NATASHA – VIRTUAL ASSISTANT

Computer Networks Project

Authors: **Andrei Grigor &**
Razvan Dologa
Supervisor: **Sl.dr.ing. Ioan-Valentin SITA**

Contents

1. Introduction

1.1 Purpose

Natasha is a virtual assistant similar to Google's Assistant or Amazon's Alexa differentiating itself by having a higher level of privacy and virtually infinitely customizable and upgradable. It is meant to simplify the completion of mundane tasks such as sending a WhatsApp message or looking up the weather conditions.

We want to prove that starting with a basic codebase that is open source and easy to understand we can create a very large community of users who can easily upgrade their own equipment, learn from each other and build a large ensemble of components which can be ready to add to every assistant.

1.2 Objectives

The main objective of this assistant is to provide a high level of privacy which gives each user the ability to feel safe in his own environment by giving them a basic assistant with an open source format which allows them to add functionalities to suit each user's specific needs.

It can be used in any domain from healthcare for incapacitated patients children wanting to search for something on the internet.

2. Specifications

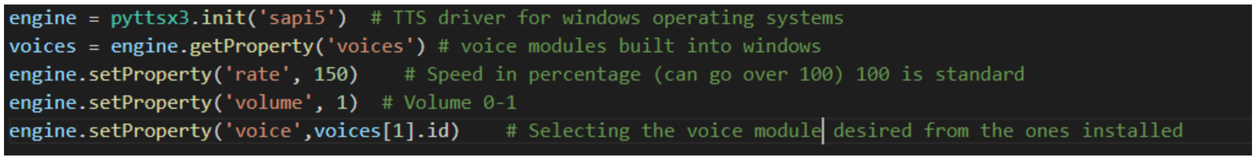
Natasha is built exclusively using python, an easy to use and learn language. Choosing this language allows us to access a multitude of free to use libraries which allows a modular design. We are using a specific library for each of the basic functionalities of the assistant. These libraries and each functionality's implementation will be presented in the following section.

3. Implementation

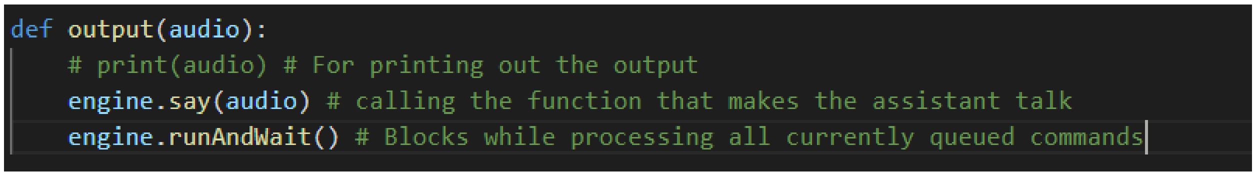
3.1 Functionalities

3.1.1 Text to Speech

The text to speech functionality is implemented using the python library PYTTSX3[1]. An application invokes the pyttsx3.init() factory function to get a reference to a pyttsx3.Engine instance. During construction, the engine initializes a pyttsx3.driver. DriverProxy object responsible for loading a speech engine driver implementation from the pyttsx3.drivers module. After construction, an application uses the engine object to register and unregister event callbacks, produce and stop speech, get and set speech engine properties such as volume or voice (language packs that are installed in your operating system), and start and stop event loops.



```
engine = pyttsx3.init('sapi5')  # TTS driver for windows operating systems
voices = engine.getProperty('voices') # voice modules built into windows
engine.setProperty('rate', 150)    # Speed in percentage (can go over 100) 100 is standard
engine.setProperty('volume', 1)  # Volume 0-1
engine.setProperty('voice',voices[1].id)    # Selecting the voice module desired from the ones installed
```



