2024 June

Maria - AI powered home assistant

Software Engineering

Krzysztof Pacyna 130648

Mateusz Rój 130672

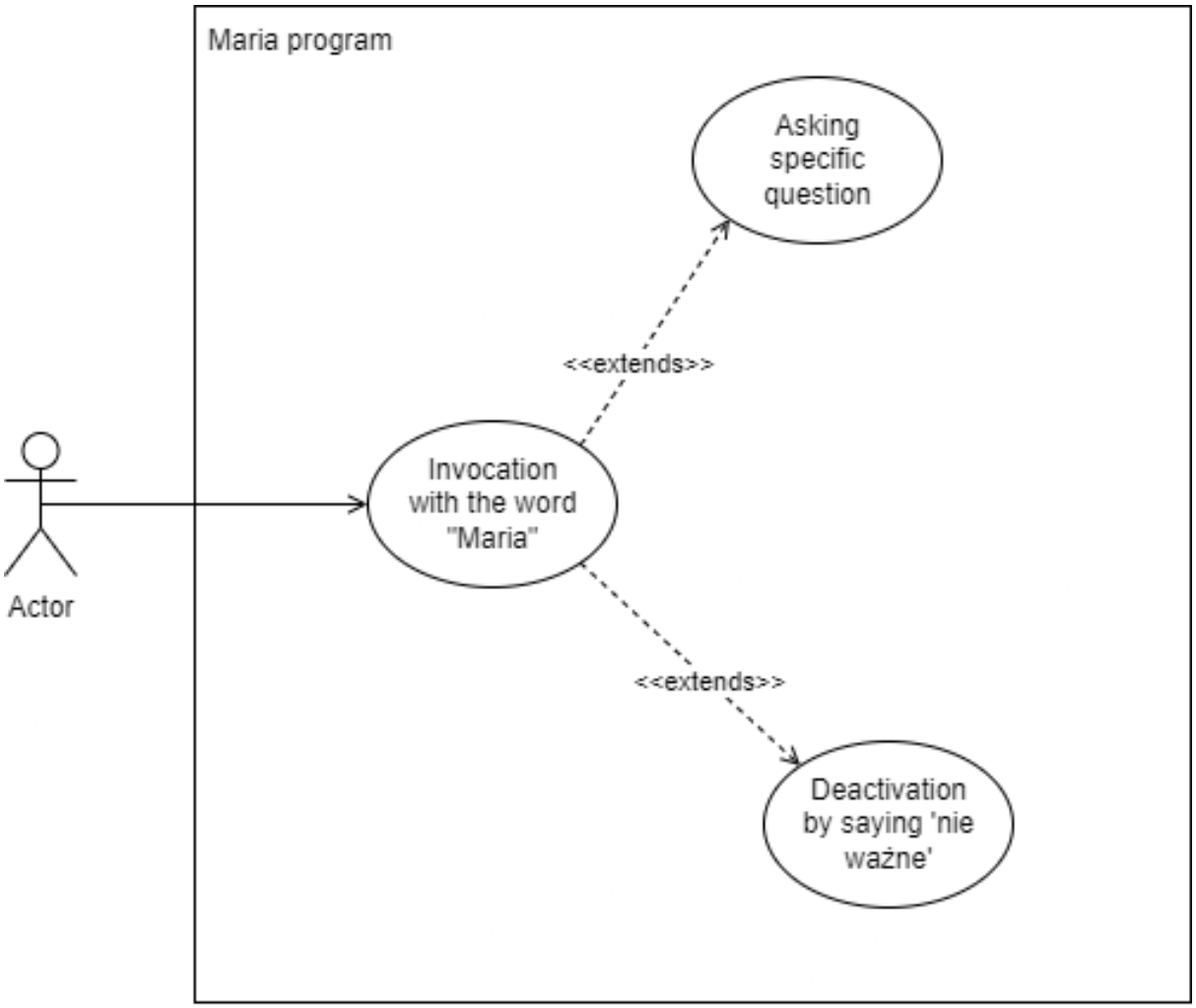
Table of contents

1. Introduction

The computer program named Maria is a home assistant utilizing artificial intelligence. The program is designed for home users to enhance their daily comfort. When called by its own name, Maria enters a listening state. Upon receiving a command, it generates the best-fitting response for the user's given question.

