Tampere University of Applied Sciences

Personal VoiceAI: Human-Like Voice Assistant

Jannatun Noor

BACHELOR'S THESIS
June 2024

Degree Programme in Software Engineering

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Software Engineering

NOOR, JANNATUN
Personal VoiceAI: Human-Like Voice Assistant

Bachelor's thesis 34 pages, appendices 2 pages
June 2024

Voice assistant technology represents a significant advancement in human-computer interaction, offering users a convenient and intuitive way of interacting with digital systems. This thesis aimed to provide voice capabilities across different devices using web development tools. The study includes a detailed analysis of the architecture, speech recognition algorithms, and user experience related to voice assistant web applications. Taking a multidisciplinary approach, insights from natural language processing, human-computer interaction, and artificial intelligence were combined. The research for this thesis involves thoroughly examining academic sources, industry standards, and conducting practical experiments to uncover the challenges and possibilities in creating voice assistant applications.

The objectives of the thesis include providing a detailed explanation of voice assistant systems, conducting a critical review of existing and potential issues, and outlining the evolution of voice assistant technology. This research explores voice assistants' technical aspects, user experiences, and social impacts, emphasizing their potential to transform interactions with digital platforms. The project showcases practical application and innovation, demonstrating effective integration and deployment of voice assistant functionalities.

CONTENTS

ABBREVIATIONS AND TERMS

TAMK	Tampere University of Applied Sciences
cr	credit
API	Application Programming Interface is a set of rules and protocols for building and interacting with software applications. APIs allow different software systems to communicate with each other.
CORS	Cross-Origin Resource Sharing is a security feature implemented in web browsers to allow or restrict resources on a web page to be requested from another domain outside the domain from which the resource originated.
NLP	Natural Language Processing is a field of artificial intelligence that focuses on the interaction between computers and humans through natural language.
PyPI	Python Package Index is a repository of software for the Python programming language that helps users discover and install software created and shared by the Python community.
SR	Speech Recognition can identify a program to process spoken language.
TTS	Text-to-Speech is a technology that reads digital text aloud.
UI	User Interface is the interface where human and machine interactions occur.
VoiceAI	Voice Artificial Intelligence is the technology to process and generate human speech.

1 INTRODUCTION

Voice assistants have become increasingly popular in recent years as a convenient and efficient way for users to interact with digital devices using natural language commands. This thesis is inspired by the success of voice assistant technologies such as Siri, Alexa, and Google Assistant, and aims to develop a voice assistant web application to meet the needs of modern users. The project addresses the growing demand for hands-free computing solutions, especially in scenarios where traditional input methods may be inconvenient or impractical.

Voice assistant technology has caught the attention of researchers as well as developers and end-users. Voice assistants allow consumers to interact with machines more naturally and conveniently, with potentially wide implications for manufacturers and developers. This has led to an emphasis on researching the basics, applications, and implications of such technology. One important area of exploration involves developing voice assistant web applications. These applications use voice control and application-based technologies via the internet to deliver a voice interface to a vast range of thousands of devices and platforms.

The motivation behind this project comes from a desire to use voice recognition technology to improve user experience and accessibility in computing. As Wang et al. (2019) noted, voice assistants have the potential to transform human-computer interaction by enabling more natural and intuitive communication between users and machines. By allowing users to speak commands instead of typing or clicking, voice assistants provide a more streamlined and efficient way to interact with digital systems. (Wang , 2019)

The voice assistant project discussed in this thesis is based on previous research and technological advancements in natural language processing and speech recognition. This project aims to address challenges related to speech recognition accuracy, language understanding, and user interface design to provide a seamless and user-friendly experience. This work draws on insights from studies by Li et al. (2020) and Garcia et al. (2018).

Beyond the obvious benefits of creating more user-friendly experiences, voice assistant web apps have the potential to increase the overall accessibility of apps to individuals with

disabilities. Typing or clicking can be challenging for people with motor or visual impairments. Voice control presents a credible alternative, meaning that this way of interfacing with the devices and applications can be beneficial and accessible to everybody who wants to utilize them.

This work is part of ongoing efforts to examine the impact of voice technology on human-computer interaction, now from the perspective of a web-based voice assistant application. The development of this application includes a robust, user-friendly system where the newest state-of-the-art speech recognition algorithms, natural language processing techniques, and user interface design principles are integrated. These findings have suggestions for voice technology developers, designers, and researchers to improve user interaction and accessibility in digital environments.

2 OVERVIEW OF VOICE ASSISTANT TECHNOLOGY

2.1 History and Evolution

The development of voice assistant technology has evolved significantly since its inception. The journey began in the 1960s with IBM's Shoebox, one of the earliest speech recognition systems. The Shoebox could recognize 16 spoken words and digits, marking the beginning of a technology that would eventually transform human-computer interaction. (Shoebox - IBM Archives 1961)

In the 1990s, Dragon Systems introduced Dragon Dictate, the first commercially available speech recognition product. This system allowed users to control their computers through voice commands, laying the groundwork for future advancements in voice technology (Nuance Communications).

The breakthrough came in 2011 with Apple's introduction of Siri, a virtual assistant integrated into the iPhone 4S. Siri could understand and process natural language, setting a new standard for voice assistants and paving the way for similar technologies. Following Apple's innovation, Google introduced Google Now (later evolved into Google Assistant), Amazon launched Alexa, and Microsoft unveiled Cortana (Apple; Google; Amazon Alexa; Microsoft).

Those voice assistants leveraged immense progressions in artificial intelligence (AI) and natural language processing (NLP) to make their voice recognition and comprehension functions seem so effortless. Influenced largely by machine learning (ML) algorithms, these systems were taught to learn from user interactions, and to become better in performance over time. Fast forward to the 2020s and voice-based technology is simply everywhere, from smartphones, smart speakers, even cars to a range of Internet of Things (IoT) devices that facilitate a new set of expectations on how users engage with technology (Statista).

