Even Semester (2022)



**BINUS UNIVERSITY**

**BINUS INTERNATIONAL**

**Assignment Cover Letter**

**(Group Work)**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  | |  | |
| **Student Information**: **Surname** | | | | | **Given Names**  **Adriel**  **Ado**  **Ikhsan**  **Jovan** | | **Student ID Number**  **2201807980**  **2201843506**  **2201807993**  **2201832856** | |
| 1.  2.  3.  4. | | **Soengadi**  **Girindra**  **Maulana**  **Novarian** |  | |
|  |  |
| **Course Code** | **: COMP6340** |  |  | | **Course Name** | | **: Analysis of Algorithms** | |
| **Class** | **: L3BC** |  |  | | **Name of Lecturer(s)** | | **:** Tri Asih Budiono  Yaya Heryadi | |
| **Major** | **: CS** |  |  | |  | |  | |
| **Title of Assignment** | : Audio to Action Compiler | |  |  | |  | |  | |
| **Type of Assignment**  **Submission Pattern** | **: Final Project** |  |  | |  | |  | |
| **Due Date** | **: 31-10-2019** |  |  | | **Submission Date** | | **: 31-10-2019** | |

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

**Plagiarism/Cheating**

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offense which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

**Declaration of Originality**

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

(Name of Student) Signature of Students:

1. Adriel 3. Ikhsan
2. Ado 4. Jovan

**TABLE OF CONTENT**

[**Introduction** 3](#_Toc23430489)

[**Background** 3](#_Toc23430490)

[**Problem Description** 3](#_Toc23430491)

[**Related Work** 3](#_Toc23430492)

[**Implementation** 4](#_Toc23430493)

[**Formal description of problem** 4](#_Toc23430494)

[**Design of Code** 4](#_Toc23430495)

[**Proof of correctness** 6](#_Toc23430496)

[**Complexity analysis:** 7](#_Toc23430497)

[**Evaluation** 8](#_Toc23430498)

[**Theoretical Analysis Of The Compiler** 8](#_Toc23430499)

[**Implementation Details** 9](#_Toc23430500)

[**Results** 10](#_Toc23430501)

[**Discussion** 10](#_Toc23430502)

[**Conclusion and Recommendation** 11](#_Toc23430503)

[**Program Manual** 11](#_Toc23430504)

[**Demo Video Link** 14](#_Toc23430505)

[**GIT Website** 14](#_Toc23430506)

# 

# 

# 

# 

# **Introduction**

## **Background**

Firstly, audio to action has been implemented in many devices such as laptops and smartphones, it works as a voice assistants to enable us to order various actions towards our devices like play music, open an application etc. Hence, the features has the potential to change how we as users to use our devices as things could be done more easily by using audio instead of typing. This makes us think that in the future people could text while driving safely by using audio to command since they would not use their hands. Audio to command also had the beneficiaries that can extend from normal people to people with disabilities in view of the fact that some people do not have the capability of typing texts, pressing buttons, in other words the physical interaction towards devices. From the reasons above, we find it interesting to be able to deconstruct voice to command and try to build it with our understanding of compiler.

## **Problem Description**

The scope of our final project is only in the area of the compiler and parsing, thus we will not try to engage to machine learning in the speech recognition and text to speech considering that the debugging process of machine learning is surprisingly hard on the point of there is an error inside the code. Based on our scope firstly we need to convert our voice to text. From the text, we need to parse it from the computer to understand the string and alter it to be a command that the computer understands. After finished parsing, we execute the command depending on the keyword from our voice by directing them to functions and libraries we use and make.

We also planned to use some libraries as a support of our project such as text to speech for the text replyment from the system or time for setting the delay in bot reply and our audio.

## **Related Work**

There are a lot of related works according to our project such as Siri and Google Assistant. Those audio to action innovation called the voice assistant or chatbot gives people benefits in both leisure and businesses. The primary creation of these sort of innovations can be attributed to convenience. But other than fulfilling our laziness, it also benefits people who have impediments in doing things, and thus need the assistance of a sort of audio-to-command program.

# **Implementation**

## **Formal description of problem**

Based on the scope that we have specified in the problem description, we have to understand the steps on how we can transform the speech to a code and make the computer understand. Thus, we must use parsing method in which we need to implement Lex and Yacc. Lex meaning we have to split the strings we have from the speech, Yacc meaning that we have to put a hierarchy on the result of the Lex and execute the strings based on the hierarchy. In our case, we put the command string that we have specified as the highest hierarchy compared to other strings from the speech. Later after parsing we need to direct the program to execute certain functions for the program to execute the command that the user wanted.