2. Specification of system requirements

2.1. Functional requirements



2.1.1. Use case

Name: Basic system usage

Preconditions: The app is running on a machine that is connected to the internet

Postconditions: User receives answer on his asked question

Primary actor(s): Resident of the house

Main scenerio	Step	Action
	1	User invokes program by saying the word “Maria”
	2	System indicates readiness by playing sci-fi sound
	3	The user vocalizes aloud the question to which they want to know the answer
	4	Program processes the user's query and vocalizes the generated response using a human voice
	5	The program ends its operation by playing different sci-fi sound
Extensions	**Step**	**Action**
	3a_1	User exits the program by saying “nie ważne”
	5	The program ends its operation by playing different sci-fi sound

2.1.2. User story: Exit on demand

Title: Exit on demand

Priority: High

Estimate: Launch

User story: It would be convenient to have the option to exit the application in case I accidentally launch it. This feature would provide more control and prevent unwanted usage.

Acceptance criteria: When the user inputs the correct word during the listening section, the program stops executing.

2.1.3. User story: Manual Query Control

Title: Manual Query Control

Priority: Medium

Estimate: Future version

User story: I would like to have the ability to manually control when my query is sent for analysis. Sometimes, I need time to think about what I want to say.

Acceptance criteria: The need is understandable, however, from a technological standpoint, it's very challenging to implement. It would require simultaneous listening and dynamic transformation of accumulated text into speech. Consequently, the speech recognition system would perform poorly as it cannot recognize speech in the full context of sentence

2.1.4. User story: No habla inglés

Title: No habla inglés

Priority: Medium

Estimate: Future version

User story: No sé inglés. Cada vez que hablo con esta maldita máquina, me responde en inglés que no entiende nada de lo que le digo. Necesito que funcione en mi idioma nativo.

Acceptance criteria: When the user starts speaking in a language other than the default English during the listening stage, the program will automatically switch to the recognized language.

2.2. Non-functional requirements

2.2.1. Usability

The program is reliable and resistant to user errors in many aspects. There are three scenarios in which the program may fail to perform its task correctly:

1. The program fails to recognize what the user said
2. The external OpenAI API is unavailable
3. Lack of internet access

Our program lacks a graphical interface and is controlled by voice, enabling absolutely anyone to use it. The instruction manual of our program is straightforward, as the program itself isnot complicated.

Additionally, when asked for instructions, our program is capable of presenting a complete user manual for itself.

2.2.2. Reliability

The program is available 24/7 because almost all of its operations occur on the local machine. The exception is sending queries to the OpenAI API, so the downtime corresponds to the downtime of the OpenAI API. This means that failures cannot be predicted unless OpenAI informs us about them.

2.2.3. Performance

In the initial versions, the program is quite slow, and this is the aspect on which future iterations of the program will focus the most. The program should be fast enough so that the user does not wait mindlessly for half a minute for a response. Current sequential operation of the program means that it is very slow and there is a lot of space for improvements.

Memory usage is minimal because the program does not save the history of sent or received questions. However, the situation is completely different when it comes to processor usage. Since artificial intelligence is also used for speech generation, the processor load is very high. If the unit on which we run our application does not have access to a graphics card capable of performing multiple parallel computations, the processor must handle the calculations, which will take exponentially longer.

2.2.4. Supportability

Individual program modules can be tested by running appropriate tests from the "tests" directory. You can monitor its operation, as all program operations are displayed in the console. When installing the software, it is recommended to use a Python virtual environment and install all dependencies using pip.

2.2.5. Design constraints

The only design limitation is the requirement for the user to be constantly connected to the computer network.

2.2.6. Implementation requirements

The program is actually quite simple, which means that there's no need to apply complex design patterns for its construction. Of course, standards should be followed, and a predefined programming paradigm should be adopted.

2.2.7. Interface requirements

The program does not have a graphical interface. The entire program is operated using voice commands, which significantly facilitates user interaction with the program.

The program does not have an API because the only thing such an API would do is send requests to OpenAI's API. Therefore such functionality is obsolete.

2.2.8. Physical requirements

The application would be best deployed on a Linux operating system, as it was originally developed on it. However, since it utilizes python packages also available on Windows, the program will also work on that system.

