш ефективним. Це допомагає визначити оптимальні рішення для інтерфейсу та взаємодії. Когортний аналіз передбачає сегментацію користувачів на групи (когорти) за певними характеристиками, наприклад, за віком, досвідом або частотою використання. Аналіз поведінки кожної когорти допомагає зрозуміти, як різні групи користувачів взаємодіють з голосовим асистентом, і виявити специфічні потреби та проблеми.

Контекстне тестування полягає у проведенні тестів у реальних умовах використання. Це дозволяє оцінити, як голосовий асистент функціонує в природному середовищі користувачів, де можуть виникати додаткові фактори, такі як фоновий шум, різні акценти та діалекти. Це тестування дає змогу зібрати дані про ефективність і зручність використання у повсякденному житті.Щоденники користувачів є методом, де користувачі ведуть записи про свій досвід взаємодії з голосовим асистентом протягом певного періоду. Це допомагає отримати глибше розуміння того, як користувачі використовують систему на щоденній основі, які проблеми вони стикаються і які функції вони вважають найбільш корисними.

Зворотний зв'язок і опитування є ефективними інструментами для збору якісної інформації від користувачів. Користувачам пропонуються анкети або форми зворотного зв'язку, де вони можуть описати свій досвід використання, висловити свої думки та запропонувати покращення. Це допомагає виявити суб'єктивні аспекти користувацького досвіду.

Аналіз журналів і даних взаємодії включає збір і аналіз автоматично зібраних даних про взаємодію користувачів з системою. Це можуть бути дані про частоту використання різних команд, час виконання завдань, помилки розпізнавання мови тощо. Аналіз цих даних дозволяє виявити тенденції та

проблеми у взаємодії, які можуть бути непомітні під час інших типів тестування.

Ітеративне тестування і вдосконалення є підходом, при якому тестування проводиться на кожному етапі розробки з подальшими покращеннями на основі отриманих результатів. Це дозволяє постійно вдосконалювати систему і забезпечувати високу якість кінцевого продукту. Такий підхід допомагає вчасно виявляти і виправляти помилки, а також адаптувати систему до змінюваних потреб користувачів.

Комбінація різних методів тестування забезпечує комплексний підхід до оцінки ефективності взаємодії користувачів з голосовим асистентом. Це дозволяє створити систему, яка не лише виконує свої функції, але й забезпечує позитивний користувацький досвід, що сприяє її широкому використанню і задоволеності користувачів.

3.3.4 Забезпечення доступності та інклюзивності голосового асистента

Забезпечення доступності та інклюзивності голосового асистента є ключовим завданням, яке вимагає врахування різноманітних потреб і обмежень користувачів. Це дозволяє створити продукт, який може використовуватися максимально широкою аудиторією, включаючи людей з обмеженими можливостями, різними мовними і культурними особливостями. Нижче наведені основні підходи та методи, які ми плануємо використовувати для забезпечення доступності та інклюзивності голосового асистента.

Підтримка різних мов і діалектів. Голосовий асистент повинен бути здатний розпізнавати та інтерпретувати команди, виголошені різними мовами і діалектами. Для цього планується збір великої кількості мовних даних від носіїв різних мов і акцентів, а також використання сучасних алгоритмів машинного навчання і нейронних мереж для тренування мовних моделей. Це

допоможе забезпечити точне розпізнавання мови і правильне розуміння команд. Адаптація для людей з вадами зору.

Голосовий асистент може стати важливим інструментом для людей з вадами зору, надаючи можливість керувати пристроями і отримувати інформацію за допомогою голосових команд. Для цього необхідно забезпечити зручний і зрозумілий інтерфейс, який дозволить легко взаємодіяти з асистентом без необхідності використання зорових сигналів. Крім того, важливо підтримувати функції читання тексту вголос та опису візуальних елементів. Інтеграція з допоміжними технологіями.

Голосовий асистент повинен бути сумісний з різноманітними допоміжними технологіями, такими як пристрої для людей з вадами слуху, клавіатури з великими кнопками, спеціалізоване програмне забезпечення для людей з обмеженими фізичними можливостями тощо. Це дозволить користувачам використовувати голосовий асистент у поєднанні з пристроями, які вони вже використовують для полегшення свого повсякденного життя.

Підтримка людей з вадами слуху. Для користувачів з вадами слуху важливо забезпечити можливість взаємодії з асистентом за допомогою тексту. Це може включати функції розпізнавання мови і автоматичної транскрипції голосових команд у текст, а також можливість введення команд за допомогою тексту і отримання відповідей у текстовому вигляді.Зручність для людей з когнітивними розладами. Для забезпечення доступності голосового асистента для людей з когнітивними розладами, важливо створити інтуїтивно зрозумілий і простий інтерфейс. Це включає використання простих і зрозумілих команд, уникнення складних структур діалогу, а також можливість повторного пояснення команд і функцій асистента.

Налаштування і персоналізація. Надання користувачам можливості налаштовувати інтерфейс голосового асистента відповідно до своїх потреб є важливим аспектом інклюзивності. Це може включати налаштування гучності, темпу мови, використання різних голосів і мовних стилів. Персоналізація

допомагає зробити асистента більш зручним і приємним у використанні для різних категорій користувачів.

Проведення юзабіліті-тестування з різними групами користувачів. Важливо проводити тестування з участю користувачів, які мають різні потреби і обмеження. Це допоможе ідентифікувати проблемні місця і внести необхідні покращення для забезпечення доступності і зручності використання голосового асистента для всіх категорій користувачів.

Постійне оновлення і вдосконалення. Технології швидко розвиваються, і потреби користувачів можуть змінюватися. Тому важливо постійно оновлювати і вдосконалювати голосовий асистент, враховуючи нові дані, відгуки користувачів і розвиток технологій. Регулярне оновлення мовних моделей, функцій і інтерфейсу допоможе підтримувати високу якість і актуальність продукту.

Забезпечення доступності та інклюзивності голосового асистента вимагає комплексного підходу, що включає технічні, функціональні і користувацькі аспекти. Врахування різноманітних потреб користувачів і використання сучасних технологій допомагає створити продукт, який є корисним і зручним для всіх.

4. ЕКОНОМІЧНИЙ РОЗДІЛ

Основна мета економічної частини дипломного проекту полягає в проведенні розрахунків, спрямованих на визначення економічної ефективності розробки програмного забезпечення "Голосовий асистент". Це дозволить прийняти обґрунтоване рішення щодо подальшого розвитку цього продукту.

Для того, щоб розрахувати вартість розробки та реалізації програмного забезпечення необхідно виконати наступні етапи:

- описати технологічний процес розробки із зазначенням трудомісткості кожної операції;
- визначити суму витрат на оплату праці основного і допоміжного персоналу, включаючи відрахування на соціальні заходи;
- визначити суму матеріальних затрат;
- обчислити витрати на електроенергію для науково-виробничих цілей;
- розрахувати транспортні витрати;
- нарахувати суму амортизаційних відрахувань;
- визначити суму накладних витрат;
- скласти кошторис та визначити собівартість обслуговування програмного продукту;
- розрахувати ціну розробки та реалізації програмного забезпечення «Голосовий асистент»;
- визначити економічну ефективність та термін окупності продукту.