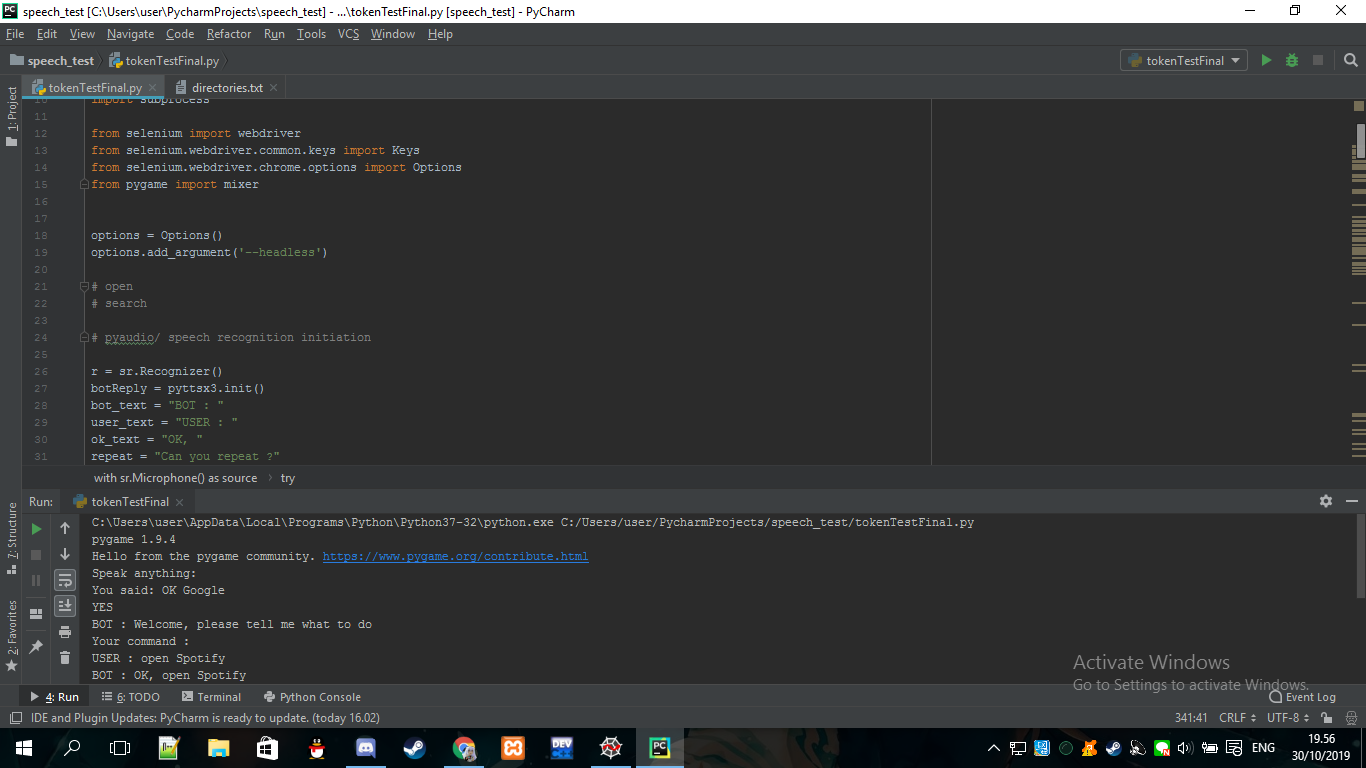
## **Design of Code**

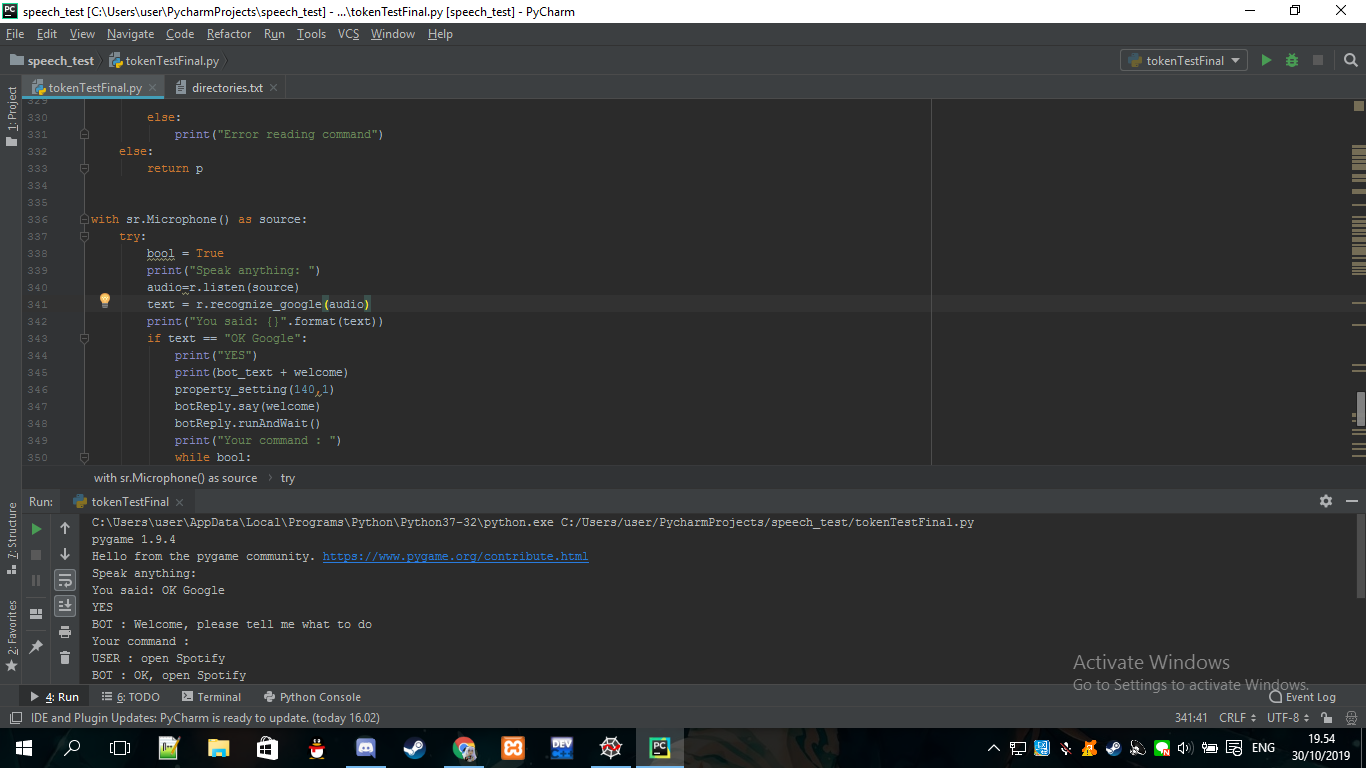
1. **Parsing/Tokenizer**

Parsing is the main part of our code as it is the part that handles the output from the speech to text and parse it so that it can be easily decided on what the action that can be executed by the main. The main hierarchy of the parsing is only the command word that signals the computer which later function would be executed. The parsing is done through a python library in which we would implement Lex and Yacc.