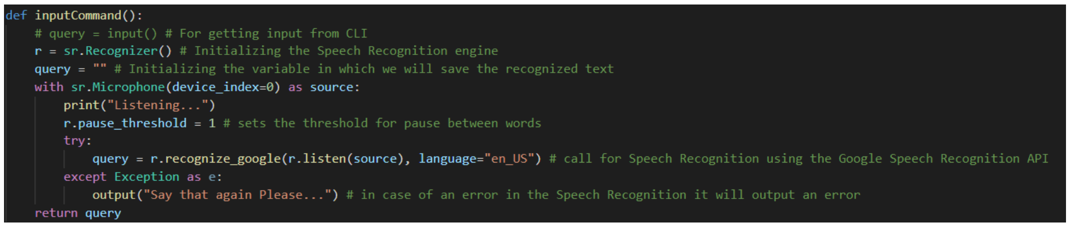
```
def output(audio):
    # print(audio) # For printing out the output
    engine.say(audio) # calling the function that makes the assistant talk
    engine.runAndWait() # Blocks while processing all currently queued commands
```

3.1.2 Speech Recognition

Speech Recognition is implemented by the SpeechRecognition[2] library for python. It supports the use of several speech recognition engines even having offline capabilities. The default one is google speech recognition.

PyAudio is required if and only if you want to use microphone input. PyAudio version 0.2.11+ is required, as earlier versions have known memory management bugs when recording from microphones in certain situations.

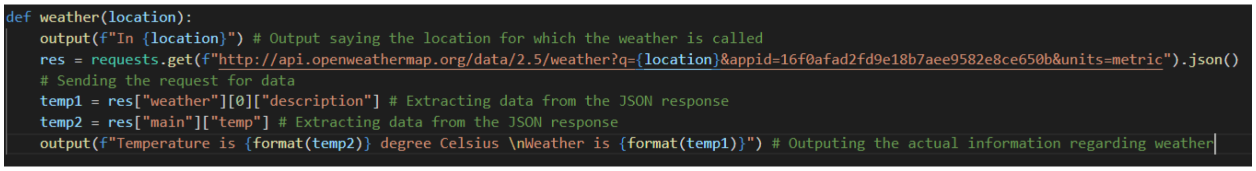
A Speech-to-Text API synchronous recognition request is the simplest method for performing recognition on speech audio data. Speech-to-Text can process up to 1 minute of speech audio data sent in a synchronous request. After Speech-to-Text processes and recognizes all of the audio, it returns a response.



```
def inputCommand():
    # query = input() # For getting input from CLI
    r = sr.Recognizer() # Initializing the Speech Recognition engine
    query = "" # Initializing the variable in which we will save the recognized text
    with sr.Microphone(device_index=0) as source:
        print("Listening...")
        r.pause_threshold = 1 # sets the threshold for pause between words
        try:
            query = r.recognize_google(r.listen(source), language="en_US") # call for Speech Recognition using the Google Speech Recognition API
        except Exception as e:
            output("Say that again Please...") # in case of an error in the Speech Recognition it will output an error
    return query
```

3.1.3 Sending Requests to Specific APIs

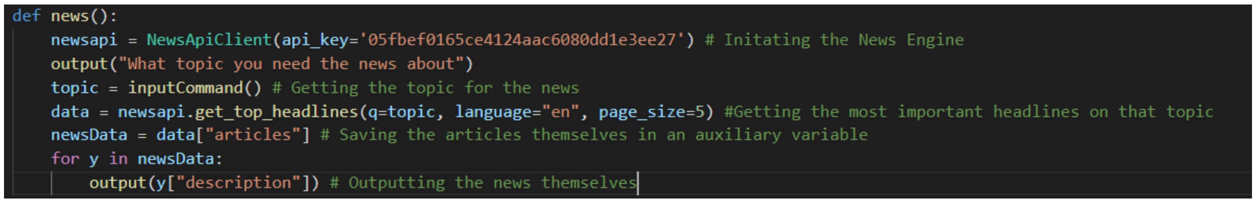
This is used for the weather data functionality and is implemented using the Requests library[3]. This is a library which can be used to make any API call via links using HTTP requests. There's no need to manually add query strings to your URLs, or to form-encode your PUT & POST data and receive responses in JSON format.



```python
def weather(location):
    output(f"In {location}") # Output saying the location for which the weather is called
    res = requests.get(f"http://api.openweathermap.org/data/2.5/weather?q={location}&appid=16f0afad2fd9e18b7aee9582e8ce650b&units=metric").json()
    # Sending the request for data
    temp1 = res["weather"][0]["description"] # Extracting data from the JSON response
    temp2 = res["main"]["temp"] # Extracting data from the JSON response
    output(f"Temperature is {format(temp2)} degree Celsius \nWeather is {format(temp1)}") # Outputing the actual information regarding weather
```

