SCUBA VOICE ASSISTANT

PROJECT REPORT

NAME:ADHITHI.P.K

REGISTRATION NUMBER:25BSA10074

COURSE:CSE(CLOUD COMPUTING AND AUTOMATION)

INSTITUTE:VIT BHOPAL UNIVERSITY

Project Report: Voice Assistant – “Scuba”

Scuba – A Python-Based Voice Assistant

Introduction

Scuba is a lightweight, Python-based voice assistant designed to recognize user speech commands and respond through text-to-speech.
It performs simple tasks such as telling the time, speaking jokes, opening YouTube, and answering basic questions.
The project demonstrates core concepts in **speech recognition**, **text-to-speech**, and **command automation** using Python libraries.In an increasingly digital world, voice-controlled interfaces have become essential for enhancing user experience and

accessibility. Voice assistants like Siri, Alexa, and Google Assistant have revolutionized how users interact with technology, making hands-free operation and natural language interaction commonplace. However, understanding the underlying technology and building such systems from scratch remains a valuable learning experience for developers.

Problem Statement

Traditional computer interaction requires manual input through keyboard and mouse, which can be:

- Time-consuming for simple tasks
- Inaccessible for users with mobility impairments

- Inconvenient when hands are occupied with other activities
- Less intuitive than natural speech communication

Objective

The main objective of this project is to build a functional console-based voice assistant that:

- Listens to the user’s voice input
- Interprets commands using speech recognition
- Responds verbally using text-to-speech
- Performs simple predefined tasks automatically

Features of Scuba

✓ Speech Recognition

Uses `speech_recognition` library to convert user speech into text.

✓ Text-to-Speech

Utilizes `pyttsx3` to convert computer-generated responses into spoken output.

✓ System Actions

Scuba can:

- Tell its name
- Tell its age
- Speak out the current time
- Open YouTube
- Tell science jokes

✓ Error Handling

- Handles situations where speech is not understood
- Handles internet unavailability (for speech recognition)
- Responds politely to unknown commands

✓ Exit Command

User can say “exit”, “quit”, or “stop” to close the assistant.

Functional Requirements

These describe *what the system must do.*

FR1. Speech Recognition

The system must listen to the user’s voice and convert it into text.

FR2. Command Processing

The system must analyze the recognized text and match it with predefined commands.

FR3. Text-to-Speech Response

The system must speak out responses clearly using TTS.

FR4. Time Retrieval

The system must tell the current time when requested.

FR5. Joke Telling

The system must generate jokes using the `pyjokes` library.

FR6. Web Browser Integration

The system must open YouTube when the user commands.

FR7. Exit Command

The system must stop running when the user says “exit”, “quit”, or “stop”.

6. Non-Functional Requirements

These describe *how the system should behave.*

NFR1. Usability

- The assistant must be easy to use and understand.
- Responses must be clear and friendly.

NFR2. Performance

- Speech recognition should process input within 1–3 seconds.
- The system should respond without noticeable lag.

NFR3. Reliability

- The system should handle unrecognized speech gracefully.
- Should not crash even if voice input fails.

NFR4. Portability

- Should run on any computer with Python installed.
- All required libraries should be installable via pip.

NFR5. Maintainability

- Code must be modular and easily extendable with new commands.

Technologies Used

Component	Library / Module
Speech Recognition	`speech_recognition`
Text-to-Speech	`pyttsx3`
Jokes API	`pyjokes`
Browser Handling	`webbrowser`
Date & Time	`datetime`
Microphone Handling	Built-in (`speech_recognition` backend)

System Requirements

Hardware:

- A computer with Python installed
- Microphone for recording commands

- Speaker/headphones for responses

Software:

- Python 3.8 or above
- Required libraries:

```
pip install pyttsx3
speechrecognition pyjokes
pyaudio
```

Project Architecture

```
+----------------+
|   User Voice   |
+--------+-------+
         |
         v
+------------------------+
| Speech Recognition     |
| (speech_recognition)    |
+--------+----------------+
         |
         v
+-------------------------+
| Command Processing      |
| (if/elif tree)          |
+--------+----------------+
         |
         v
+-------------------------+
| System Actions          |
| TTS, opening browser…   |
+-------------------------+
```

Program Workflow

1. Start the assistant

2. Greet the user
3. Listen for a voice command
4. Convert voice to text
5. Match the text with available commands
6. Perform the corresponding task
7. Speak out the response
8. Continue until exit command is given

Results & Output

Scuba successfully:

- Recognizes user voice commands
- Responds using TTS
- Executes predefined system tasks

- Handles invalid or unclear speech
- Provides a smooth conversational experience

How to Run the Project

1.Open a terminal or command prompt in the project folder:

```
cd path/to/Scuba-Voice-Assistant
```

1. Run the Python script:

```
python scuba.py
```

2. Wait for Scuba's greeting (e.g., *"Hello, I am Scuba."*).
3. Speak commands like:
   a. "What is your name?"
   b. "How old are you?"
   c. "What is the time?"

d. “Tell me a joke.”

e. “Open YouTube.”

f. “Exit.”

Testing Approach

A structured test plan was followed:

1. Unit Testing

Each function was tested separately:

- `listen_command()` — tested for recognized/unrecognized speech
- `speak_text()` — tested for proper audio output
- Command handlers — tested with sample text commands

2. Integration Testing

Modules were combined and tested together:

- Speech recognition → command processor → TTS
- Checking if recognized text triggers correct responses

3. Functional Testing

Commands tested:

- “your name”
- “how old are you”

- “what is the time”
- “tell me a joke”
- “open youtube”
- “exit"

4. Negative Testing

Test cases with:

- Background noise
- Silence
- Random words not in command list
- Network failure during speech recognition

Challenges Faced

1. Speech Recognition Accuracy

- Background noise affected results, requiring noise adjustment.

2. Microphone Sensitivity Issues

- Different systems required different gain levels.

3. Time Delays

- Speech recognition sometimes took longer depending on connectivity.

4. Import & Installation Errors

- Users often face issues installing `pyaudio` on Windows.

5. Handling NoneType Responses

- If recognition fails, it returns `None` — had to add checks to avoid crashes.

Learnings & Key Takeaways

- Gained hands-on experience with **speech recognition** and **text-to-speech** systems.
- Understood how to create modular, scalable Python applications.
- Learned how to handle real-world issues such as noise, errors, and delays.
- Improved skills in debugging audio-related problems and managing dependencies.
- Saw how voice assistants operate internally and how commands are parsed.
- Understood the importance of structured testing before deployment.

Future Enhancements

You can expand this project further by adding:

- Wake-word detection (e.g., “Hey Scuba”)
- Weather updates
- Email/SMS automation
- Music player integration
- GUI interface

- Custom command training
- Conversational AI (GPT integration)

Conclusion

This project demonstrates how voice assistants work at a basic level and showcases the integration of speech recognition, text-to-speech, and command automation in Python. Scuba is simple, extensible, and forms a foundation for developing more advanced AI-powered assistants