4.1 Визначення стадій технологічного процесу та загальної тривалості розробки та реалізації програмного забезпечення «Голосовий асистент»

Для того, щоб визначити загальну тривалість розробки та реалізації програмного забезпечення, необхідно створити таблицю, у якій вказати час,

необхідний для кожної окремої операції, дані витрат часу по окремих операціях зведемо у таблицю 4.1.

Таблиця 4.1 — Середній час розробки та реалізації програмного продукту «Голосовий асистент», та стадії технологічного процесу

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1.	Постановка задачі	Керівник	35
2.	Збір інформації	Інженер	20
3.	Розробка проекту	Інженер	168
4.	Підготовка документації	Керівник	35
5.	Доставка обладнання	Інженер	18
6.	Тестування програмного забезпечення	Інженер	25
7.	Здача в експлуатацію	Інженер	5
Разом			306

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

Заробітна плата (оплата праці) у розумінні Закону України "Про оплату праці" – це винагорода, обчислена, як правило, у грошовому виразі, яку власник або уповноважений ним орган виплачує працівникові за виконану ним роботу.

Структура заробітної плати складається з наступних її видів:

- **основна заробітна плата** — винагорода за виконану роботу відповідно до встановлених норм праці (норми часу,

виробітку, обслуговування, посадові обов'язки). Вона встановлюється у вигляді тарифних ставок (окладів) і відрядних розцінок для робітників та посадових окладів для службовців;

- **додаткова заробітна плата** — винагорода за працю понад установлені норми, за трудові успіхи та винахідливість і за особливі умови праці. Вона включає доплати, надбавки, гарантійні і компенсаційні виплати, передбачені чинним законодавством; премії, пов'язані з виконанням виробничих завдань і функцій
- **інших заохочувальні та компенсаційні виплати**. До них належать виплати у формі винагород за підсумками роботи за рік, премії за спеціальними системами і положеннями, виплати в рамках грантів, компенсаційні та інші грошові і матеріальні виплати, які не передбачені актами чинного законодавства або які провадяться понад встановлені зазначеними актами норми.

Основна заробітна плата розраховується за формулою:

$$З_{осн.} = \sum Т_с \cdot К_г, \quad (4.1)$$

де $Т_с$ – тарифна ставка, грн.;

$К_г$ – кількість відпрацьованих годин.

Отже, включаючи рекомендовані тарифні ставки, потрібно встановити тарифну ставку для керівника 100 грн./год. — для інженера 70 грн./год.

$$З_{осн.} = 100 \cdot 70 + 70 \cdot 236 = 23\,520 \text{ (грн)}$$

Додаткова заробітна плата від основної суми становить 10–15%.

$$З_{дод.} = З_{осн.} \cdot К_{дод.} \quad (4.2)$$

де $К_{дод.}$ – коефіцієнт додаткових виплат працівникам, 0,1–0,15.

$$К_{дод.} = 23\,520 \cdot 0.1 = 2\,352 (грн)$$

Звідси загальні витрати на оплату праці ($B_{о.п.}$) визначаються за формулою:

$$В_{о.п.} = З_{осн.} + З_{дод.} \quad (4.3)$$

$$В_{о.п.} = 23\,520 + 2\,352 = 25\,872\ (грн.)$$

Провівши дані економічні операції, слід визначити відрахування на соціальні заходи, а саме:

Єдиний соціальний внесок — 22%

Отже, точна загальна сума відрахувань на соціальні заходи буде становити:

$$В_{с.з.} = ФОП \cdot 0,22 \quad (4.4)$$

де ФОП – фонд оплати праці, грн.

$$В_{с.з.} = 25\,872 \cdot 0,22 = 5\,691,84\ (грн.)$$

Провівши відповідні економічні розрахунки витрат на оплату праці, тепер відомо, яка саме загальна сума необхідна для реалізації виплат усім працівникам.

Таблиця 4.2 — Економічні розрахунки щодо оплати праці та відрахування на соціальні заходи

№ п/п	Категорія працівників	Основна заробітна плата, грн.			Додаткова заробітна плата, грн.	Нарахув. на ФОП, грн.	Всього витрати на оплату праці, грн.
		Тарифна ставка, грн.	К-сть від-працьов. год.	Фактично нарах. з/пл., грн.			
А	*Б*	*1*	*2*	*3*	*4*	*5*	*6*
1	Керівник	100	70	7 000	700	---	---
2	Інженер	70	236	16 520	1 652	---	---
Разом				23 520	2 352	5 691,84	31 563,84

4.3 Розрахунок витрат на електроенергію

Так як для розробки програми необхідне певне обладнання, яке працює за допомогою електроенергії, отже витрати на електроенергію теж слід врахувати.

Затрати на електроенергію одиниці обладнання визначаються за формулою:

$$З_е = W \cdot T \cdot S \qquad (4.5)$$

де W – необхідна потужність, кВт;

T – кількість годин роботи обладнання;

S – вартість кіловат-години електроенергії.

Маючи формулу, можна підставити свої значення, отже для розробки програмного забезпечення використовується один ПК з потужністю W = 0,6 кВт.

$$З_е = 0,6 \cdot 288 \cdot 7 = 1210,00 \text{ грн}$$

4.4 Розрахунок суми амортизаційних відрахувань

Основною особливістю використання основних засобів у виробничому процесі є їхнє відновлення. Для того, щоб відновити засоби праці в натуральній формі, необхідно відшкодувати їх вартість, що здійснюється через амортизацію.

Амортизація — це процес перенесення вартості основних засобів на вартість новоствореної продукції з метою їх повного відшкодування.

Для визначення амортизаційних відрахувань необхідно застосувати формулу:

$$А = \frac{Б_В \cdot Н_А}{100\%} \quad (4.6)$$

де А – амортизаційні відрахування за звітний період, грн.;

БВ – балансова вартість групи основних фондів на початок звітного періоду, грн.;

НА – норма амортизації, %.

В результаті, для розробки даного програмного продукту використовується один комп’ютер (вартість якого становить 15500 грн.), який працює 288 годин. Отже отримавши дану формулу, можна провести такі економічні розрахунки:

$$А = 15500 \cdot 0,04 \cdot 288 / 150 = 1190,4 \text{ (грн.)}$$

4.5 Обчислення накладних витрат

Обчислення накладних витрат включає підрахунок всіх витрат, які не можуть бути прямо приписані до конкретного продукту чи послуги, але які необхідні для забезпечення діяльності компанії.

В залежності від організаційно-правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 20–60% від суми основної та додаткової заробітної плати працівників.