3.1.4 Receiving News

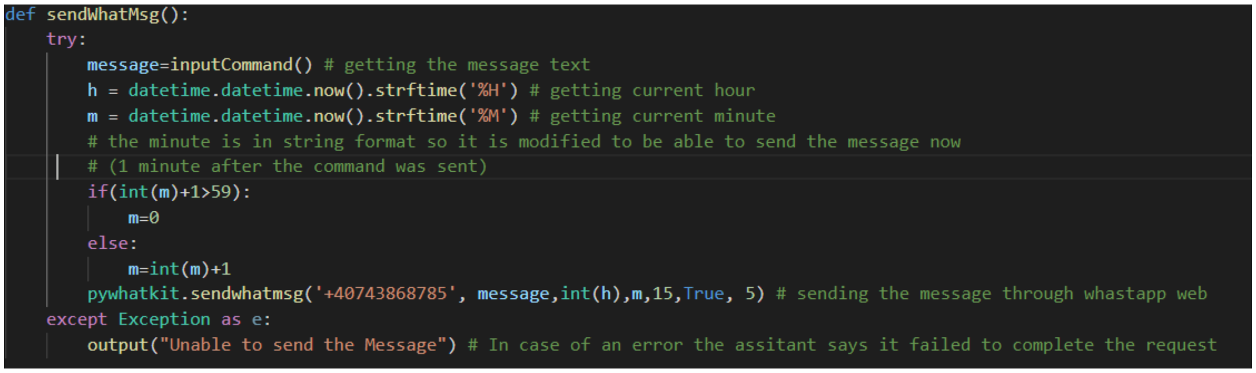
This is done using the NewsAPI[4] for python which is an unofficial Python client library to integrate News API into your Python application without having to make HTTP requests directly. The main use of News API is to search through every article published by over 80,000 news sources and blogs in the last 4 years.



```python
def news():
    newsapi = NewsApiClient(api_key='05fbef0165ce4124aac6080dd1e3ee27') # Initating the News Engine
    output("What topic you need the news about")
    topic = inputCommand() # Getting the topic for the news
    data = newsapi.get_top_headlines(q=topic, language="en", page_size=5) #Getting the most important headlines on that topic
    newsData = data["articles"] # Saving the articles themselves in an auxiliary variable
    for y in newsData:
        output(y["description"]) # Outputting the news themselves
```

3.1.5 Sending WhatsApp Messages

This functionality is built using the python library PyWhatKit[5]. It is a library for scheduling and sending WhatsApp messages with various other functions like playing a video on YouTube, Converting an image to ASCII art, Converting a string to an image with Hand Written Characters.