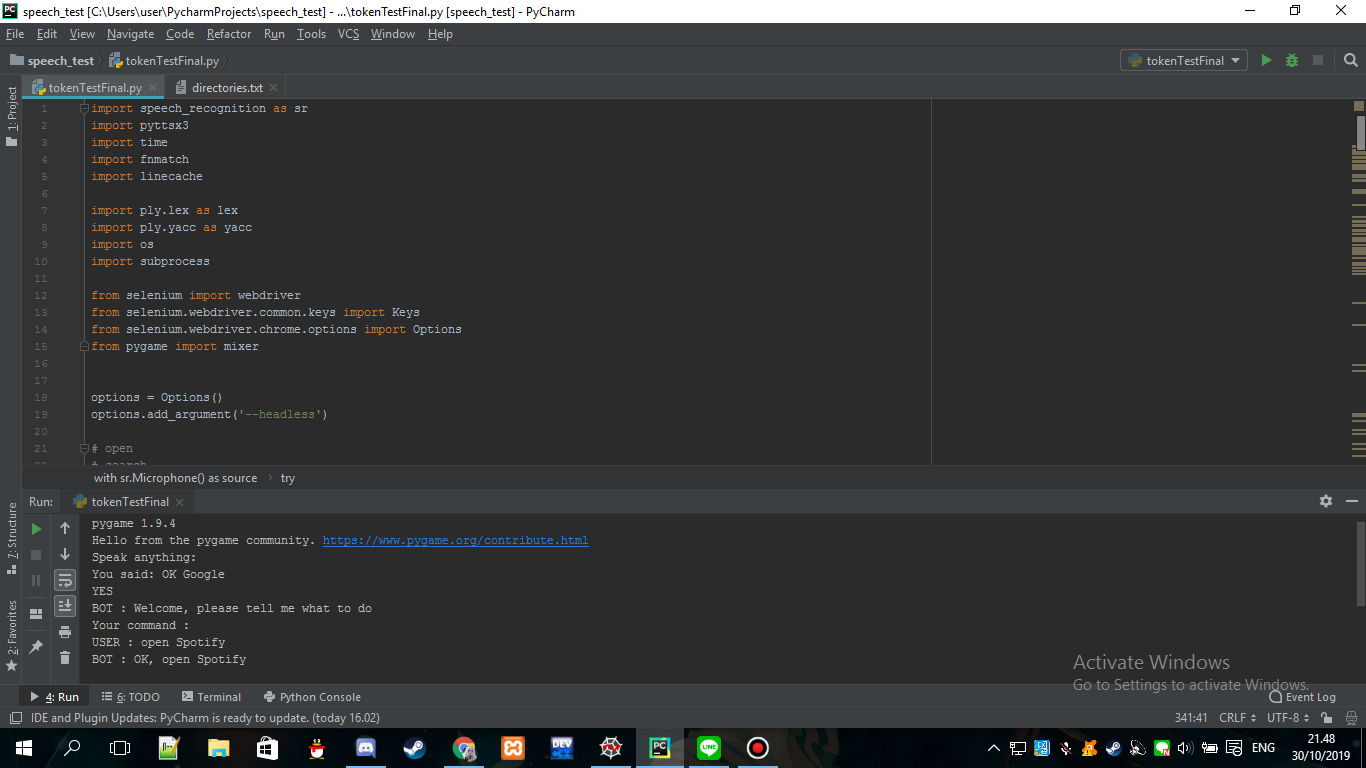
1. **Speech to text**

As it was written in our problem description, we could say that speech to text is our second step to finish our project. We found an API call the speech recognition for audio conversion to text, the way how program converts our audio , there are actually similar speech recognition api such as google-cloud-speech, pocketsphinx, the reason that we chose speech\_recognition was because it was very simple to understand and code implementation is also considerably easy considering combining codes of “text to command” from our parsed tokenizer. The picture below indicates the initiation of it and how it detects voice from microphone.



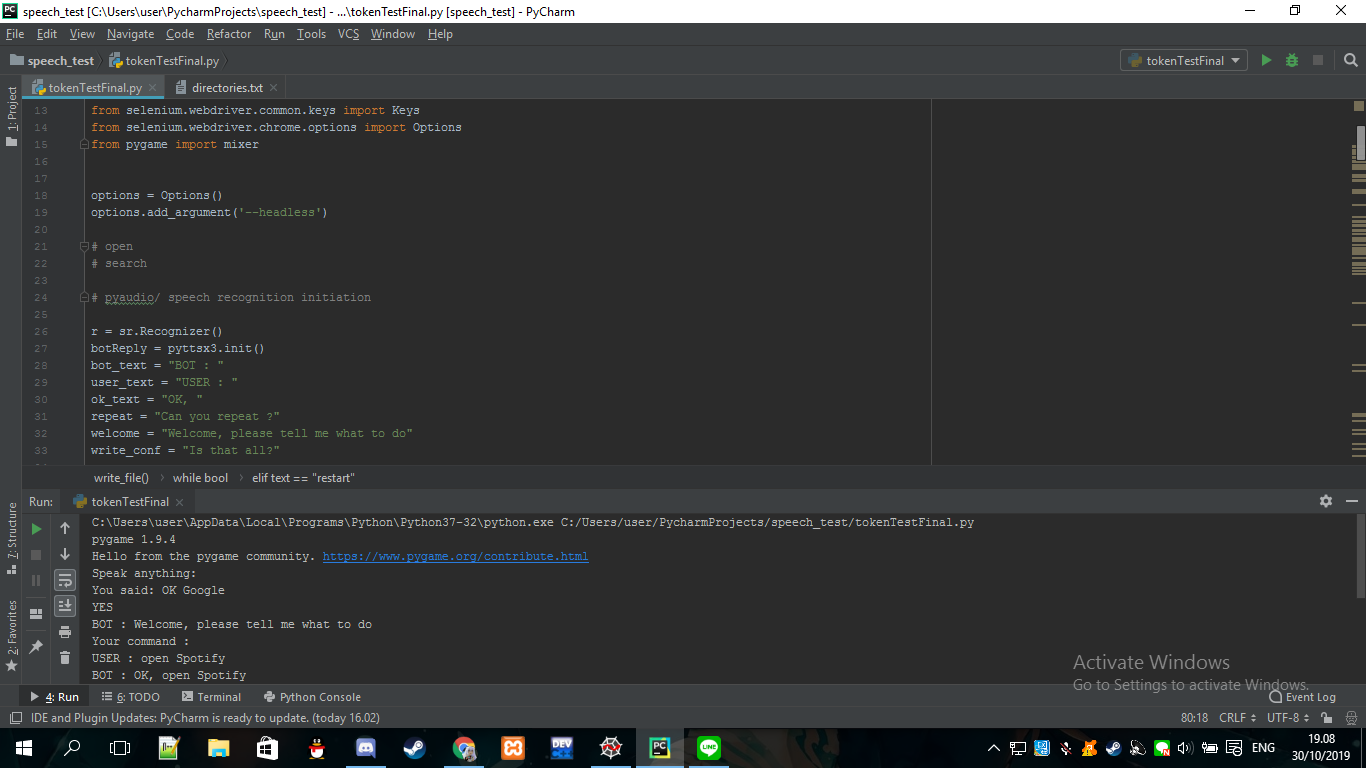


Before testing it, we make sure to install pyaudio in view of the fact that it enables our microphone from our IDE. The picture indicated the terminal result of the speech recognition implementation.