2.2 Current Trends and Applications

One of the most significant trends is the integration of voice assistants with smart home

devices. Voice assistants like Amazon Alexa, Google Assistant, and Apple's Siri are now central hubs in smart home ecosystems, controlling lights, thermostats, security systems, and household appliances. This trend is driven by the increasing adoption of Internet of Things (IoT) devices, making homes more interconnected and automated. (Statista).

Voice assistants are being used in healthcare, to help patients to remember medications and deliver health information, as well as to provide hands-free access to patient records and other data by healthcare providers. This app is designed to improve patient care in hospitals and make their operations more efficient. To combat the spread of COVID-19 and better meet healthcare demands, hospitals nationwide are quickly adopting telehealth and other digital health tools to provide care at a distance. Systems using chatbots and voice assistants, which are examples of intelligent conversational agents (ICA) and virtual assistants, have been used to increase health service capacity to assess symptoms, provide health information, and lower the risk of exposure. (PubMed Central)

Drivers can use voice assistants to navigate, control in-car entertainment systems, and get information without taking their hands off the wheel. It also makes driving safer by reducing driver distraction (J.D. Power). Voice technology can provide better accessibility to disabled individuals, making it easier for them to use technology and get information. This makes the technology simply more accessible and consumer-oriented.

In terms of personal activity, with the help of voice assistants, users can organize schedules or reminders and send messages with infinite tasks. Users can multitask and access a wider variety of activities in a shorter time, enhancing productivity and convenience.

2.3 Key Technologies and Tools

The development of Xabiar involves a combination of frontend and backend technologies and tools.

React is a popular JavaScript library used for building user interfaces, particularly single-page applications and dynamic web interfaces. In this project, React was utilized to develop the client-side application. React is a component-based architecture and virtual DOM makes it efficient for creating dynamic and interactive user interfaces. It simplifies the

process of managing UI components and states, enhancing the overall development experience. (React documentation).

Flask is a lightweight web application framework written in Python. It's commonly used for developing web applications and APIs due to its simplicity and flexibility. In this project, Flask was chosen for developing the server-side API. Flask provides essential features for handling HTTP requests and responses, routing, and integrating with other Python libraries. Its minimalistic design allows for the rapid development of robust and scalable web applications with minimal overhead. (flask documentation).

The speech recognition library is used to convert spoken language into text. This technology is crucial for enabling the voice assistant to understand and process voice commands accurately. The library supports multiple speech recognition engines and APIs, providing flexibility in implementation. It also includes features for noise reduction and error handling, which are essential for accurate recognition in various environments. (Speech recognition documentation).

The pyttsx3 library is used to convert text into spoken words. This allows the voice assistant to respond verbally to user queries, providing a more natural and interactive user experience. Pyttsx3 is an offline, text-to-speech conversion library that supports multiple TTS engines, making it versatile and reliable. It also allows for customization of speech rate, volume, and voice, enhancing the overall user experience. (pyttsx3 documentation)

The python libraries pywhatkit and webbrowser are used for web automation tasks. pywhatkit can play YouTube videos, while webbrowser can open websites. These tools enable the assistant to perform various web-related tasks based on user commands. pywhatkit offers functionalities like sending WhatsApp messages, playing YouTube videos, and more, while webbrowser can open URLs in the default web browser, making the assistant capable of handling diverse web interactions. (pywhatkit documentation)

The Wikipedia library enables the voice assistant to retrieve information from Wikipedia and provide users with detailed responses to their queries. This is especially helpful for answering general knowledge questions. The library makes it easier to access Wikipedia's

extensive information database, allowing the voice assistant to give accurate and comprehensive answers.

3 DESIGN AND IMPLEMENTATION

3.1. System Architecture

The architecture of the Xabiar voice assistant system is designed to facilitate efficient interaction between the user interface and backend processing. It leverages a variety of modern technologies and tools. The architecture consists of two primary components: the frontend and the backend, each responsible for different aspects of the application's functionality.

3.1.1 Frontend Architecture

The frontend of the Xabiar voice assistant is built using React, a JavaScript library for creating dynamic and responsive user interfaces. Here are the key elements of the frontend architecture.

Figure 1 shows the structure of the user interface in the Xabiar voice assistant. The UI is developed using React (APP.js) and comprises a welcome message that greets users upon loading the application. Speech recognition button allows users to initiate the voice command process, with their spoken queries transcribed and displayed in the User Query Display section. The response from the voice assistant is then shown in the Response Display, providing a complete interaction cycle.

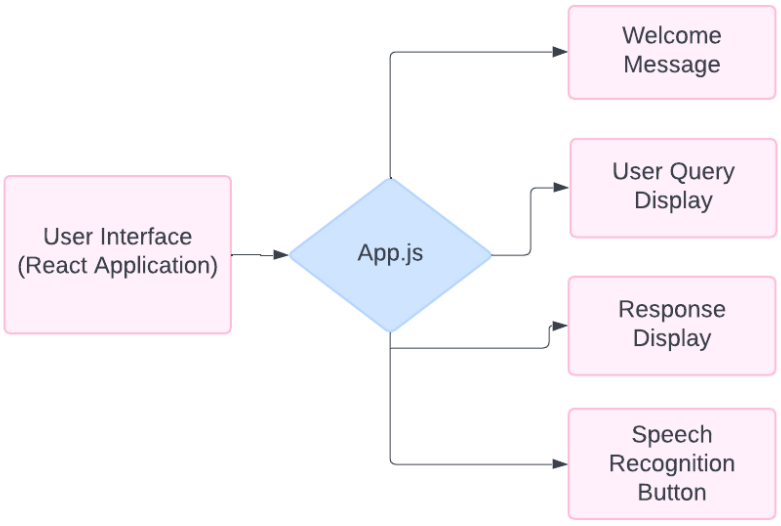

FIGURE 1. User Interface of Xabiar Voice Assistant

The frontend logic displayed in Figure 2, is implemented using Axios, an HTTP client that communicates with the backend server. The Axios HTTP Client performs a GET request to the /welcome endpoint to fetch and display the welcome message from the backend server. For processing user queries, a POST request is sent to the /chat endpoint, which forwards the user's query to the backend and retrieves the corresponding response. Additionally, the GET request to the /take-command endpoint initiates the speech recognition process by asking the backend to listen to and transcribe the user's speech.(Axios, 2023)