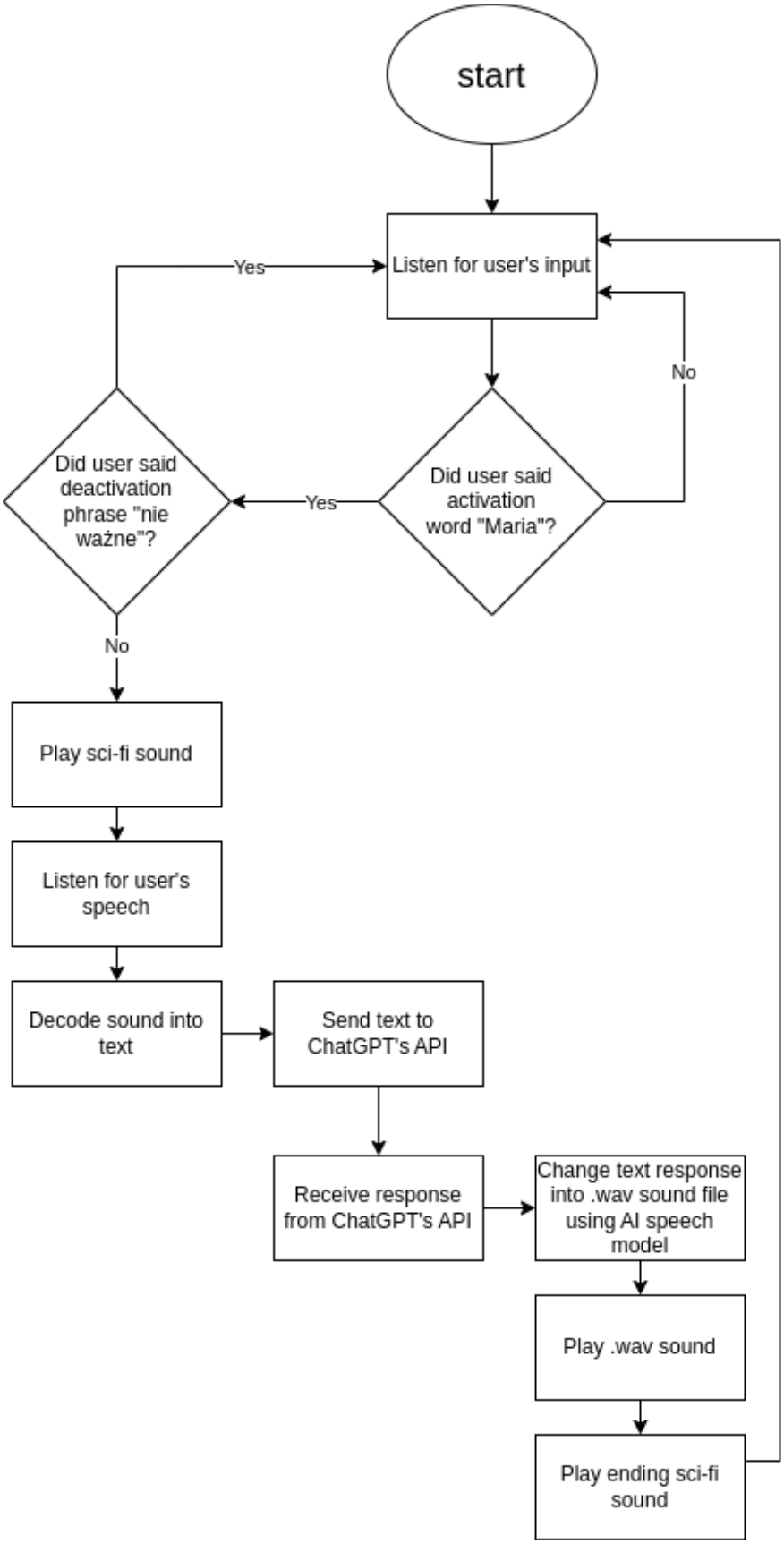
It's crucial for the application to have access to graphics card acceleration. Otherwise, all computations will have to be handled by the processor, which would be highly inefficient.

Additionally, the system on which the application will be deployed must have a microphone and speaker connected and set as the default.