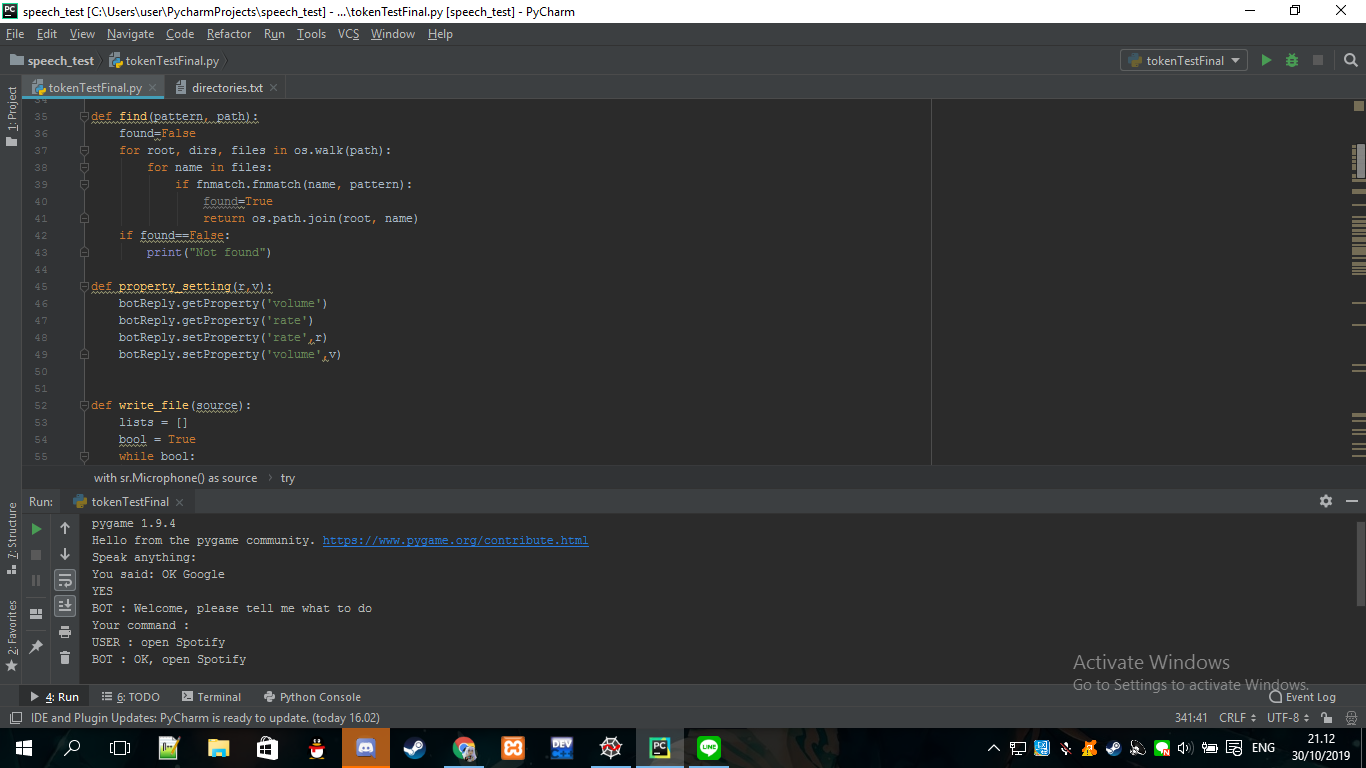


1. **Text to speech**

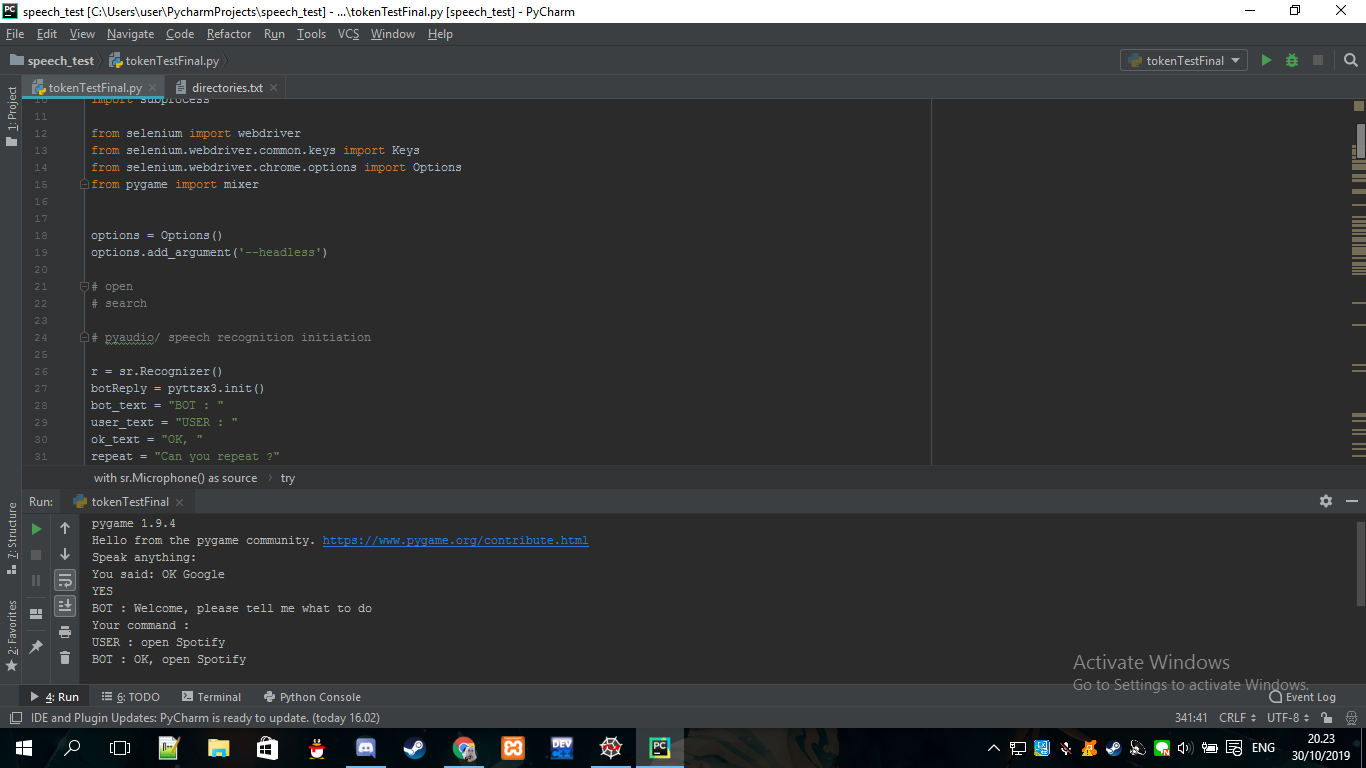
Our group was planning to create a simple interface for the interaction between the program and us, we used an API as a support for our code call pyttsx3. Pyttsx3 stands for python text to speech 3 (python version 3) which converts a text into an audio that is going to be delivered by a bot from the IDE. The picture below indicates the initiation of the module and the command interface as a result from the interaction.