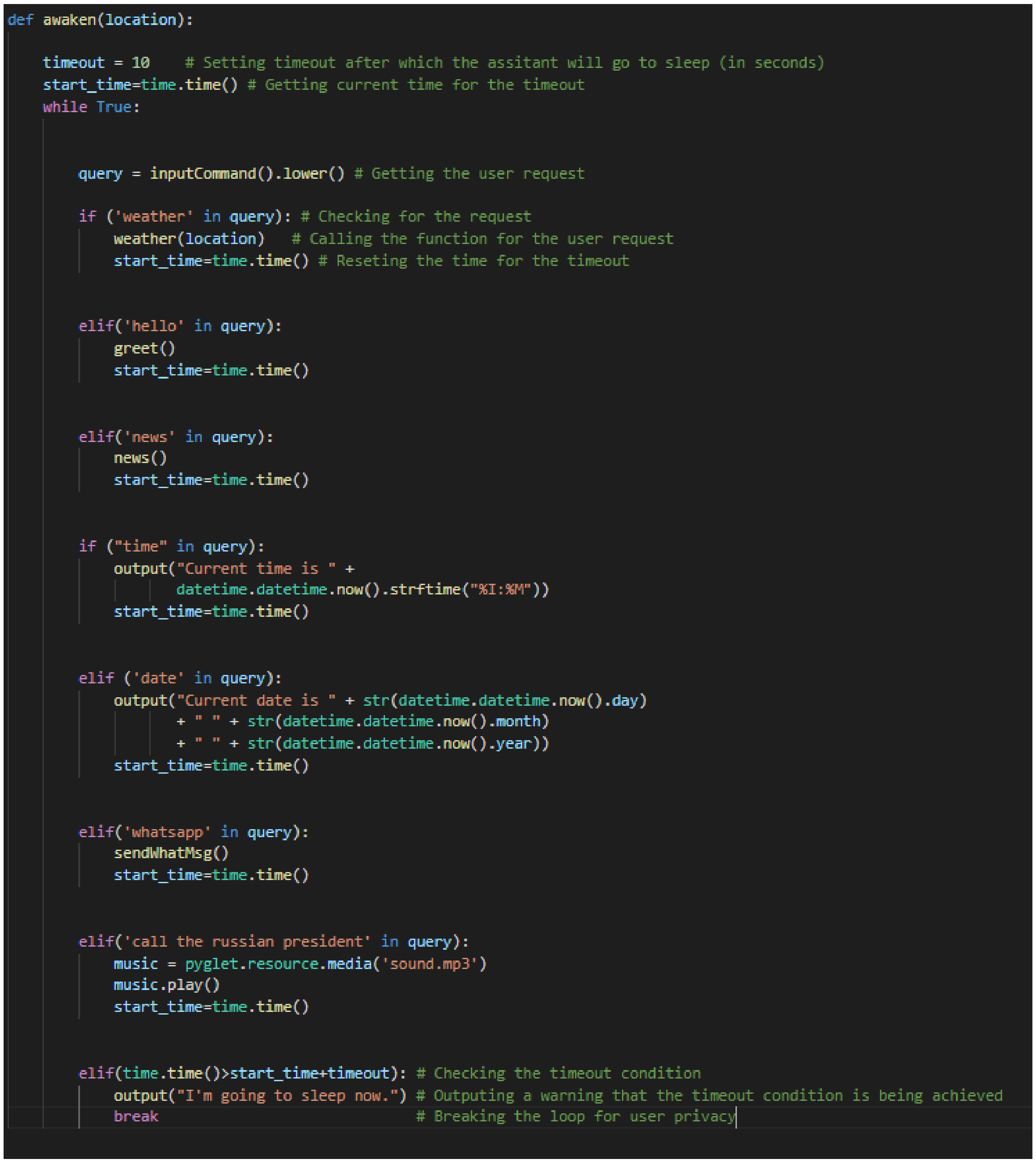


```python
def sendWhatMsg():
    try:
        message=inputCommand() # getting the message text
        h = datetime.datetime.now().strftime('%H') # getting current hour
        m = datetime.datetime.now().strftime('%M') # getting current minute
        # the minute is in string format so it is modified to be able to send the message now
        # (1 minute after the command was sent)
        if(int(m)+1>59):
            m=0
        else:
            m=int(m)+1
        pywhatkit.sendwhatmsg('+40743868785', message,int(h),m,15,True, 5) # sending the message through whastapp web
    except Exception as e:
        output("Unable to send the Message") # In case of an error the assitant says it failed to complete the request
```

3.1.6 The Decision Making

The making of the decision of which functionality to call is done using a simple if-else structure which runs in a while loop with a 10 second timer (which can be very easily modified).