Відповідна до цього формула:

$$Н_в = В_{о.п.} \cdot 0{,}2 \ldots 0{,}6 \quad (4.7)$$

де НВ – накладні витрати.

Отже, підставивши до цієї формули власні дані, ми отримаєм такий результат:

$$Н_в = 25\,872 \cdot 0{,}4 = 10\,348{,}80 \text{ грн.}$$

4.6 Складання кошторису витрат та визначення собівартості НДР

Результати економічних розрахунків проведених вище зведемо у таблицю 4.3

Таблиця 4.3 - Кошторис витрат на НДР

Зміст витрат	Сума, грн.	В % до заг. суми
Витрати на оплату праці (основну і додаткову заробітну плату)	25 872	58,38
Відрахування на соціальні заходи	5 691,84	12,84
Витрати на електроенергію	1210	2,73
Амортизаційні відрахування	1190,40	2,68
Накладні витрати	10 358,80	23,37
Собівартість	44 313,04	100

Собівартість ($С_В$) НДР розрахуємо за формулою:

$$С_в = В_{о.п.} + В_{с.з.} + З_е + Т_в + А + Н_в \quad (4.8)$$

$$С_в = 44\,313{,}04\ (грн.)$$

4.7 Розрахунок ціни НДР

Ціну НДР можна визначити за формулою:

$$Ц = \frac{С_в \cdot (1 + Р_{рен}) + К + В_{н.і.}}{К} \quad (4.9)$$

де Ррен. – рівень рентабельності;

К – кількість замовлень, од.;

Ві.н. – вартість носія інформації, грн.;

ПДВ – ставка податку на додану вартість, (20 %).

$$Ц = 44\,313{,}04 \cdot (1 + 0{,}3) \cdot (1 + 0{,}2) = 69\,128{,}34\ (грн)$$

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу.

Для визначення ефективності продукту розраховують чисту теперішню вартість (ЧТВ), а також термін окупності (), який можна визначити за формулою 4.10.

$$ЧТВ = -K_B + \sum_{i=1}^{t} \frac{ГП}{(1+i)^t}, \quad (4.10)$$

де КВ – затрати на проект;

Гп – грошовий потік за t – ий рік;

t – відповідний рік проекту;

i - величина дисконтної ставки (10…15%).

Якщо ЧТВ ≥ 0, то проект має потенціал і може бути рекомендований до впровадження.

$$ЧТВ = -44\,313{,}04 + \frac{24815{,}3}{(1+0{,}1)} + \frac{24815{,}3}{(1+0{,}1)^2} + \frac{24815{,}3}{(1+0{,}1)^3} = 17\,399{,}00$$

Термін окупності визначається за формулою 4.11:

$$T_{ОК} = T_{ПВ} + \frac{H_В}{Г_{ПР}} \qquad (4.11)$$

де ТПВ – період до повного відшкодування витрат, років;

НВ – невідшкодовані витрати на початок року, грн.;

ГПР – грошовий потік на початок року, грн.

$$T_{ОК} = 2 + \frac{1245{,}14}{24815{,}3} = 2{,}1$$

Всі дані розрахунків внесемо в зведену таблицю 4.4 техніко-економічних показників.

Таблиця 4.4 — Техніко – економічні показники НДР

№п/п	Показник	Значення
1.	Собівартість, грн.	44 313,04
2.	Плановий прибуток, грн.	30 305,53
3.	Ціна НДР, грн.	69 128,34
4.	Чиста теперішня вартість	17 399
5.	Термін окупності, рік	2,1

Враховуючи основі економічні показники, зведені у таблицю 4.4, можна зробити висновок, що при терміні окупності – 2,1 років проводити роботи по розробці даного програмного продукту є доцільним та економічно вигідним. Як можна побачити із розрахунків, основними є витрати на оплату праці.

5 ОХОРОНА ПРАЦІ, ТЕХНІКИ БЕЗПЕКИ ТА ЕКОЛОГІЧНІ

5.1 Комплекс заходів, спрямованих на боротьбу із шумом

На даний момент у сучасному світі в умовах науково-технічного прогресу шум став одним із суттєвих несприятливих чинників, що впливають на людину.

Шум — це будь-який небажаний звук, якій наносить шкоду здоров'ю людини, знижує його працездатність, а також може сприяти отриманню травми в наслідок зниження сприйняття попереджувальних сигналів. З фізичної точки зору — це хвильові коливання пружного середовища, що поширюються з певної швидкістю в газоподібній, рідкій або твердій фазі.

Шум буває такого походження:

- механічного походження, який виникає внаслідок вібрації при роботі механізмів та устаткування, а також поодиноких чи періодичних ударів у з'єднаннях деталей та конструкцій;
- аеродинамічного походження, даний вид шуму виникає при подачі газу чи повітря по трубопроводах, вентиляційних системах, або їх стравлюванні в атмосферу;
- гідродинамічного походження, який виникає внаслідок процесів, що відбуваються у різних рідинах (гідравлічні удари, кавітація, турбулентність потоку);
- електромагнітного походження, цей вид шуму виникає внаслідок коливання елементів електромеханічних пристроїв під впливом змінних магнітних полів.

Кожен з наведених видів шуму є проблемним чинником, так як основна технологія програмного забезпечення напряму пов'язана із записом звуку, тому це може шкідливо вплинути на розробку програмного продукту, бо може зменшитись продуктивність процесу.

Для успішної боротьби з шумом необхідно знати його фізичні характеристики, закономірності виникнення та поширення. Шумом прийнято вважати звуки, які негативно впливають на організм людини, заважають його роботі та відпочинку

Тому шум, часто називають несприятливим звуком. Зазвичай шум створюється при хаотичному чергуванні звуків різної частоти та інтенсивності. Звук як фізичне явище являє собою коливальний рух, що поширюється хвилеподібно у пружному середовищі (газоподібному, рідинному чи твердому).

Частота звуку визначається кількістю коливань пружного середовища за одиницю часу і вимірюється в герцах (1 Гц - це одне коливання за секунду). За частотою звукові (акустичні) коливання поділяються на три діапазони:

- інфразвукові з частотою коливання менше ніж 20 Гц;
- звукові (сприймаються органом слуху людини) від 20 до 20 000 Гц;
- ультразвукові - понад 20 000 Гц.

У свою ж чергу, звуковий діапазон прийнято поділяти на низькочастотний — до 400 Гц, середньочастотний — 400-1000 Гц, високочастотний — понад 1000 Гц.

Звук, що поширюється у повітряному середовищі, називається повітряним звуком, а в твердих тілах - структурним. Повітряний простір, в якому поширюються звукові хвилі, називається звуковим полем. У результаті коливань, що генеруються джерелом звуку, в повітрі виникає звуковий тиск, який накладається на атмосферний.