We created a function for property setting, getProperty and setProperty are one of the module inside the text to speech api where you can set volume ( loudness of the speech ) and the rate ( speed of speech ) based on the properties you get.



Here is the output

****

User : Our Audio

BOT : Audio from converted text by pyttsx3

1. **Main (Run)**

The execution of the main relies from the tokenizer as explained in number 1, from the tokenizer the machine will call functions depending on the command string that is picked up by the tokenizer. Moreover, the machine will decide which function should be called to execute what the user wants. Below will be the commands that can be executed by our program alongside its proof of correctness.

## **Proof of correctness**

The table below specified the expected result based on the command from our audio.

|  |  |
| --- | --- |
| **Command** | **Expected Result** |
| Open Application (Chrome, Spotify) | The program will open a recognized application within your system. |
| Play music (music.mp3) | The program will play the music that was inputted, by searching for the music file in the directory provided in the code. |
| Search something (any keyword) | The program will open google chrome and search for the keyword input using google. |
| Write a note | The program will open notepad and writes the input keyword(s) in. |
| Open application and/then search something | The program will open a recognized application within your system. If that application is a web browser, then that browser will then search the input keyword. |
| Search something and/then open application | The program will open google chrome and search for the keyword input, and then immediately open an app within your system. |
| Stop music (music.mp3) | The program will terminate the current music file that is playing. The command will not result in anything if there are no music currently being played. |
| Open document | The program will search for a document in your system that matches the keyword input and then promptly opens it. |

**Efficiency of expected Result relies on:**

**Good Internet access:**

Some API’s will need a stable internet connection for it to work properly. No internet connection at all will result in a compiler error, and an unstable or slow connection will result in the program compiling much slower. This is because the API is provided by google, and accessing google will require you to be connected to the internet.

**Good microphone:**

The program will also need a decent microphone system, as an sub-par microphone will result in the program not fully recognizing the audio input, either misinterpreting and mistaking the word you said as another word entirely, or not able to find the word at all in its library and will require you to repeat the command.

## **Complexity analysis:**

In this project, we implemented the PLY package in Python. The PLY package is a pure-Python implementation of the compiler construction tools lex and yacc. The package consists of lex.py and yacc.py, where the yacc.py will invoke grammar rules on to the tokens that lex.py returned. The yacc.py includes plenty features such as extensive error checking, grammar validation, support for empty productions, error tokens, and ambiguity resolution via precedence rules. Furthermore, the parsing done by the PLY package is based on the LR-parsing which is fast, memory efficient, and better suited to large grammars. Hence, the PLY package has allowed the parsing done in this project to be fast and memory efficient.

Aside from parsing, our project also has a feature where it opens an application by accessing the directory of the application from a .txt file inside the project. The .txt file consists of the directories of applications that the user frequently used, therefore not every application directory is inside this .txt file. Because not every application is inside the .txt file, if the user commands the program to open an application that is not inside the file, the program will access every folder in the computer’s drives to find the directory of the application. Hence, we could say the complexity of this algorithm is O(N\*log N). The reason behind this complexity is because if the application’s folder directory is not that far off from the computer’s drives (e.g. Windows(C:) > Program Files(x86) > Google > Chrome), the computing time won’t take a long time. While if the application’s folder directory is far off from the computer’s drives, it will take quite a long time to find the directory of the file. Hence, we append the directory of the application after the program searches for it and append it to the .txt file to avoid more time complexity.

# **Evaluation**

## **Theoretical Analysis Of The Compiler**

**Lexical Analysis:**

To start the lexical analysis phase, the compiler scans the input from the conversion of the user’s audio to text. The pyaudio package that we used for the program will return a string from the user’s audio input. The string will be scanned and then breaks up into the smallest possible parts (known as lexemes). Then the stream of lexemes will be converted into a stream of tokens, which are regular expressions that are understood and analyzed by the lexical analyzer (we implemented *lex.py* from PLY for our program).

**Syntax Analysis:**

Next, after the lexical analysis is complete and we have received the tokens that are converted from the lexemes, it is now time for the syntax analysis phase. In this project, since we are implementing the PLY package (*yacc.py* for the parsing phase) in python, the syntax analyzer or the parser will take the token one by one to construct an Abstract Syntax Tree (AST). Then, the will be input checked to see whether it is in the desired format or not, and a syntax error can be detected at this level if the input is not accordance with the grammar. After the AST is constructed, it will proceed to the next phase which is the semantic analyzer where the AST will be verified. The semantic analyzer will also do type checking, label checking, and flow control checking.