The main loop functions by setting a 10 second timer and starting to listen for user commands. If a command has been recognized it will attempt to fulfill the request and reset the timer. If no command is recognized it will announce the user that it will exit the loop in order to keep privacy.



```python
def awaken(location):

    timeout = 10    # Setting timeout after which the assitant will go to sleep (in seconds)
    start_time=time.time() # Getting current time for the timeout
    while True:


        query = inputCommand().lower() # Getting the user request

        if ('weather' in query): # Checking for the request
            weather(location)   # Calling the function for the user request
            start_time=time.time() # Reseting the time for the timeout


        elif('hello' in query):
            greet()
            start_time=time.time()


        elif('news' in query):
            news()
            start_time=time.time()


        if ("time" in query):
            output("Current time is " +
                    datetime.datetime.now().strftime("%I:%M"))
            start_time=time.time()


        elif ('date' in query):
            output("Current date is " + str(datetime.datetime.now().day)
                    + " " + str(datetime.datetime.now().month)
                    + " " + str(datetime.datetime.now().year))
            start_time=time.time()


        elif('whatsapp' in query):
            sendWhatMsg()
            start_time=time.time()


        elif('call the russian president' in query):
            music = pyglet.resource.media('sound.mp3')
            music.play()
            start_time=time.time()


        elif(time.time()>start_time+timeout): # Checking the timeout condition
            output("I'm going to sleep now.") # Outputing a warning that the timeout condition is being achieved
            break                             # Breaking the loop for user privacy
```

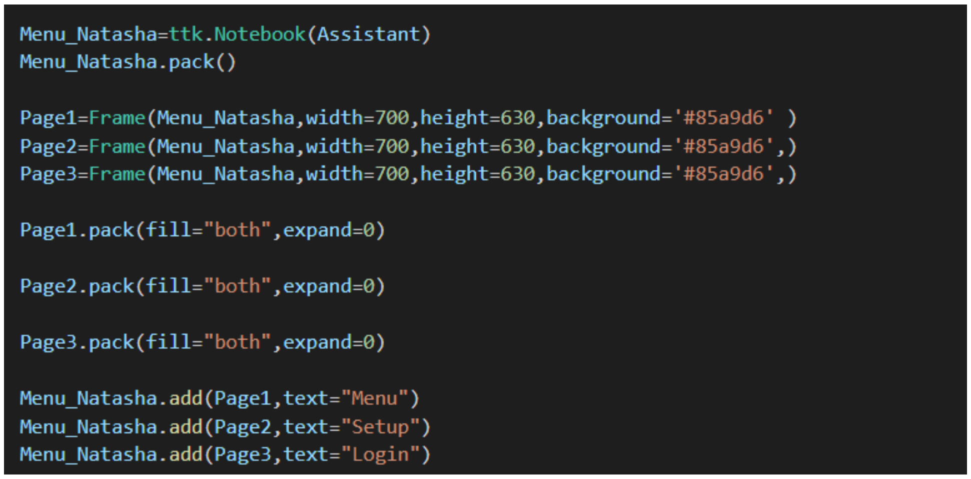
3.2 The User Interface

3.2.1 Introduction

The interface of the assistant was made in python using the Tkinter library in order to provide graphical support without being dependent on operating system used by the user relying only on libraries which will come along with the installing package.