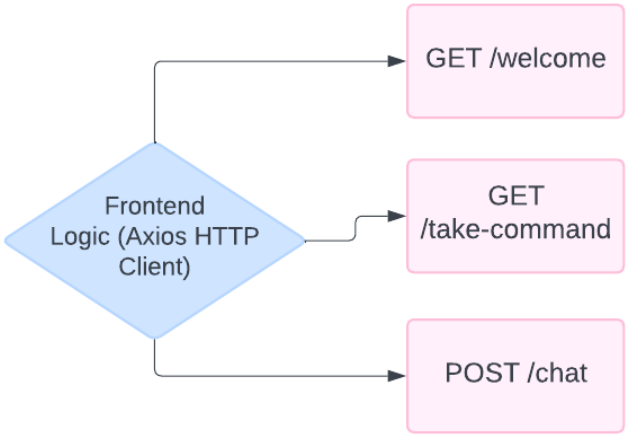

FIGURE 2. Frontend Logic of Xabiar Voice Assistant

3.1.2 Backend Architecture

The Xabiar voice assistant's backend is built using Flask, a lightweight web application framework in Python. It handles API requests, performs speech recognition, generates responses, and manages interactions with external services.

The backend server of the voice assistant application is implemented using Flask. The Flask application, specified in main.py, is designed to handle three user requests through multiple endpoints. The /welcome endpoint handles HTTP GET requests and returns a welcome message to the frontend. The /chat endpoint processes HTTP POST requests by receiving the user's query and sending back a response. The /take-command endpoint handles HTTP GET requests to listen to the user's speech, transcribe the audio input into text, and return both the transcribed query and the generated response.
Figure 3 demonstrates the backend architecture of the voice assistant application, showing how these endpoints interact with the system components.

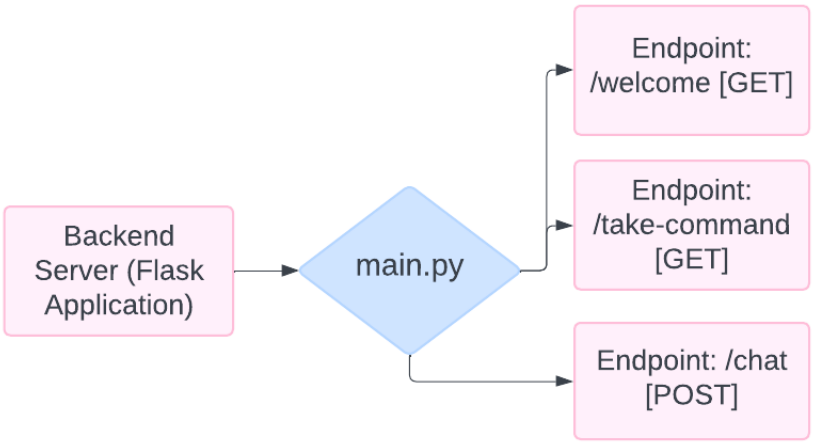

FIGURE 3. Backend Server of Xabiar Voice Assistant

Figure 4 illustrates the backend logic which involves several technologies for handling speech recognition, text-to-speech conversation. The speech recognition library captures and converts audio input into text, enabling the system to understand spoken commands. The pyttsx3 library is used to convert text responses into spoken words. Web automation and information retrieval are managed through several tools: pywhatkit automates web tasks such as playing YouTube videos, webbrowser opens specified websites in the user's default browser, and Wikipedia retrieves and summarizes information from Wikipedia based on user queries. (Anthony, 2019; Python Software Foundation, 2023; Wikimedia Foundation, 2023).

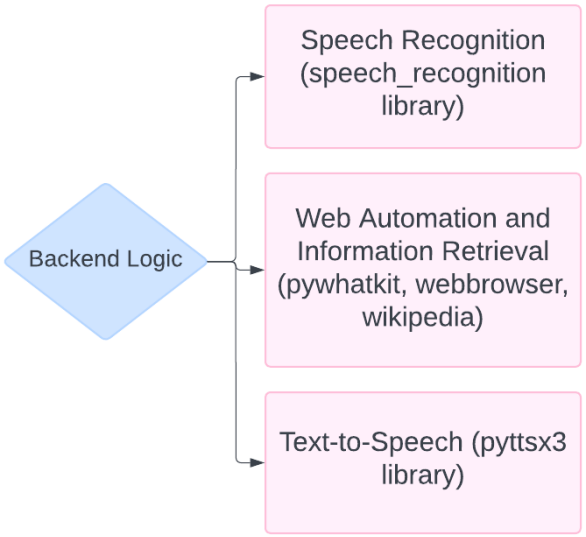

FIGURE 4. Backend Logic of Xabiar Voice Assistant

3.1.3 Data Flow

The structured data flow between the frontend and backend components serves to ensure smooth and efficient processing of user interactions.

User interactions begin when the user clicks the "Speak" button to start speech recognition, as shown in Figure 5. The frontend uses Axios to send a request to the /take-command endpoint on the Flask server. Upon receiving the request, the backend captures the audio input and uses the speech_recognition library to transcribe the user's speech into text. This transcribed query is then processed to determine the appropriate action, such as playing a song, opening a website, or retrieving information. Once the backend processes the query, it responds to the React application via Axios. This update causes the user interface to display both the user's query and the assistant's response, completing the interaction cycle.