**Intermediate Code Generator:**

After the AST is verified, this phase will turn the AST into an independent intermediate code which will later be turned into the targeted code that can be executed by the machine. In our project, we want to transform the AST that was created from the parser into an executable code that the program will execute based on the input. For example, if the input is “open spotify”, the input will then be converted into an AST which then later be turned into the targeted code so that the program could open spotify for the user.

## **Implementation Details**

API Implemented:

* Selenium
* PLY
* Pygame
* Pyaudio
* Pyttsx3
* Speech Recognition
* fnmatch
* Linecache
* time

**Selenium:**

We implemented selenium in order for the program to open Google Chrome and search for the keywords that the user input through audio. The selenium API is actually able to open both Google Chrome and Firefox by downloading their driver, but for this project we wanted to use Google Chrome since it is more popular and faster.

**PLY:**

We implemented PLY for the parsing part of our program. As mentioned before, PLY allows us to execute the parsing quickly and easier since the parsing done by the PLY package is based on the LR-parsing which is fast, memory efficient, and better suited to large grammars.

**Speech Recognition:**

We implemented speech recognition google API to convert the speech recognized by the computer’s microphone and convert it towards text. This API requires an online connection to function as the conversion from speech to text is done over the net. It is implemented mainly in every part of our code that requires an input of either the command from the user or the continuation of a command such that in our write a note function.

**Fnmatch:**

We implemented this to make it easier for the program to search through the files and directories with regards to music and applications. This library is implemented in the making of the find function in our code.

**Pygame:**

We implemented this to play and stop the music. We used the mixer function inside the pygame library to play and stop the music in which the stop function can be called even when the music is playing. Also, pygame can play both .wav and .mp3 file.

**Pyaudio:**

Pyaudio is not directly imported into our program. However, for the speech recognition to work we need pyaudio imported inside the speech recognition. Pyaudio is used to find the microphone in the laptop and constantly listen towards input from the microphone.

**Time:**

The time module if for creating a delay between the bot ( text to speech) and when user is going to execute their command in voice.

**Pyttsx3:**

Bot reply by the system from text to speech. This is implemented to be used as an easier error recognition compared to just read through error message.

**Linecache:**

Linecache module is just to access a line from a .txt file. In our case, we want to access the lines in a .txt file where the directories of an application is stored.

## **Results**

The results of our program would be the compiling of the audio input which will be processed and translated into a command that we have implemented in our program, listed on the table above. From the user’s voice, the program should be able to execute actions to the user’s wish considering the list of commands is from what we have stated previously. The efficiency of the execution of the code varies depending on the internet connection, the complexity of the commands and the pronunciation of the user in which are well beyond our scope of the project we are making.

# **Discussion**

From the results of our evaluation of the program, we were satisfied that the program could be able to do useful things that could assist our daily lives. We say that it is useful for our daily lives because our program allows the user to write a note, open application, and search for something in Google by audio. With being able to write a note through the user’s audio alone, it provides convenience to the user since he could be doing something else in the meantime while writing this note by audio.

However, there are some things that we did not expect and could not avoid, which is finding the directory of the applications in order to open them. As mentioned before, we have created a .txt file that consists of the directory of the applications that the user frequently opened. However, when the user wishes to open an application that does not exist in the .txt file yet, the program will run through every folder on the computer to find its directory and then append it to the .txt file, which could take a long time (~10 to 20 seconds).

Another thing that is considered as a liability for our program usage is the heavy - dependant internet API for the speech recognition. The speech recognition API that we used is heavily - dependant on the internet connection, there is only one API that we found that required no internet connection but in the view of the fact that machine learning is required in order to use the API, hence we did not use it as it exceeds the scope of our project.

Besides the application directory issue, we think that our expectations were met with the results shown in our program.

# **Conclusion and Recommendation**

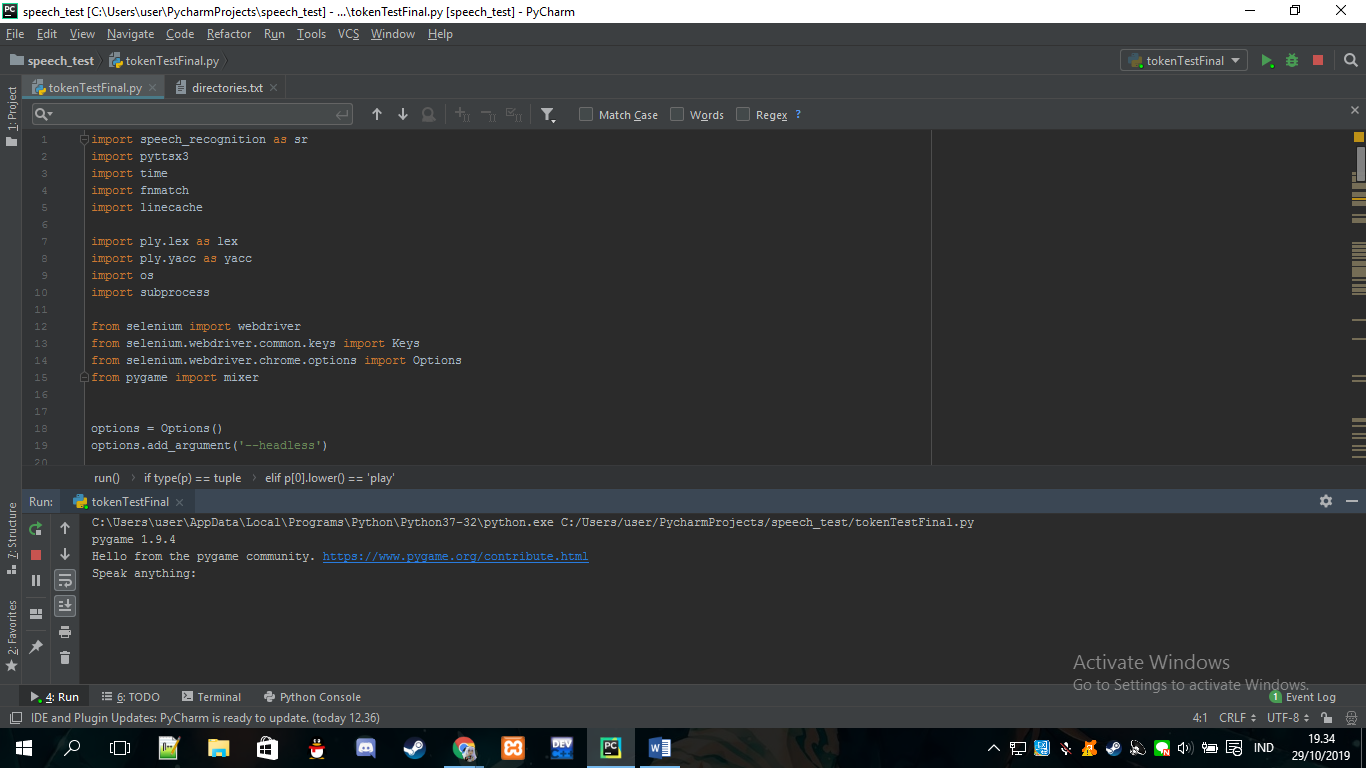
To conclude, our group has successfully created a working audio-to-command program that fulfills our original goals. The commands that we have implemented has been explained thoroughly in the table above. The program also relies heavily on the library provided by google, as creating our own library will not be possible given the time frame that we were given and also it far exceeds the materials that we studied as it would require proper understanding on machine learning.