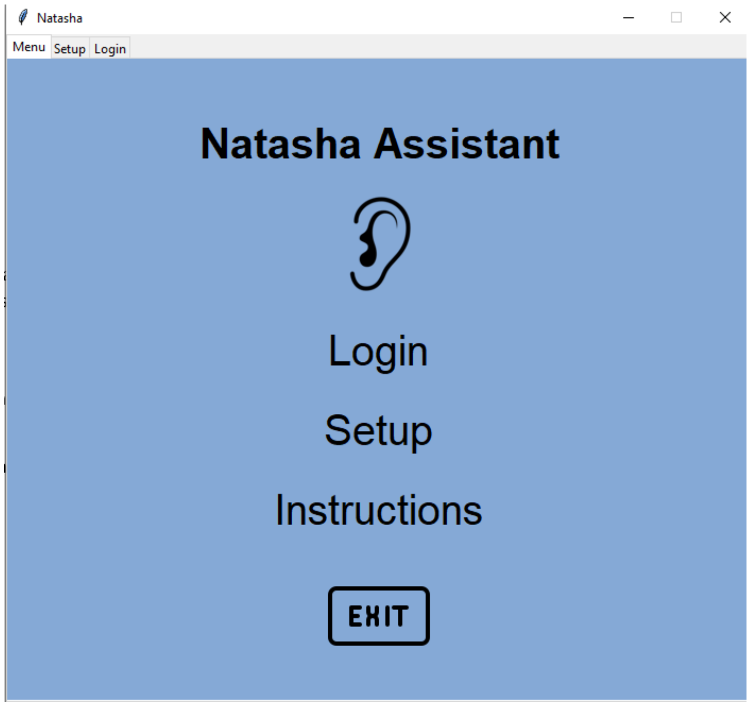
3.2.2 Structure

As for the structure the interface consists of 3 main parts those being the Home Menu, Setup Page and Login Page. For this we are using the “ttk” library[7] for allowing the creation of a “notebook” like structure which allows us to swap between pages.

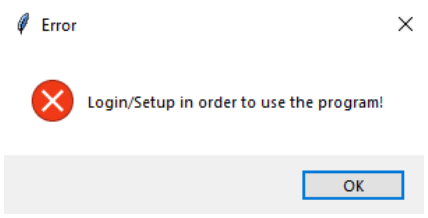


```
Menu_Natasha=ttk.Notebook(Assistant)
Menu_Natasha.pack()

Page1=Frame(Menu_Natasha,width=700,height=630,background='#85a9d6' )
Page2=Frame(Menu_Natasha,width=700,height=630,background='#85a9d6',)
Page3=Frame(Menu_Natasha,width=700,height=630,background='#85a9d6',)

Page1.pack(fill="both",expand=0)

Page2.pack(fill="both",expand=0)

Page3.pack(fill="both",expand=0)

Menu_Natasha.add(Page1,text="Menu")
Menu_Natasha.add(Page2,text="Setup")
Menu_Natasha.add(Page3,text="Login")
```

Once launched the app shows the Home Menu represented by a series of buttons which have different functionalities such as opening the user manual, triggering the assistant, moving through Menu.

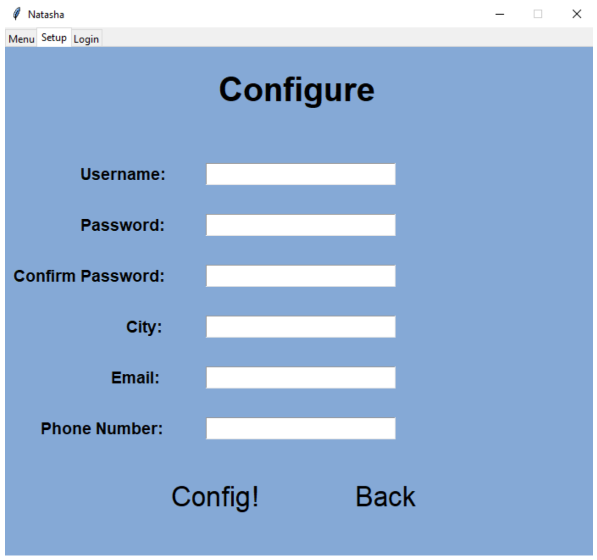


In order to use the "Trigger" button (Ear), the user will have to Setup and Login (or just to Login, once he has created an account).


Error

Login/Setup in order to use the program!

3.2.3 Setup



Configure

Username:

Password:

Confirm Password:

City:

Email:

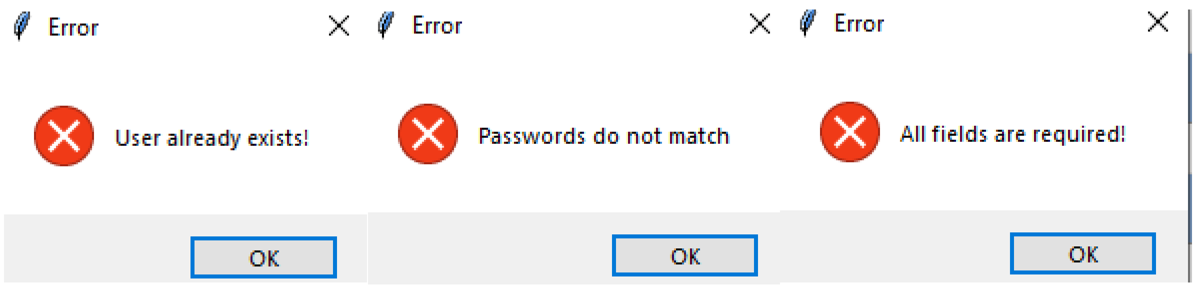
Phone Number:

In order to Configure his account the user will have to provide a series of information needed for some of the assistant functions. For example, the assistant will require a valid city name in order to use functions such as date, time, weather or the phone number for the Whatsapp function.

As for the security there is no risk of data leaks since all the data is stored locally in users' pc as files.

Setup is making a series of checks before creating the new user such as:

- Checking if the user already exists
- Checking if passwords match
- Checking if all fields were completed


Error

User already exists!

Error

Passwords do not match

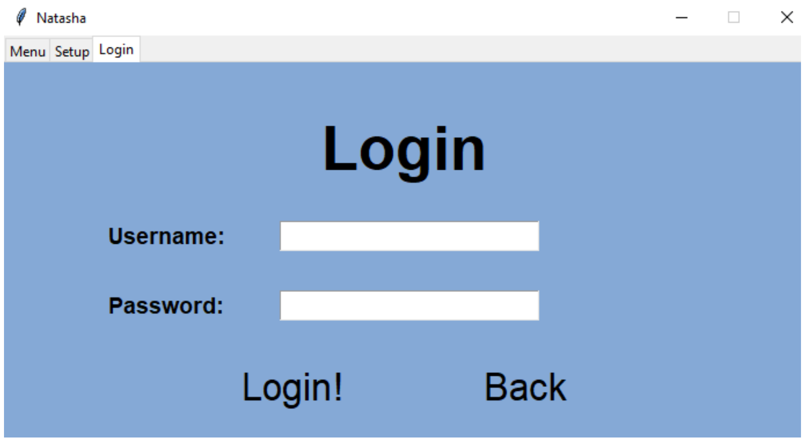
Error

All fields are required!

In case of any of these, pressing the "Ok" button will clear all input fields.

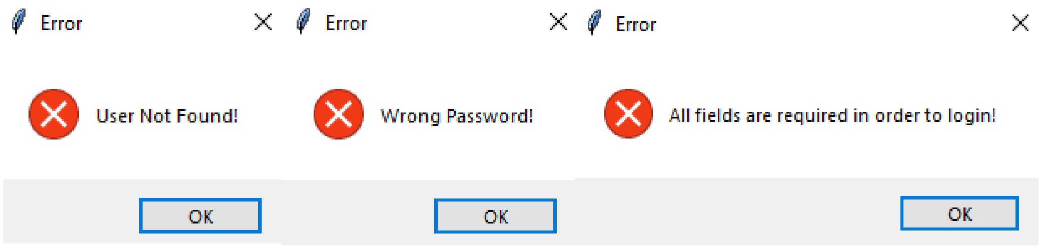
In case of which all fields are completed correctly user will get a proper message and be sent back to the home page. In order to use the assistan , the user will have to login in his account.

3.2.4 Login



Just like the Setup , the login is performing a series of checks such as:

- Checking if the user already exists
- Checking if passwords is correct
- Checking if all fields were completed
-

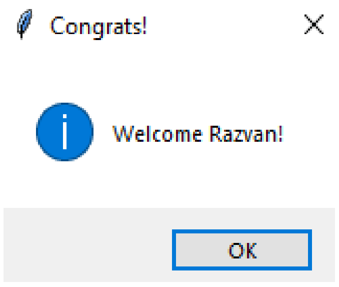


User Not Found!

Wrong Password!