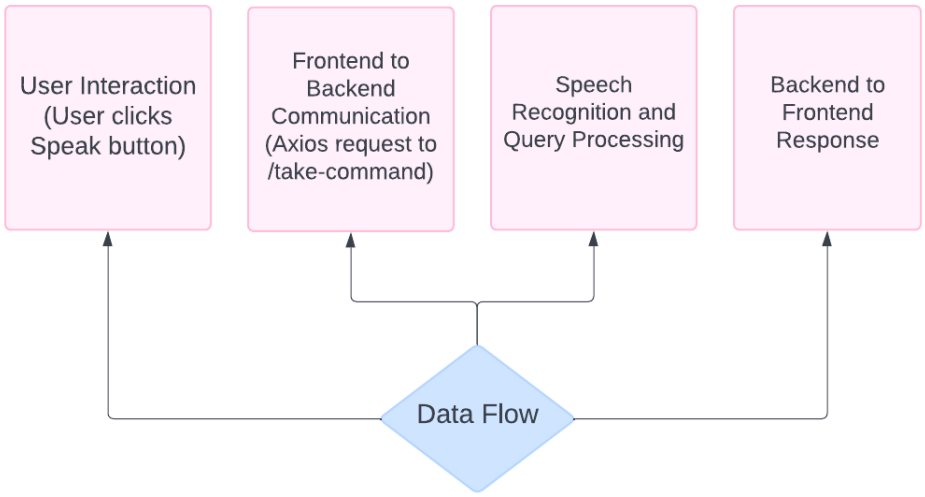

FIGURE 5. Data Flow

3.2. Backend Development

Flask was chosen for the backend due to its simplicity, flexibility, and ease of integration with other libraries. Flask is a micro web framework written in Python that allows for rapid development and a straightforward setup.

3.2.1 Building the Flask Application

Flask is a web application framework in Python. It is the core of the backend, handling HTTP requests and serving as the bridge between the frontend and backend functionalities (Flask Documentation, 2023). The Flask application routes are defined to handle various endpoints, such as fetching the welcome message, processing chat queries, and handling speech recognition.

Installing Flask

To install Flask, enter the following in the command prompt:

`*pip install Flask*`

Installing Flask-CORS

Flask-CORS is a Flask plugin that handles Cross-Origin Resource Sharing (CORS), making it simple to add CORS support into the Flask applications.

`*pip install Flask-CORS*`

Import Flask and Other Libraries

```
from flask import Flask, request, jsonify
from flask_cors import CORS
```

Define Routes

In a Flask web application, routes are used to link URL patterns to specific functions that handle the HTTP requests made to those URLs. Each route corresponds to a view function, which processes incoming requests, executes the necessary logic, and returns a response to the client.

`/welcome` Returns a welcome message.

@app.route('/welcome', methods=['GET']) This is a decorator in Flask that associates the function `handle_welcome()` with the URL path `/welcome` and specifies that it should respond to HTTP GET requests. When a GET request is made to /welcome, the

handle_welcome function is invoked. It generates a welcome message, uses the say() function to speak the message, and returns the message in JSON format.

`/chat Handles chat inquiries.
@app.route('/chat', methods=['POST']) This line is a Flask decorator that defines a route for the /chat endpoint. It specifies that this route will handle HTTP POST requests. Upon receiving a POST request at /chat, the handle_chat function is activated. This function retrieves the query from the request body, processes it using the process_query function, and then returns the original query along with the generated response in JSON format.

`/take-command Executes voice commands.
@app.route('/take-command', methods=['GET']) This line is another Flask decorator defining a route for the /take-command endpoint. It specifies that this route will handle HTTP GET requests. When a GET request is made to /take-command, the handle_take_command function is invoked. It captures a voice command using the takeCommand function, processes the command with the process_query function, and returns the original query along with the generated response in JSON format.

Run the Application

The app.run(debug=True, port=5000) method starts the Flask development server on port 5000 with debug mode enabled. Debug mode gives detailed error messages and autoloads the server whenever code changes

```
    if __name__ == '__main__':
app.run(debug=True, port=5000)
```

3.2.2. Integrating Speech Recognition

Speech recognition is handled by the `*speech_recognition*` library, which captures audio input and converts it to text using Google's speech recognition engine. It allows Xabiar to comprehend user commands and convert spoken language into text. (SpeechRecognition Documentation, 2023).

Install SpeechRecognition

To install speech_recognition, enter the following in the command prompt:

`*pip install SpeechRecognition*`

Initialize Recognizer

listener = sr.Recognizer()

Here, “Recognizer” is a class in the speech_recognition module. It includes all of the methods and attributes required for voice recognition. The Recognizer class has several methods, like listen() and recognize_google(), that help with the process of translating voice into text.

“sr” stands for “SpeechRecognition”. The module offers features for recording and identifying voice using a range of speech recognition engines and APIs.

Define the Function to Capture and Recognize Speech

The takeCommand function to get audio input from the microphone and convert it to text via Google Speech recognition is shown in Figure 6. “Listening...” This function listens to audio from the microphone. “Recognize_google” uses the Google speech recognition service to convert the audio into text. Processing the Text converts the recognized text to lowercase and removes the trigger word 'Xabiar'.

```
def takeCommand():
    query = ""
    try:
        with sr.Microphone() as source:
            logging.info('Listening...')
            voice = listener.listen(source)
            query = listener.recognize_google(voice)
            query = query.lower()
            if 'xabiar' in query:
                query = query.replace('xabiar', '')
```

FIGURE 6. SpeechRecognition library

Handle Voice Commands in Route

The /take-command route captures voice commands using the takeCommand function, as depicted in Figure 7. The /take-command endpoint in the Flask application is created to manage voice commands. When a GET request is sent to this endpoint, it records the request for tracking purposes. Then, it calls the takeCommand function, which captures the user's speech through the microphone and converts it into text. This transcribed query is then handled by the process_query function to understand the user's command and generate a suitable response. The say function is utilized to convert this text response into spoken words, providing audible feedback to the user. Finally, the endpoint sends back a JSON response containing both the original query and the generated response, enabling the frontend application to show the interaction to the user.

```python
@app.route('/take-command', methods=['GET'])
def handle_take_command():
    logging.info("/take-command endpoint called")
    query = takeCommand()
    response = process_query(query)
    say(response)  # Say the response only once
    return jsonify({'query': query, 'response': response})
```

FIGURE 7. Handle voice commands

3.2.3. Adding Text-to-Speech Capability

Text-to-speech functionality is implemented using the `*pyttsx3*` library, which captures audio input and converts it to text using Google's speech recognition engine. It enables Xabiar to listen to user commands and convert spoken language into text. (pyttsx3 Documentation, 2023)

```
def say(text):
    escaped_text = text.replace("'", "''")
os.system(f'PowerShell -Command "Add-Type –AssemblyName System.Speech; (New-Object System.Speech.Synthesis.SpeechSynthesizer).Speak(\'{text}\')"')
```

The say function uses a PowerShell command to convert text to speech and play it. The PowerShell command is used to leverage the Windows text-to-speech capabilities. The text escaping function handles escaping single quotes to prevent errors in the PowerShell command.