# **Program Manual**

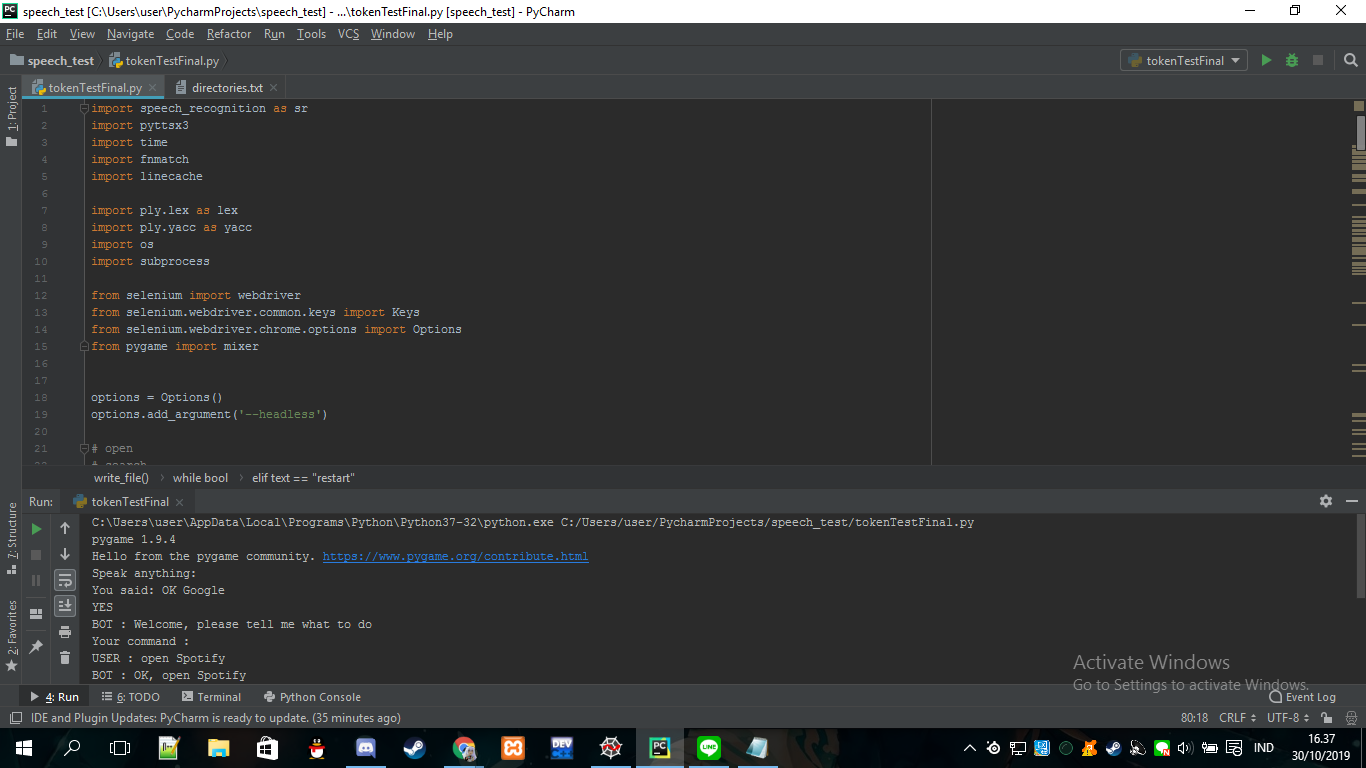
1. Open your pycharm.



1. Run your program.



1. The permission of using the microphone has been activated,speech will be recognized by the API. Before you can access the “project”, make sure to say “OK Google” to gain permission for the next loop, or else the program will be terminated.



1. The permission of using the microphone has been activated for the second time, speech will be recognized by the API, speak any commands to tell your program what to do.

Here are the list of commands that you could do:

1. open *application* (e.g. open spotify)

2. search *keyword* (e.g. search binus international)

3. play *music name* (e.g. play safe and sound or play human)