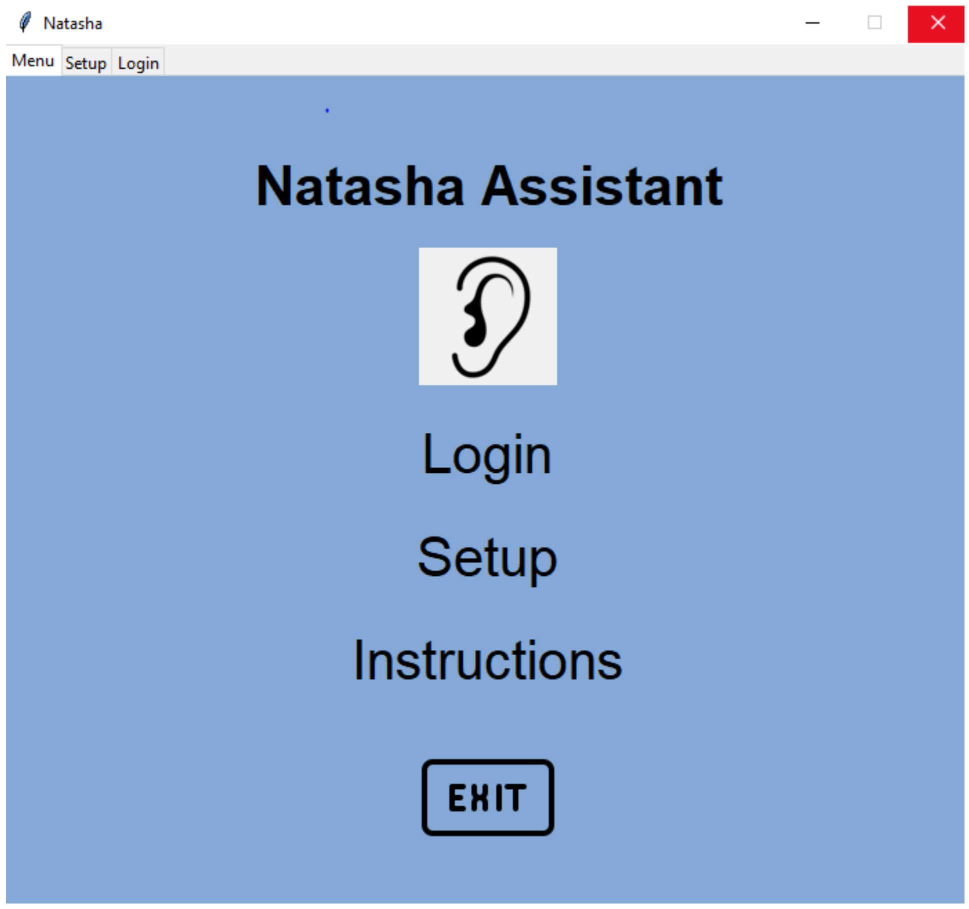
All fields are required in order to login!

In case that both passwords and username match with any of the saved accounts, the user gets a personalized welcome message with his name both vocally and visually and it’s sent back to the menu now being able to use the assistant.



Welcome Razvan!

In order to trigger the assistant user will have to press the “Ear” button which will trigger the listen function making the background of the button white. If the assistant does not get any command in the interval (20 seconds) the assistant will close itself automatically.



In the end we have an “Exit” button which closes the app and the “Instructions” button which gives the user access to user manual.

4. Conclusion

In the end we tried to update the concept of virtual assistant solving the main concern of our user: privacy/security while having a modular/upgradable structure able to fit 21st century user needs.

Results

As for the results, the assistant may have some issues recognizing certain accents or ignoring the background noise but that's normal considering that at this point we are talking about a prototype.

5. References

[1] Python PYTTSX3 - https://pypi.org/project/pyttsx3/

[2] Python Speech Recognition - https://pypi.org/project/SpeechRecognition/

[3] Python Requests - https://docs.python-requests.org/en/latest/

[4] Python News API - https://newsapi.org/docs/client-libraries/python

[5] Python Whatsapp Message Library - https://github.com/Ankit404butfound/PyWhatKit/wiki

[6] Webbrowser https://docs.python.org/3/library/webbrowser.html

[7] Tkinter https://docs.python.org/3/library/tkinter.html

[8] OS https://docs.python.org/3/library/os.html