Різницю між атмосферним тиском і значенням явного тиску в даній точці звукового поля прийнято вважати звуковим типом (p). Поширення звукової хвилі супроводжується перенесенням звукової дії. Середній потік звукової енергії в будь-якій точці середовища за одинцю часу, віднесений до одиниці поверхні, перпендикулярної до напрямку поширення хвилі,

називається інтенсивністю, або силою звуку, в даній точці / і вимірюється у Вт/м2.

Отже, щоб шум припинив свій вплив на процес розробки, потрібно вжити певних заходів.

Заходи та засоби захисту від шуму поділяються на колективні та індивідуальні, причому останні застосовуються лише тоді, коли заходами та засобами колективного захисту не вдається знизити рівні шуму на робочих місцях до допустимих значень.

Звукоізоляція є ефективним засобом зменшення рівня шуму у напрямку його поширення, що реалізується шляхом встановлення звукоізоляційних перешкод (перегородок, кабін, кожухів, екранів). Принцип звукоізоляції базується на тому, що більша частина звукової енергії, яка потрапляє на перешкоду, відбивається і лише незначна її частина проходить крізь неї.

Рівень шуму у приміщенні залежить не лише від прямого, але й відбитого звуку. Тому, якщо у приміщенні неможливо знизити енергію прямого звуку, то необхідно зменшити енергію звукових хвиль, які відбиваються від внутрішніх поверхонь приміщення. Для цього проводять акустичне оброблення всіх або частини стін та стелі за допомогою звукопоглинального облицювання та (або) підвішують до стелі штучні звукопоглиначі.

Процес поглинання звуку відбувається при переході коливної енергії частинок повітря в теплоту внаслідок втрат на тертя в порах звукопоглинального матеріалу. Тому для ефективного звукопоглинання матеріал повинен мати пористу структуру, причому необхідно, щоб пори були відкриті з боку звукової хвилі і мали якнайбільше з'єднань між собою. Штучні звукопоглиначі найдоцільніше розміщувати в зонах, де концентруються звукові хвилі, що відбиваються від внутрішніх поверхонь приміщення.

Звукопоглиначі можуть мати різну форму (куля, куб, ромб, піраміда) і виготовляються з перфорованих листів твердого картону, пластмаси чи металу, які зі середини покриті звукопоглинальним матеріалом.

Глушники шуму — це ефективний засіб боротьби з шумом аеродинамічного походження, який виникає при роботі вентиляційних систем, та деяких інших джерел шуму. За принципом дії глушники поділяють на активного, реактивного та комбінованого типу. У глушників активного типу зниження шуму відбувається внаслідок його затухання в порах звукопоглинального матеріалу.

В глушниках реактивного типу шум знижується шляхом відбивання звукових хвиль у системі розширювальних та резонансних камер, що з'єднані між собою за допомогою труб, щілин та отворів. У комбінованих глушниках відбувається як поглинання, так і відбивання шуму.

5.2 Нервово-емоційна напруженість праці розробника ПЗ

Функціональне напруження організму під час праці можна охарактеризувати з двох сторін — енергетичного та інформаційного. Перший переважає при фізичній, другий при розумовій праці. Характеристику роботи, що потребує інтенсивної праці головного мозку при отриманні та аналізі інформації, фізіологи називають напруженістю. Напруженість праці — характеристика трудового процесу, що відображає навантаження переважно на центральну нервову систему, органи чуттів, емоційну сферу працівника.

Напруженість трудового процесу оцінюють відповідно до Державних санітарних норм та правил «Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу», затверджених наказом МОЗ України від 08.04.2014 р. № 248.

Основними показниками напруженості праці є: тривалість зосередження уваги або щільність сигналів, ступінь ризику для власного життя та життя інших осіб або ступінь відповідальності за життя інших осіб, змінність при роботі виключно в нічну зміну. Допоміжні показники враховують лише в разі перевищення нормативних значень.

До показників, що характеризують напруженість праці, належать:

- інтелектуальні, сенсорні, емоційні навантаження;
- ступінь монотонності навантажень;
- режим роботи.

Інтелектуальні навантаження — використовуються виключно для оцінки професій розумової праці. Тривале розумове навантаження впливає на психічну діяльність, може погіршувати функції уваги, пам’яті, сприйняття (збільшується частота помилок).

Під час значної розумової напруженості може виникати тахікардія (частішання пульсу), підвищення кров’яного тиску, зміни в електричній активності серцевого м’язу та мозку, збільшення легеневої вентиляції і споживання кисню.

Такі функціональні зміни в організмі під час довготривалої дії можуть спричинити розвиток гальмівних процесів у центральній нервовій системі, послаблення пильності й уваги, розвиток втоми.

Ергонометричні показники напруженості праці:

- Кількість об’єктів одночасного спостереження.
- Тривалість зосередженого спостереження чи часу активних дій (у % від загального часу робочого дня).
- Щільність сигналів (оголошень) за 1 год.
- Емоційна напруженість.
- Змінність.
- Напруженість функцій аналізаторів.
- Обсяг оперативної пам’яті.
- Інтелектуальна напруженість.
- Монотонність та інші.

Розумові завдання, які не вимагають прийняття рішення, вважаються найлегшими. Такі умови праці вважаються оптимальними. Такі умови роботи вважаються допустимими, якщо оператор працює і приймає рішення в рамках наказу.

Стрес-токсичні умови включають роботу, яка передбачає використання відомих алгоритмів для вирішення складних завдань або використання кількох інструкцій. До важкої роботи слід віднести творчу діяльність, яка потребує вирішення складних завдань без очевидного алгоритму вирішення.

Обробка будь-якої інформації або виконання завдання без оцінки його результатів не є надто складним завданням, що дозволяє оцінити його як оптимальне. Такі умови роботи допускаються, якщо необхідно перевірити отримані результати та додати їх до зазначеної операції. Розподіл виробничих завдань між іншими людьми та контроль за їхньою роботою — важка праця.

Напруженість праці залежить від тривалості зосередженого спостереження і числа одночасно спостережуваних об’єктів (контрольно-вимірювальні прилади, продукт виробництва і тому подібне). При тривалості зосередженого спостереження до **50%** від тривалості робочої зміни умови праці характеризуються як оптимальні, **51–75%** — допустимі, більше **75%** — напружена праця 1-го ступеня.

Істотний вплив на міру напруженого стану виконавця має відповідальність за кінцевий або проміжний результат праці. Якщо оператор несе відповідальність за виконання тільки окремих елементів виробничого завдання, то напруженість такої праці оцінюється як оптимальна.

Підвищення міри відповідальності, наприклад, за функціональну якість допоміжних операцій спричиняє додаткові емоційні зусилля з боку безпосереднього керівника (бригадира, майстра та ін.). У цих випадках напруженість праці оцінюється як допустима.

Якщо на виконавцеві лежить відповідальність за функціональну якість основної роботи, що може мати необхідність ухвалення рішень, пов’язаних з

виправленням результатів за рахунок додаткових зусиль усього колективу, то такий вид діяльності є напруженим 1-го ступеня. (клас 3.1).