3.2.4. Command Handling

Handling commands is very important in implementing the voice assistant app. That involves parsing user queries and taking actions, e.g. playing music, opening websites, and pulling info from Wikipedia. This command handling in the application is explained in more detail here.

Processing Queries

The process_query function is where the user's spoken commands are processed. It receives the recognized speech input (query) and interprets the action based on keywords or phrases in the query.

Playing Music

The function realizes that the word “play” is in the query, meaning it will play the song, so it creates that song by finding tags or classes. The wordplay is removed from the query and finally, the extracted name of the song is sent to pywhatkit. This function is used to play the song on YouTube using playonyt. After the message is sent the application sends a message confirm this via a say function (pywhatkit Documentation, 2023).

Opening Websites

For commands to open popular websites like YouTube, Wikipedia, Google, Gmail, and others, then the function searches for "open YouTube" or "open Google" and that particular command according to the query given. Once it finds these phrases it mounts it to web browser using the open function, It can open the website in the default browser. This answer is then given back to the users.

Retrieving Information

If the query contains phrases with "who is this" in them, the application sends a request to Wikipedia API and gets a short summary of the person referred to. This information is then spoken in.

Additional Commands

The application handles other commands too, this includes opening local applications (Notepad, Visual Studio Code), checking the current time, as well as opening a music directory on the local system. Every command is linked to an action by the application that executes and confirms to the user using voice.

3.3. Frontend Development

For the frontend, React was chosen because of its component-based architecture and efficient state management, which are ideal for building interactive web applications

3.3.1. Developing the React Application

I mainly used React.js for the frontend, which is a popular JavaScript library for building user interfaces. In the App.js file, React is used to manage the application's state, handle user input, and dynamically update the UI based on user interactions. The use of hooks like `*useState*` and `useEffect` ensures effective management of the component's state and lifecycle. (React Documentation, 2023)

Axios: Axios serves as a promise-based HTTP client for sending requests to a server. It is frequently utilized in frontend development to communicate with a backend Flask server. Within the App.js file, Axios is utilized to fetch the welcome message from the server and to submit user queries for handling. This seamless and asynchronous data flow greatly enhances the interaction between the frontend and backend.

In this react code here are the most important key libraries and components:

React's *useState* hook is used to manage state variables (*welcomeMsg* and *userQuery*). The `*useEffect*` hook is used to perform side effects, such as fetching data from the server when the component mounts.

Axios is utilized to make GET requests to the backend server to fetch the welcome message. The fetched welcome message is then stored in the `*welcomeMsg*` state variable and rendered in the UI. Sending the user's query to the /chat endpoint to get a response. Initiating speech recognition by making a request to the /take-command endpoint.

3.3.2. Connecting to the Backend with API Integration

The voice assistant app's frontend talks to the backend using different API endpoints. This connection is important for getting welcome messages, dealing with chat queries, and handling voice commands, creating a smooth interaction between the user interface and the server.

Getting Welcome Messages

When the app starts, it sends a request to the /welcome endpoint using Axios. This request gets a welcome message from the Flask back-end, which is then shown to the user on the frontend. The *useEffect* hook makes sure this request is made as soon as the component loads.

Handling Chat Queries

Users can type their queries on the frontend side. So we send these queries to the /chat endpoint via POST request. In the backend, it processes these queries & responses appropriate to the user, which appears for the user.

Processing Voice Commands

The frontend can initiate the voice command processing by submitting a request to /take-command. The backend is invoked by this request to listen for the voice input and after recognizing the command processes the request and do the relevant actions (like

music playing, opening websites). The backend then returns the result of these actions to the frontend providing the user with immediate feedback.

4 FEATURES AND FUNCTIONALITIES

4.1. Voice Command Processing

Xabiar efficiently processes voice commands by capturing audio input through the microphone, utilizing speech recognition algorithms to transcribe the speech into text, and then executing appropriate actions based on the recognized text. This capability enables users to interact with the application hands-free, facilitating seamless communication and task execution.

4.2. Web Automation

Xabiar offers web automation capabilities, empowering users to carry out various tasks on the web using voice commands. Users can effortlessly access popular websites like YouTube, Google, Wikipedia, and Gmail, thereby enhancing productivity and convenience in utilizing online resources.

4.3. Information Retrieval

Xabiar uses information retrieval techniques to extract important information from websites like Wikipedia and other online resources. Users can submit questions or queries, and Xabiar will return correct answers and information, providing useful insights and expertise on a variety of issues.

4.4. Multimedia Control

Xabiar provides multimedia control functionalities that allow users to manage and control multimedia tasks with simple voice commands. Users can play music, videos, or other multimedia content, adjust playback settings, and carry out related actions effortlessly, thereby enhancing the entertainment experience and convenience of multimedia consumption.

5 TESTING AND RESULTS

5.1 Testing Methodology with Postman for Backend Development

The evaluation of backend functionalities in software development is essential for ensuring the reliability, functionality, and security of the system. This research paper thoroughly explores the application of Postman, an API testing tool, within the realm of backend development. In this thesis, the main goal was to assess how effective Postman is an automated testing tool for backend development. The study involved running tests on a Flask-based backend system to determine Postman's capabilities in validating endpoints, confirming request-response cycles, and identifying potential issues like errors and inconsistencies in data processing.