Naturally, a stable internet connection is also mandatory.

3. System model

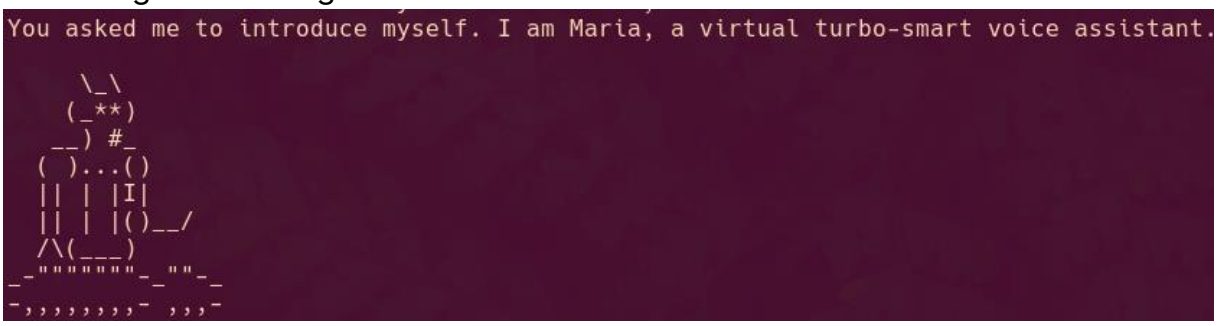
3.1. Block diagram



4. User interface design

Because our program is strictly controlled by voice, we do not have a Graphical User Interface (GUI) for users. Instead, we have something we call the Vocal User Interface (VUI). Our program is controlled strictly through sound. The program activation begins when the word "Maria" is spoken, and user input is expected after the activation sound plays. This means that at no stage does the user need to look at the monitor screen or interact with the computer through peripherals.

Despite this, for the sake of completing this task, we added some extra visual effects to our program. Because our program heavily relies on Artificial Intelligence, we added a drawing of a robot that is displayed in the terminal. When a user asks a question, the answer is of course provided by voice. However, if someone likes to see what the program is doing, we added a little easter egg in the form of an ASCII-encoded robot that appears as if it is delivering the message.



Additionally, because Windows users require graphical indication that their computer is doing something, we added a graphical installer. It shows the user the process of installing our software. Of course, after installation, our program runs in the background, exactly like Windows Cortana does.

5. System implementation

5.1 Operating System

Originally, our program was built on Linux, but thanks to Python virtualization, it can run on Windows and macOS as well. It requires no extra steps on any mentioned platform. The installation process is rather easy, but it can be more challenging for non-technical users. That's why we included a section titled "Installation" in our README.md file, so even inexperienced users can enjoy our software.

5.2 Programming Language

We decided to write our program in Python 3.11.2. It is not the latest version of Python; however, it is the version that the authors of the library we are using (TTS) recommend running. The library is crucial to us (as it stands for Text To Speech), so without it, users wouldn't hear our response. Despite this, the newer versions of Python do not offer significant enough improvements for us to upgrade.

5.3 Libraries

Before using our software, users must install the program dependencies. These are as follows:

- **SpeechRecognition** – Translates user input into text. It is one of the most used libraries, as our program constantly runs in the background and transforms heard voices into text.

- **PyAudio** – A simple library required by SpeechRecognition that listens to the operating system's default microphone.

- **python-dotenv** – Enables our program to read the .env file where information related to passwords and API keys is stored. The .env file is not sent to the repository, and every user has to fill in their own API key.

- **OpenAI** – A crucial library that sends our query to the OpenAI model. Without it, communication with external artificial intelligence wouldn't be possible. It is also the main engine of the program, as it generates responses to the user's questions.

- **TTS** – A very advanced and powerful AI-based text-to-speech library. It generates the .wav file using a model of a female voice, which is openly available for commercial use. It runs locally on the user's hardware, so no data is transmitted to external parties.

- **Pygame** – A simple library for playing .wav files.

Any of the above dependencies can be satisfied by running the following command:

pip3 install -r requirements.txt

6. The program itself

The described program can be found on the GitHub website at the following link:

www.github.com/Kennene/maria

It is a public repository available for the next 90 days.