Якщо ж працівник несе персональну відповідальність за функціональну якість кінцевого продукту, виробничого завдання в цілому або його дії можуть привести до поломки устаткування, зупинки усього технологічного процесу або створити ситуацію, небезпечну для життя, його умови праці оцінюються як напружені 2-го ступеня (клас 3).

За відсутності ризику для власного життя в процесі виконання своїх обов'язків працю виконавця вважають оптимальною, якщо ж ризик вірогідний, то умови праці відносять до класу 3 — напружена праця 2-го ступеня. Аналогічно встановлюється клас умов праці при оцінці міри ризику за безпеку інших осіб, що беруть участь у виробничому процесі.

Одноманітність виконуваних операцій призводить до певного стану людини, що називається монотомією. Ознакою монотомії є або перевантаження однаковою інформацією, або недолік нової. Це накладає відбиток на функціональний стан людини: вона втрачає інтерес до виконуваної роботи.

Для неї робочий час ніби зупинився, і тоді вона з нетерпінням чекає закінчення зміни, її хилить до сну. Монотонна робота знижує ефективність праці, збільшує плинність кадрів, аварійність і, як наслідок, травматизм на виробництві.

Тривала робота в умовах постійної нервово-емоційної напруги може призвести до серцево-судинних захворювань. Будь-яка дія, що перевищує допустимі межі, викликає порушення діяльності аналізаторів і навіть призводить до больових відчуттів. Завдання розробників технологічних процесів — не допустити перенапруження вищої нервової діяльності, інакше може настати стрес.

Поява стресу в аварійній обстановці стає причиною неправильних дій оператора, що часто посилює виробничу ситуацію. Ефективним засобом профілактики стресів за екстремальних умов є професійна підготовка на

тренажах, що імітують аварійні ситуації. Доцільно розробити та впровадити комплексні цільові програми з профілактики та зниження рівня стресу в організації.

Вони мають включати заходи на рівні компанії в цілому та індивідуальну роботу з кожним працівником:

- оптимізацію виробничих та управлінських процесів;
- поліпшення організації праці, умов роботи;
- модернізацію обладнання;
- тренінги з антистресових технік і тайм-менеджменту;
- регулярні консультації психологів;
- за потреби – медикаментозну підтримку.

Формування комфортного робочого простору – це зручне розміщення робочих місць, ергономічне обладнання, оптимальне освітлення та температурний режим, зони відпочинку та харчування для персоналу.

Також, це облаштування спеціальних кімнат психологічного розвантаження, де працівники можуть відновити сили й емоційну рівновагу протягом робочого дня. Створення сприятливого робочого середовища також суттєво допомагає зниженню стресу в колективі. Позитивно впливають на психоемоційне самопочуття робітників:

- комфортні умови праці;
- ергономічно організований простір;
- можливості для відпочинку та відновлення сил.

Доцільно офіційно встановити обов’язкові регулярні перерви протягом робочого дня для відпочинку та відновлення працівників. Наприклад, фіксовані 5-10 хвилинні паузи через кожні 1-2 години роботи або 15-20 хвилинні перерви через кожні 3-4 години.

Під час таких пауз рекомендується відволікатися від завдань, змінити вид діяльності, зробити комплекс вправ, перекусити тощо. Це значно

підвищує продуктивність праці, оскільки дає змогу вчасно знизити накопичувану втому та монотонність, запобігти перевтомі та вигоранню.

5.3 Організація безпечної поведінки працівника в процесі праці

Одним із напрямів підвищення безпеки праці є організація безпечної поведінки працівника в процесі праці.

Для цього необхідно:

- створювати психологічний настрій на безпечну поведінку;
- стимулювати безпечну поведінку;
- навчати безпечній діяльності;
- виконувати та контролювати правила безпеки праці;
- виховувати безпечну поведінку;
- створювати комфортний психологічний клімат у колективі. Для створення психологічного настрою працівника на безпеку праці необхідна загальна політика керівництва у галузі охорони праці.

Ставлення керівництва і, особливо, керівника організації, установи, підприємства до питань охорони праці проявляється в тому, яке значення надають вони цим питанням в загальному процесі праці та в якій мірі показник безпеки враховується при оцінці її ефективності.

Працівник вірить у небезпеку тільки в тій мірі, в якій вірить у неї його безпосередній керівник. Тому всі ланки управління виробництвом повинні постійно проявляти інтерес до забезпечення безпеки праці, проявляти підвищену увагу та турботу щодо безпеки та благополуччя працівників. Причому працівники повинні це постійно відчувати.

Настрій на безпечну роботу з'явиться у працівника тоді, коли він буде бачити, що на підприємстві, в організації та установі існує суворий контроль за виконанням правил безпеки.

На жаль, сьогодні у невиробничій сфері й недержавному секторі економіки стан охорони праці викликає тривогу. Більшість керівників у цих сферах не мають спеціальної підготовки і досвіду роботи з охорони праці. Вони проявляють байдужість до проблем охорони праці і небажання серйозно займатися їхнім вирішенням.

Першочерговим завданням охорони праці у невиробничій сфері є формування у роботодавців думки про те, що охороні праці необхідно приділяти пріоритетну увагу. Тоді працівник повірить, що безпека його праці є однією з ключових цінностей підприємства, а це є одним із мотивів його безпечної поведінки.

Зазвичай, для виховання безпечної поведінки в процесі праці використовується як негативне стимулювання - покарання за порушення правил безпеки (штрафи, позбавлення премії, дисциплінарне покарання), так і позитивне - заохочування за безпечну роботу (грошові надбавки до заробітної плати, моральне стимулювання).

В Україні найчастіше застосовується негативне стимулювання за порушення правил безпеки праці. Найбільш типовою причиною навмисних порушень правил техніки безпеки є прагнення за рахунок цього досягти будь-яких вигод (полегшення, прискорення, спрощення роботи). Незважаючи на покарання, за таких умов працівник все одно буде продовжувати ці порушення, доки вони не перестануть бути джерелом вигод.

У таких випадках при застосуванні негативного стимулювання слід зробити так, щоб витрати від порушення правил безпеки перевищували отримані за рахунок цього вигоди. Тоді їх буде невигідно порушувати.

Покаранн за ненавмисне порушення правил безпеки праці, як свідчать психологічні дослідження, мають незначну ефективність. Такі покарання корисно застосовувати у процесі навчання, при формуванні навичок до безпечного поводження в процесі праці. У таких випадках покарання, по-перше, будуть перешкоджати закріпленню недоцільних і небезпечних навичок

в роботі і, по-друге, будуть сприяти створенню мотивів до обережного типу поведінки.

Найбільш доцільно та ефективно використовувати позитивне стимулювання. Застосування заохочень за безпечну роботу, як свідчить міжнародний досвід, є найбільш дійовим засобом підвищення безпеки праці.