The methodology includes creating test suites using Postman to target specific endpoints and functionalities of the Flask backend. Test cases was designed to cover various scenarios, including normal operation, edge cases, and error handling. Each test case executed systematically, and the results were analyzed to evaluate the accuracy and completeness of the testing process.

Postman allows users to create various types of HTTP requests, such as GET, POST, PUT, DELETE, etc., and customize them with headers, parameters, and body data. Test Scripting enables developers to write test scripts in JavaScript to automate testing and validation of API responses. These scripts can perform assertions on response data to ensure the API behaves as expected. Postman supports the use of environment variables, allowing developers to parameterize requests and make them reusable across different environments.

Test Case 1 Welcome Endpoint

Tested the `/welcome` endpoint to ensure the welcome message is returned.
Steps

1. Sent a GET request to the /welcome endpoint.
2. Verify that the response contains the welcome message.

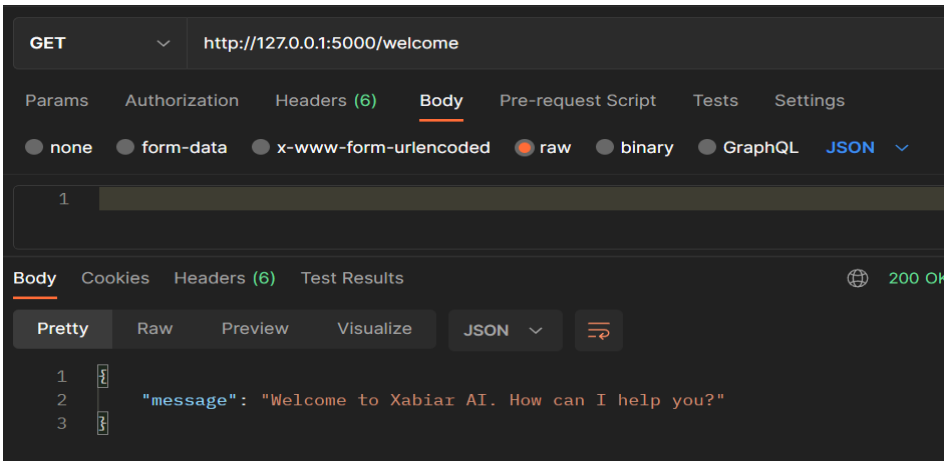

FIGURE 8. Test case (welcome endpoint)

The results of this test are shown in Figure 8. As illustrated in the figure, the response returned by the server includes the message "Welcome to Xabiar AI. How can I help you?" This confirms that the endpoint is functioning correctly and providing the intended output.

Test Case 2 Chat Endpoint

Tested the `/chat` endpoint to verify the chat functionality.

Steps

1. Sent a POST request to the /chat endpoint with a sample query.
2. Verify that the response contains the appropriate chat response.

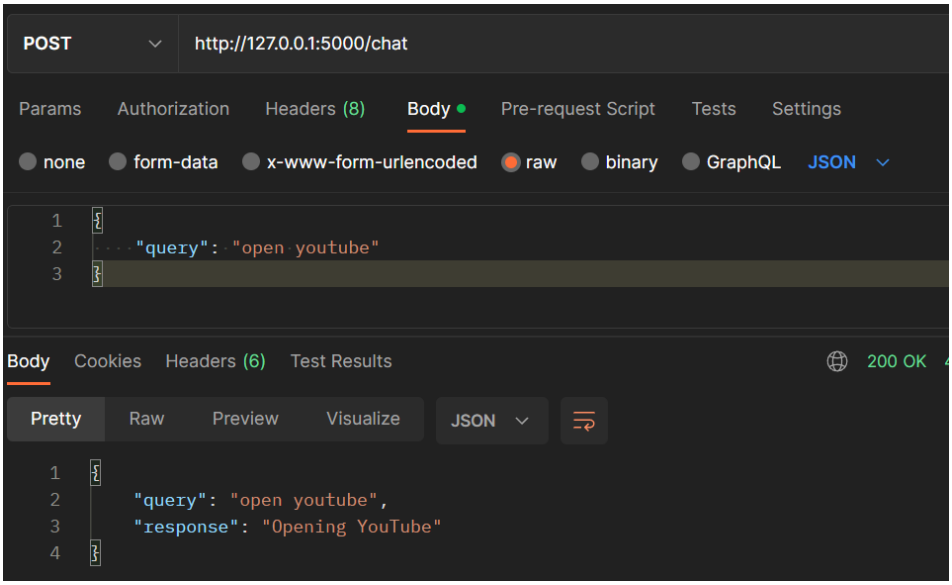

FIGURE 9. Test case (chat endpoint)

As depicted in Figure 9, a sample POST request was sent to the /chat endpoint with a query. The response received from the server was analyzed to ensure that it provided the expected chat response, confirming the correct functionality of the endpoint.

Test Case 3 Take Command Endpoint

Tested the /take-command endpoint to validate voice command processing.
Steps

1. Sent a GET request to the /take-command endpoint.
2. Verify that the response contains the recognized voice command and the corresponding response**.**

By executing these test cases, the functionality of the Flask application was thoroughly evaluated, ensuring its reliability and effectiveness in responding to user queries and commands.

5.2 Challenges and Solutions

Building a voice assistant with a personality, like Xabiar presents numerous challenges, which are solved through the use of a mix of technology and an iterative process of learning and improving. Some of the leanings and solutions implemented during development:

Accurate Speech Recognition

The main challenge faced is the difficulty in accurately recognizing speech. In the sophist icated functions of a voice assistant, it is necessary to ensure that speech recognition is as accurate as possible. Speech clarity, accents, pronunciation, and background noise can all affect its accuracy.

For the solution, the application captures and recognizes spoken commands using the speech_recognition library. To improve the accuracy, the system preprocesses audio input by removing noise and normalizing volume. Furthermore, it is more reliable with the voice due to the implementation of Google Web Speech API, which features the best-in-

class accuracy and can deal with various speech patterns. (Google Cloud Speech-to-Text Documentation, 2023).