Стимулювання охорони праці має бути індивідуальним. Для організації стимулювання повинні бути розроблені критерії оцінки рівня безпеки праці кожного робітника (бали, коефіцієнти тощо). Треба періодично підводити підсумки безпечної роботи. Показники, умови, форми та розміри стимулювання охорони праці конкретизуються в колективних договорах, положеннях про оплату праці, трудових договорах (контрактах) з урахуванням особливості організації праці на підприємстві, в установі, організації.

Практичне застосування системи стимулювання безпечної праці показує, що вона сприяє суттєвому зменшенню нещасних випадків, підвищує продуктивність праці, а отриманий прибуток значно перевищує витрати, пов'язані з таким стимулюванням.

Значне місце у підвищенні безпеки праці, в організації безпечної поведінки займає процес навчання працівників з питань охорони праці.

Недостатній рівень знань працівника виявляє його некомпетентність з питань охорони праці. В процесі праці він не може точно визначити, що є небезпечним, а що безпечним; де наслідки помилки малі, а де великі. Крім того, він не може швидко орієнтуватися і знаходити рішення в складних і небезпечних ситуаціях. Такий працівник розуміє, що він може легко допустити небезпечну помилку, усвідомлює, що у нього малі можливості протидіяти небезпеці.

Все це породжує тривогу, невпевненість у собі, у безпеці своєї праці і призводить до його небезпечних дій. Перераховані фактори визначаються як прояв недосвідченості.

ВИСНОВКИ

У курсовій роботі було розроблено програмне забезпечення «Голосовий асистент». Протягом виконання роботи було сформовано технічне завдання, в якому були визначені основні напрямки і спеціальні вимоги, необхідні для правильного та ефективного функціоналу програмного продукту.

На етапі розробки технічного проекту була побудована структурна схема програмного продукту та описана взаємодія її функціональних елементів. Програма була написана в середовищі програмування **Visual Studio Code**, що набагато спрощує розробку програми, завдяки легкому синтаксису.

Програмний виріб є доцільним, оскільки він буде використовуватися у звичайних або рутинних задачах, як системного адміністратора так і звичайного користувача з базовими знаннями про персональний комп’ютер.

В програмі використано такі методи: **main, main_program, command, filter_cmd, tts, read_file, respond, add_cmd, del_cmd, cmd_clear, hide_widget, come_widget**. Під час наступного етапу реалізації було розроблено код програмних компонентів, а також здійснено аналіз можливостей.

На останньому етапі складено алгоритм тестування та проведено ряд тестів для перевірки правильності функціонування програмного продукту. Перевагами даного програмного продукту, є те що він полегшує звичайні чи рутинні задачі і його може використовувати як досвідчений розробник, так і звичайний користувач.

ПЕРЕЛІК ПОСИЛАНЬ

1) Підручник з Python — Python 3.12.1 documentation. URL: https://docs.python.org/uk/3/tutorial/index.html.

2) Презентація "Мова програмування Python. Вступ. Основні можливості мови. URL: https://vseosvita.ua/library/prezentacia-mova-programuvanna-python-vstup-osnovni-mozlivosti-movi-258160.html.

3) Презентація "Мова Python" - «На Урок». URL: https://naurok.com.ua/prezentaciya-mova-python-271561.html.

4) Презентація "Мова програмування Python. Середовище програмування". URL: https://naurok.com.ua/prezentaciya-mova-programuvanya-python-seredovische-programuvannya-325102.html.

5) Бібліотека для розпізнавання мовлення. SpeechRecognition 3.10.4. URL: https://pypi.org/project/SpeechRecognition

6) Мова програмування Python для початківців. URL: https://uk.wikipedia.org/wiki/Python.undefined.

7) Інсталяція PYPI для мови програмування Python. URL: https://packaging.python.org/en/latest/tutorials/installing-packages

8) Організація безпечної поведінки працівника в процесі праці. URL: https://pidru4niki.com/1657072238170/bzhd/organizatsiya_bezpechnoyi_povedinki_pratsivnika_protsesi_pratsi

9) Зниження рівня стресу працівників на підприємстві. URL: https://hrliga.com/index.php?module=news&op=view&id=16919

ДОДАТКИ

Додаток А - Код програми

```
print("Loading...")
from speech_recognition import RequestError, Recognizer, Microphone, UnknownValueError
from time import sleep
from pygame import mixer
import webbrowser
import os
import sys
from random import randint, choice
import threading
mixer.init()
sys.path.append(f'{sys.path[0]}\\sounds')

import data_base

def tts(text: str):
    rand_int, time_int = 1, 2
    lst_sound = data_base.AS_SOUNDS_SYS[text]
    rand_int = randint(1,lst_sound[0])
    time_int = lst_sound[1]
    mixer.Sound(f"{sys.path[-1]}/{text} {rand_int}.mp3").play()
    sleep(time_int)

def read_file():
    try:
        file = open("open_close_cmd.txt", encoding="utf-8")
        contain_file = file.readlines()
```

```
            file.close()
            return contain_file
        except FileNotFoundError:
            file = open("open_close_cmd.txt", encoding="utf-8",mode='x')
            file.write('command#path')
            file.close()
            return ['command#path']



def filter_cmd(text: str):
    # reads the file and divides it into a word and a path
    contain_file = read_file()
    AS_KEY_WORDS, AS_KEY_PATH, lst_sort = [],[],[]
    for i in contain_file:
        lst_sort.append(i.split("#"))
    for i in lst_sort:
        AS_KEY_WORDS.append(i[0])
        AS_KEY_PATH.append(i[1])
    lst_sort = []
    # find type do
    for lst in data_base.AS_TYPE_DO:
        for k in lst:
            ind = text.find(k)
            if ind >= 0:
                id_list = data_base.AS_TYPE_DO.index(lst)
                lst_sort.append(id_list)
                # find name new txt file
                if id_list == 2:
                    text = text[ind:].replace('мені','')
```

```
            text = text.replace('будь ласка','')
            text = text.replace('запиши','').strip().capitalize()
            lst_sort.append(text)
            break
        # find key word
        elif id_list == 3:
            for i in data_base.AS_CLEAR:
                ind = text.find(i)
                if ind >= 0:
                    lst_sort.append('чат')
            break
        elif id_list == 4:
            for i in data_base.AS_COMMANDS:
                ind = text.find(i)
                if ind >= 0:
                    lst_sort.append('команди')
            break
        elif id_list == 5:
            ind = text.find('пароль')
            if ind>=0:
                ind = text.find(' на ')
                count = text[ind+4:text.rfind(' ')]
                lst_sort.append(count)
        else:
            for i in data_base.AS_DEF_WORDS:
                ind = text.find(i)
                if ind >=0 :
                    lst_sort.append(i)
                    break
            for i in AS_KEY_WORDS:
```

```
                    ind = text.find(i)
                    if ind >= 0:
                        path = AS_KEY_PATH[AS_KEY_WORDS.index(i)]
                        if path[-1] == '\n':
                            path = path[:-1]
                        lst_sort.append(path)
                        break
        return lst_sort

    def respond(text: str, resp: bool):
        if text in data_base.AS_ALIAS:
            tts("listening")
            resp = True
        else:
            resp = False
        return resp

    def command(text: str, resp: bool):
        global loop_listen, test_or_no, name_file
        try:
            txt2['text'] = 'Активність :   Виконує команду...'
        except:
            pass
        if text in ['бувай','папа','щасливо']:
            tts('bye')
            try:
                txt2['text'] = 'Активність :   Відпочиває :)'
                field.insert(END,"\nЩоб покликати Емілію, перезапустіть будь ласка програму!")
            except:
```

```
            pass
        print("Bye!")
        quit()
    cmd = filter_cmd(text)
    print(cmd)
    resp = False
    if text == 'скасуйте сеанс':
        os.system('shutdown /a')
        tts('process_completed')
    elif len(cmd)<2:
        tts("not_found_command")
    elif cmd[1] in ["комп'ютер","користувач"]:
        if cmd[0]==1 and cmd[1]=="комп'ютер":
            os.system('shutdown /s /f /t 30')
        elif cmd[0]==1 and cmd[1]=="користувач":
            os.system('shutdown /l')
        elif cmd[0]==6 and cmd[1]=="комп'ютер":
            os.system('shutdown /r /f /t 15')
    else:
        if cmd[0]==0:
            print(cmd[1])
            txt = 'open'
            if cmd[1]=='тест мікрофона':
                loop_listen = True
                test_or_no = True
                resp = True
                txt = 'test_micro_started'
            elif cmd[1]=='браузер':
                webbrowser.open('google.com')
            elif cmd[1]=='командний рядок':
```

```
            os.system('start')
        elif cmd[1]=='диспетчер завдань':
            os.system('start taskmgr')
        else:
            try:
                os.startfile(cmd[1])
            except:
                pass
        tts(txt)
    elif cmd[0]==1:
        path = cmd[1]
        print(path)
        if cmd[1]=='браузер':
            path = 'msedge.exe'
        else:
            if '/' in path:
                i = path.rfind('/')+1
                path = path[i:]
                print(path)
            if os.system(f'taskkill /f /t /im "{path}"') == 128:
                tts('not_found_process')
            else:
                tts('process_completed')
    elif cmd[0]==2:
        name_file = cmd[1]
        loop_listen = True
        resp = True
        try:
            txt2['text'] = 'Активність :   Записує ...    '
        except:
```

```
                pass
            tts('start_to_write')
        elif cmd[0]==3:
            if cmd[1] in data_base.AS_CLEAR:
                try:
                    field.delete("1.0",END)
                except:
                    pass
        elif cmd[0]==4:
            if cmd[1] in data_base.AS_COMMANDS:
                try:
                    field.insert(END,'Список команд: \n1.Відкрий/закрий')
                    field.insert(END,'\n\nСистемні команди:\n')
                    field.insert(END,f'\nвідкрий/закрий    браузер\nвідкрий/закрий
командний рядок\nвідкрий/закрий диспетчер завдань')
                    field.insert(END,'\n\nКоманди користувача:')
                    cmd_list = read_file()
                    print(cmd_list)
                    if len(cmd_list) == 1:
                        field.insert(END,'\nПусто')
                    else:
                        AS_KEY_WORDS,lst_sort = [],[]
                        for i in cmd_list:
                            lst_sort.append(i.split("#"))
                        for i in lst_sort:
                            AS_KEY_WORDS.append(i[0])
                        lst_sort = []
                        for i in AS_KEY_WORDS[1:]:
                            field.insert(END,f"\nвідкрий/закрий {i}")
                    field.insert(END,'\n\n2. Запиши/занотуй (назва тескту)')
```

```
                field.insert(END,'\n3. Почисти/очисти (чат/консоль)')
                field.insert(END,'\n4. Покажи/виведи (список команд/що ти вмієш)')
                field.insert(END,'\n5. Запусти/включи (тест мікрофона)')
                field.insert(END,'\n6. Придумай пароль на (n) символів')
                field.insert(END,'\n7. Вимкни/перезапусти комп\'ютер')
                field.insert(END,'\n8. Вийди з користувача')
                field.insert(END,'\n9. Скасуй сеанс (скасовує заплановане відключення ПК)')
                field.insert(END,'\n10. Бувай/папа (вимкнення асистента)')
            except:
                pass
        elif cmd[0]==5:
            try:
                count = int(cmd[1])
                if count <=4 or count>255:
                    tts('few_sybvol')
                else:
                    sybvol = 'QWERTYUIOPASDFGHJKLZXCVBNMqwertyuiopasdfghjklzxcvbnm1234567890$@#'
                    password = ''.join(choice(sybvol) for i in range(count))
                    field.insert(END,f'\nЗгенерований пароль на {count} символів: {password}')
            except:
                tts('invalid_type')

        print(text)
        print(resp)
        return resp
```

```
cmd_activated, loop_listen, test_or_no = False, False, False
name_file = ''
recognizer = Recognizer()

from tkinter import *
from PIL import Image, ImageTk
from tkinter.messagebox import showerror, showinfo, askyesno

window = Tk()
window.geometry("450x700+1450+150")
window.title("Emilia")
window.resizable(False,False)

photo = PhotoImage(file='microphone.png')
window.iconphoto(False, photo)

image = Image.open("full_bg.png")
resize_image = image.resize((450, 700))
img = ImageTk.PhotoImage(resize_image)
label1 = Label(image=img)
label1.image = img

txt1 = Label(window, text='Чат спілкування з Емілією',
relief='sunken',bg="#4B4B4B",foreground='white',