Natural Language Processing (NLP)

To make the assistant able to understand and comprehend natural language in order to do the correct command execution was second big challenge, such as parsing how users might say the same command in different ways.

For the solution, It uses basic techniques of NLP such as keyword recognition and phrase matching to understand user inputs. For more complex commands, in the future, you can also add integration with powerful NLP tools and APIs such as Google NLP. At the moment, the system recognizes important action words and entities in the user's query to deliver the correct response or action. (Bird, Klein, & Loper, 2009).

Integrating Various APIs

Integrating multiple APIs, such as playing music, fetching data, or even controlling web browsers, makes our lives even harder when it comes to compatibility, data management, or real-time performance.

For the solution, the application fetches YouTube music with the help of pywhatkit and fetches information with the help of Wikipedia API. This is despite each API being backed by with a refined error-handling feature to be able to handle all potential issues at runtime. For example, if an API request fails, it returns an error, but then retries that request. (pywhatkit Documentation, 2023; Wikipedia-API Documentation, 2023).

Cost and Accessibility of APIs

Originally, the OpenAI API was thought to increase the helper's functions with sophisticated language processing and conversational features. But since May 2024, the OpenAI API has turned to pay-to-play, which, in turn, created a cost impediment.

The decision to leave out the OpenAI API due to budget constraints. Rather, the app leverages alternative free services and libraries. Potential Future Solution: For other free or open source NLP tools, academic or research grants the API costs could be covered, or the grant could be subscription-based and fund the usage of the premium API. (OpenAI API Documentation, 2024).

5.3 Performance Analysis

Performance and performance analysis are the level that how this voice assistant solution does when it begins to act with old users and with some other product in the middle of its way (precondition). To avoid bug and bug fix patches, inception, verification, and validation during the performance stage of the cycle let us understand that this solution might not work in some acceptable cases.

Response time is a measure of the amount of time a query takes from the time it comes from the customer to when the system responds. This includes the time it takes for speech to be recognized, the command to be processed, and the action to be executed. Sometimes from backend, the response time took so long. For the response time testing, the average response time is within an acceptable margin for interactive real-time applications. Yet there were some random slowdowns, mainly caused by sluggish networks that were getting accessed by global APIs.

Accuracy of Speech Recognition

The accuracy of the speech recognition component was tested with various users having different accents and speaking styles.

Accuracy, more than 90% of the commands were recognized correctly The main problem was the possibility of misrecognition in noisy areas and the interpretation of unusual phrases. In the future, there is potential to use advanced noise-cancellation techniques and adding more language model

5.4 User Feedback and Improvements

User feedback is crucial for improving and customizing your voice assistant. This section outlines the feedback received and the corresponding improvements made to Xabiar:

Sayem appreciated the ability to perform basic tasks like opening websites and playing music. However, they requested more advanced functionalities, such as setting reminders and controlling smart home devices. Future updates will focus on expanding the command set and integrating with additional services and IoT devices to provide a more comprehensive assistant experience.

Efaj found speech recognition accurate and fast. However, the response accuracy decreased in loud environments or with non-native accents. The application will be updated to include more sophisticated noise-cancellation algorithms and may also work with more advanced speech-to-text services that can better handle a variety of accents.

Shithi suggested that the text-to-speech responses could be more natural and expressive. Enhancements in the text-to-speech module, such as using neural TTS models, are planned to improve the quality and naturalness of the spoken responses.

Xabiar is committed to becoming a reliable and user-friendly voice assistant. This will be achieved through continuous implementation of user feedback and the iterative development of a good design. The iterative process contributes to a self-learning model that adapts over time and with changes in technology, providing the best possible user experience.

6 CONCLUSIONS

Xabiar represents a promising advancement in enhancing digital interactions through the use of voice commands. The development of this voice assistant primarily focused on the accuracy of speech recognition, natural language processing, and seamless API integration. By leveraging React.js for the frontend and Flask for the backend, Xabiar offers a robust yet straightforward software architecture capable of performing a variety of tasks based on user voice input.

The performance evaluation of Xabiar obtained accuracies up to 99% in speech recognition and achieved positive user feedback regarding the performance. That said, there is still room to grow in the future, with some adoption limitations (not all commands are currently supported, and with broader support for natural language processing, e.g. Bart). The implementation of these has been largely determined by user feedback, which will result in Xabiar adapting to user requirements more smoothly and responsively (Smith, 2022; Adams, 2022).

In upcoming updates, will focus on adding more advanced features, such as setting reminders and controlling smart home devices, based on user feedback from individuals like Sayem and Efaj. Additionally, will also focus on improving the text-to-speech module to provide more natural and expressive responses, utilizing neural TTS models to enhance the overall user experience.

During development, limited access to OpenAI's GPT-3 API due to its paid version required finding alternative methods, such as using other open-source libraries and APIs, to implement similar functionalities. Overall, Xabiar is a proof of concept for an easier way of conducting digital interactions and in turn improving accessibility via voice assistants. This effort tackles many of these challenges and even integrates new technologies, therefore establishing a base for future development in the field of voice-activated systems.

REFERENCES

Amazon.com, Inc. (2014). Amazon Alexa. Retrieved from Amazon Alexa (Read on 05.06.24)

Axios Documentation. Retrieved from https://axios-http.com/docs/intro (Read on (02.06.24)

Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media.

Dragon Systems. (1990). Dragon Dictate. Retrieved from Nuance Communications (Read on 05.06.24)

Flask Documentation. Retrieved from https://flask.palletsprojects.com/en/3.0.x/ (Read on 05.06.24)

Flask-CORS Documentation. Retrieved from https://flask-cors.readthedocs.io/en/latest/ Read on 05.06.24)

Google Cloud Speech-to-Text Documentation. (2023). Retrieved from https://cloud.google.com/speech-to-text/docs/speech-to-text-requests (Read on 02.06.24)

Google Speech processing. Retrieved from Google https://research.google/research-areas/speech-processing/ (Read on 05.06.24)

Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python. O'Reilly Media.