        font=('Arial',9,'bold'),padx=10)
txt2 = Label(window, text='Активність : Відпочиває :)',
relief='sunken',bg="#4B4B4B",foreground='white',
        font=('Arial',9,'bold'),padx=10)
def main_program():
```

```
print("Start")
recognizer = Recognizer()
global cmd_activated, loop_listen, test_or_no, name_file, field

while True:
    try:
        txt2['text'] = 'Активність :   Відпочиває :)'
    except:
        pass
    try:
        with Microphone() as mic:
            print(cmd_activated, loop_listen, test_or_no, name_file)
            recognizer.adjust_for_ambient_noise(mic, duration=0.5)
            if cmd_activated:
                try:
                    txt2['text'] = 'Активність :   Слухає . . .    '
                except:
                    pass
            print('listening...')
            audio = recognizer.listen(mic)
            print('Recognize...')
            rec_result = recognizer.recognize_google(audio, language="uk-
UA")
            print(rec_result)
            if cmd_activated:
                if loop_listen:
                    if test_or_no:
                        if rec_result == 'кінець':
                            loop_listen = False
                            cmd_activated = False
```

```
                    test_or_no = False
                    print('record micro to finished!')
                    tts('test_micro_finished')
                else:
                    try:
                        field.insert(END,f"{rec_result} ")
                    except:
                        pass
            else:
                try:
                    file = open(f'{name_file}.txt',encoding="utf-8",mode='x')
                except FileExistsError:
                    file = open(f'{name_file}.txt',encoding="utf-8",mode='a')
                finally:
                    if rec_result == 'кінець':
                        loop_listen = False
                        cmd_activated = False
                        file.write('\n')
                        print('notate txt to finished!')
                        tts('finish_to_write')
                    else:
                        if name_file.startswith('Рецепт') or
name_file.startswith('Слова пісні'):
                            print(file.write(f'{rec_result.capitalize()}\n'))
                        else:
                            print(file.write(f'{rec_result.capitalize()}. '))
                    file.close()
                print('finish to write')
            continue
        else:
```

```
                        cmd_activated = command(rec_result.lower(), cmd_activated)
                        continue
                    cmd_activated = respond(rec_result.lower(), cmd_activated)
                except UnknownValueError:
                    recognizer = Recognizer()
                    print('шум ...')
                    continue
                except RequestError as e:
                    print("Could not request results from Google Speech Recognition service; {0}".format(e))

        a = threading.Thread(target=main_program, name='background listening')
        a.start()

        def hide_widget(wid1,wid2,wid3):
            wid1.place_forget()
            wid2.place_forget()
            wid3.place_forget()

        def come_widget(wid1,wid2,wid3):
            wid1.place(x=12,y=350)
            wid2.place(x=142,y=331)
            wid3.place(x=429,y=351, height=325)

        def add_cmd(in_word,in_path):
            if in_word == "or in_path == '':
                showerror(title='Помилка введення даних',message='Будь ласка введіть ключове слово та шлях до файлу!')
            else:
                try:
```

```
                file = open("open_close_cmd.txt", encoding="utf-8")
                file.close()
                file = open("open_close_cmd.txt", encoding="utf-8",mode='a')
            except FileNotFoundError:
                file = open("open_close_cmd.txt", encoding="utf-8",mode='x')
                file.write('command#path\n')
            finally:
                file.write(f"{in_word}#{in_path}\n")
                file.close()
                showinfo(title='Інформація про файл',message='Команду успішно додано у файл!')
                key_path.delete(0,END)
                key_word.delete(0,END)

        def del_cmd(in_word,in_path):
            if in_word == '' and in_path == '':
                showerror(title='Помилка введення даних',message='Будь ласка введіть ключове слово або шлях до файлу!')
            else:
                try:
                    file = open("open_close_cmd.txt", encoding="utf-8",mode='r')
                    contain_file = file.readlines()
                    valu = len(contain_file)
                    print(contain_file)
                    file.close()
                    file = open("open_close_cmd.txt", encoding="utf-8",mode='w')
                    AS_KEY_WORDS, AS_KEY_PATH, lst_sort = [],[],[]
                    for i in contain_file:
                        lst_sort.append(i.split("#"))
                    for i in lst_sort:
```

```
                AS_KEY_WORDS.append(i[0])
                AS_KEY_PATH.append(i[1])
            print(AS_KEY_PATH)
            print(AS_KEY_WORDS)
            count = 0
            while count < len(contain_file):
                if in_word == AS_KEY_WORDS[count] or in_path == AS_KEY_PATH[count][:-1]:
                    print('here')
                    del contain_file[count]
                    showinfo(title='Інформація про файл',message='Команду успішно видалено з файлу!')
                    break
                count +=1
            for i in contain_file:
                print('write')
                file.write(i)
            if len(contain_file) == valu:
                showerror(title='Помилка зчитування файлу',message='Введена команда відсутня у файлі!')
        except FileNotFoundError:
            file = open("open_close_cmd.txt", encoding="utf-8",mode='w')
            file.write('command#path\n')
            showerror(title='Інформація про файл',message='Список з командами пустий!')
        finally:
            file.close()
            key_path.delete(0,END)
            key_word.delete(0,END)
```

```
def cmd_clear():
    yes = askyesno(title='Підтвердження операції',message='Ви дійсно хочете очистити файл?',
                   detail='Якщо ви підтвердите дію, то всі команди записані раніше, будуть видалені назавжди!')
    if yes:
        file = open("open_close_cmd.txt", encoding="utf-8",mode='w')
        file.write('command#path\n')
        file.close()


but1 = Button(window,text="Головна", padx=95,borderwidth=1,bg="#4B4B4B",foreground='white',
              font=('Colibri',9,'bold'),command=lambda: come_widget(field,txt1,scrollbar))
but2 = Button(window,text="Меню команд", padx=90,borderwidth=1,bg="#4B4B4B",foreground='white',
              font=('Colibri',9,'bold'),command=lambda: hide_widget(field,txt1,scrollbar))

but_add = Button(window,text="Додати команду", padx=46,pady=5,borderwidth=1,bg="#4B4B4B",foreground='white',
              font=('Colibri',9,'bold'),command=lambda: add_cmd(key_word.get(),key_path.get()))
but_del = Button(window,text="Видалити команду", padx=42,pady=5,borderwidth=1,bg="#4B4B4B",foreground='white',
              font=('Colibri',9,'bold'),command=lambda: del_cmd(key_word.get(),key_path.get()))
but_clear = Button(window,text="Очистити всі команди", padx=130,pady=5,borderwidth=1,bg="#4B4B4B",foreground='white',
```

```
                font=('Colibri',9,'bold'),command=lambda: cmd_clear())

    text_key_word = Label(window, text='Ключове слово :',bg="#4B4B4B",foreground='white',
                font=('Arial',11),border=1,padx=4,relief='groove')

    text_key_path = Label(window, text='Шлях до файлу :',bg="#4B4B4B",foreground='white',
                font=('Arial',11),border=1,padx=4,relief='groove')

    key_path = Entry(relief='sunken',border=1, font=("Arial",14),bg="#4B4B4B",width=30,foreground='white')
    key_word = Entry(relief='sunken',border=1, font=("Arial",14),bg="#4B4B4B",width=20,foreground='white')

    field = Text(padx=10, width=49, height=20, bd=2, bg="#4B4B4B", foreground='white',
                font=("Verdana", 10), relief='solid', wrap="word",)

    scrollbar = Scrollbar(window,orient="vertical", command=field.yview, relief='solid')
    field["yscrollcommand"]=scrollbar.set

    scrollbar.place(x=429,y=351, height=325)
    label1.place(x=0,y=0)

    but1.place(x=0,y=300)
    but2.place(x=225,y=300)

    field.place(x=12,y=350)
```

```
txt1.place(x=142,y=331)
txt2.place(x=143, y=265)

text_key_word.place(x=60,y=391)
text_key_path.place(x=60,y=491)

key_word.place(x=60,y=410)
key_path.place(x=60,y=510)

but_add.place(x=35,y=590)
but_del.place(x=229,y=590)
but_clear.place(x=35,y=620)

window.mainloop()
```