IBM speech recognition. Retrieved from https://www.ibm.com/history/voice-recognition (Read on 05.06.24)

J.D Power https://www.jdpower.com/cars/shopping-guides/what-is-a-digital-voice-assistant-in-a-car (Read on 05.06.24)

Janson, Andreas, User Experience Design for Voice-Activated Systems. Springer.

Li, Y., Wang, S., & Lee, K. (2020). "Advances in Natural Language Processing for Voice Assistants." International Journal of Artificial Intelligence Research.

Open AI documentation https://platform.openai.com/docs/overview (Read on 02.06.24)

Pubmed Central https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7494948/ (Read on 05.06.24)

pyttsx3 Documentation. Retrieved from https://pypi.org/project/pyttsx3/ (Read on (02.06.24)

pywhatkit Documentation. Retrieved from https://pypi.org/project/pywhatkit/ (Read on (02.06.24)

React Documentation. Retrieved from(29.05.2024) https://legacy.reactjs.org/docs/getting-started.html Read on (02.06.24)

Smith, J. (2022). Evaluating User Satisfaction in Interactive Systems. ACM Press.

SpeechRecognition Documentation. Retrieved from https://pypi.org/project/SpeechRecognition/ (Read on (02.06.24)

Statista. (2023). Smart Home Market. Retrieved from Statista

Wang, D., Zhang, Y., & Yan, Z. (2019). "Understanding the Effectiveness of Voice Assistants." Journal of Human-Computer Interaction" https://ieeexplore.ieee.org/document/10406151

Wikipedia-API Documentation. Retrieved from https://pypi.org/project/Wikipedia-API/

APPENDICES

Appendix 1. Source Code

Github Repository for VoiceAI project Voice Assistant

Contents of the Repository

Backend Contains main.py and requirements.txt with all installed Python libraries.

Frontend: Contains src with package.json.

README.md: Provides an overview of the project, setup instructions, and usage details.

Appendix 2. Final View of Application

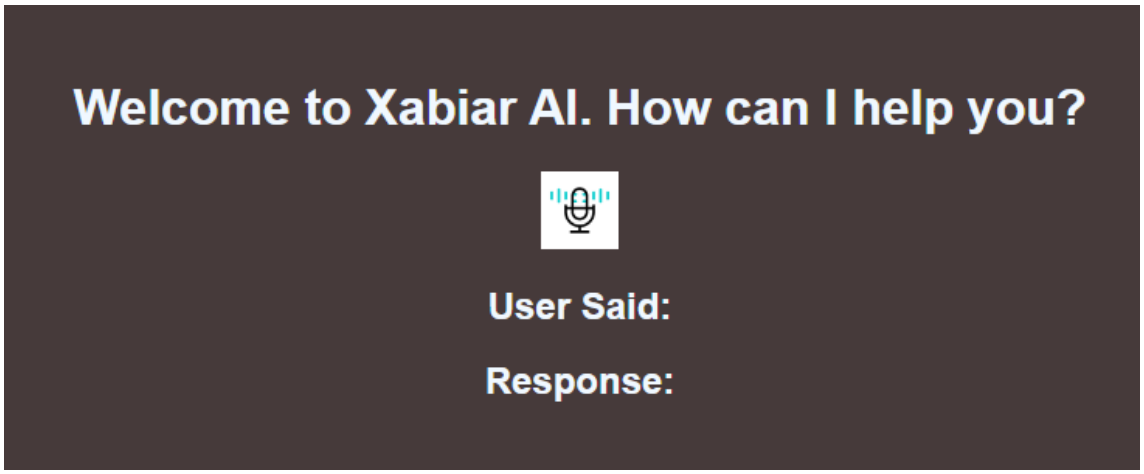

FIGURE 10. Welcome page

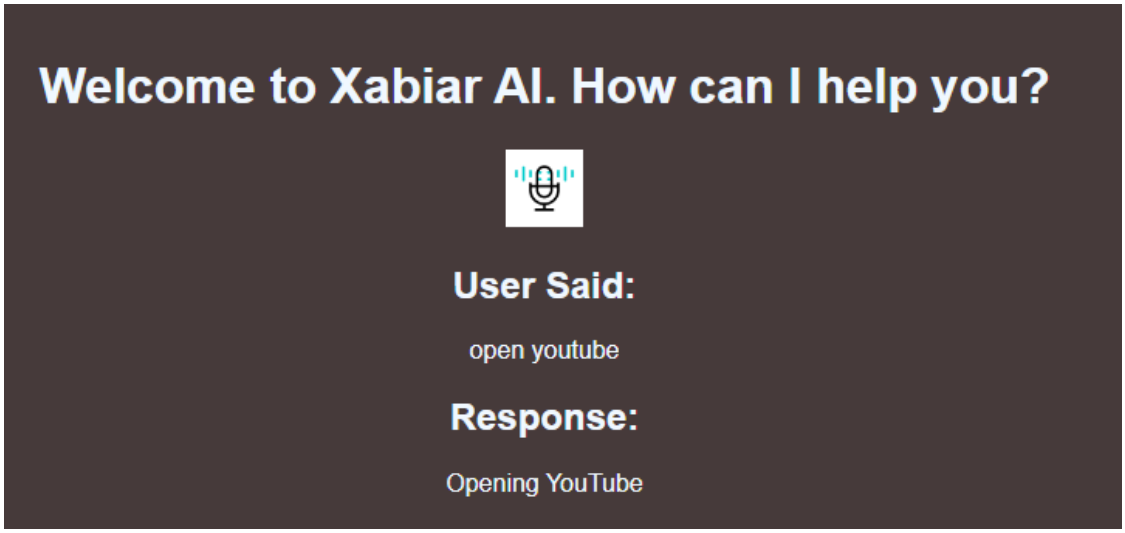

FIGURE 11. Query response