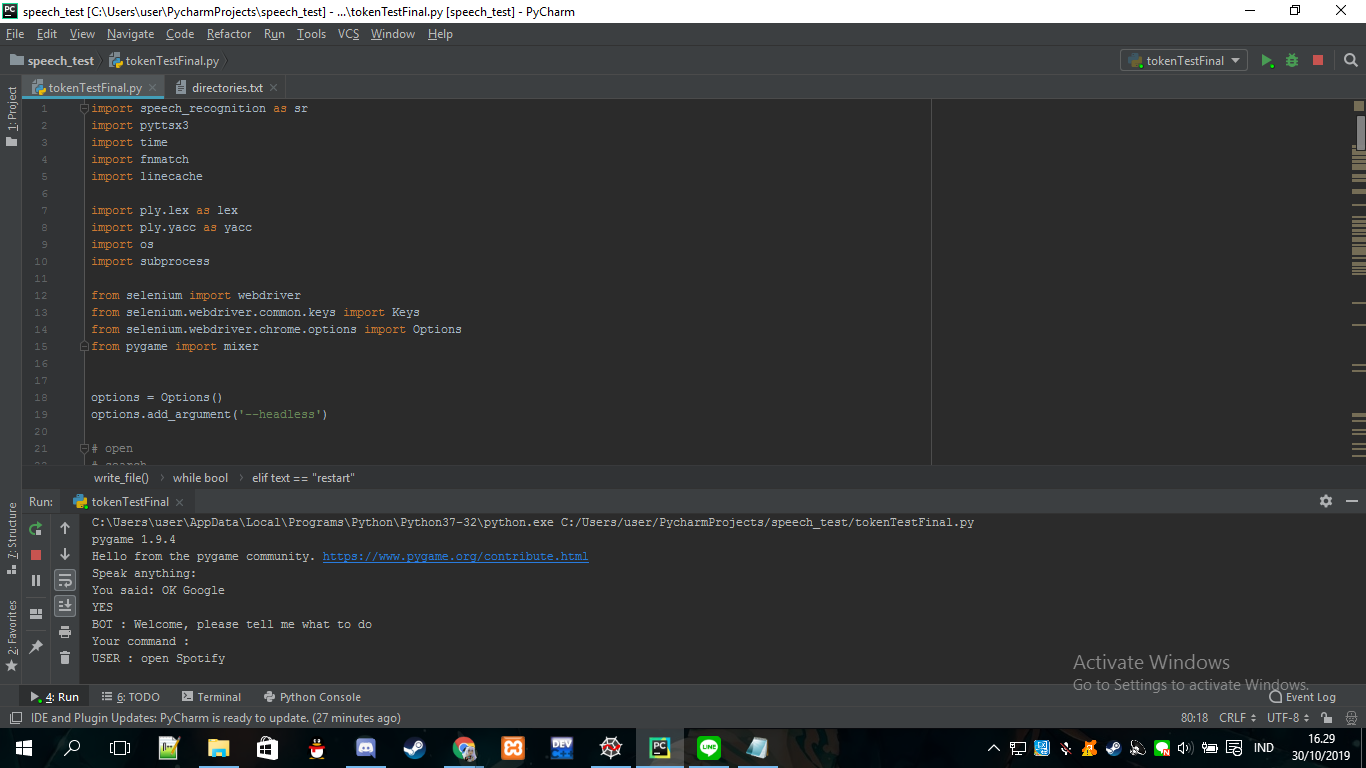
4. open *application* and search *keyword (*e.g. open chrome and search binus)

5. search *keyword* and open *application* then open *application.*

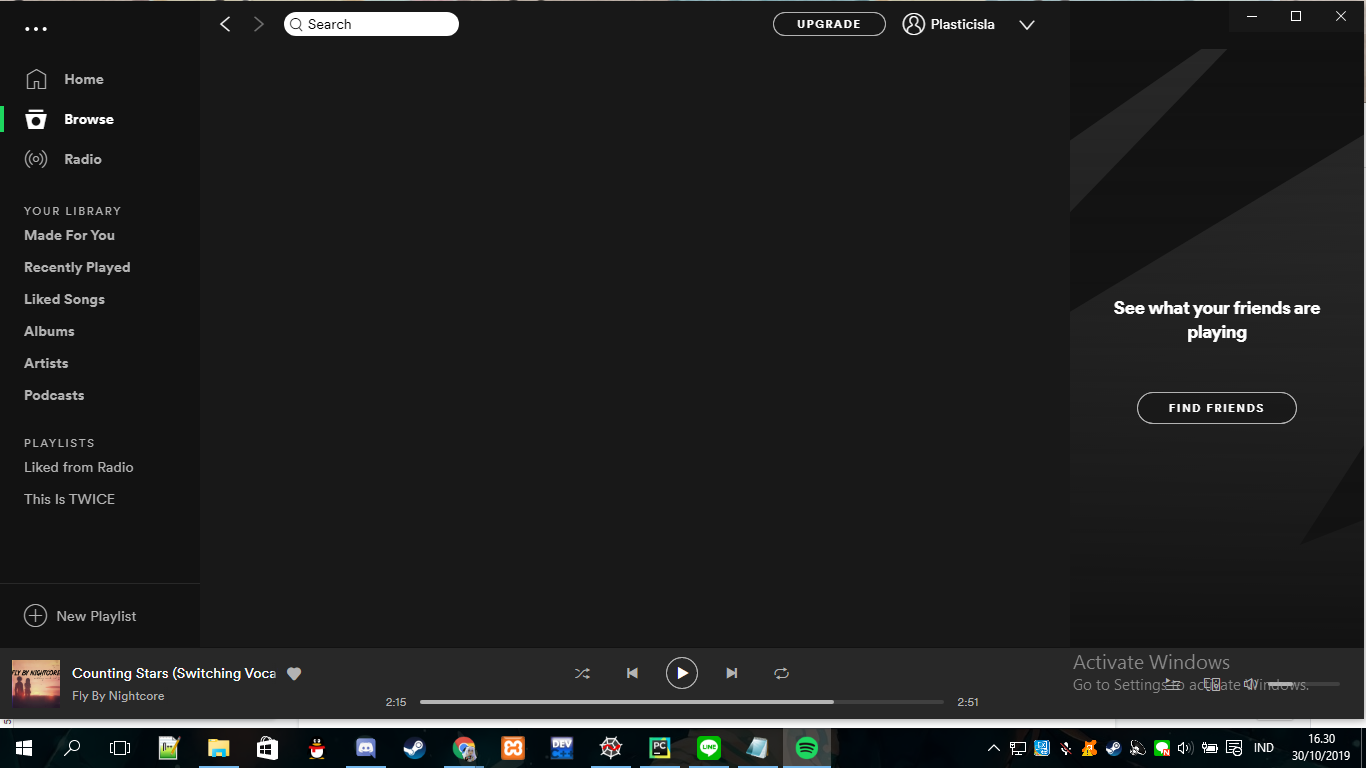
(e.g search binus and open chrome and open spotify)

6. write a note

7. stop *music name* (e.g. stop safe and sound)*.*



1. And there you go, your command will be executed. You can try to do some other steps which was written in the table in proof of correctness, but make sure some commands requires a specific directories setted in the code before compiler such as “play music”.



# 

# 

# **Demo Video Link**

The demonstration video will be located in our final project zip file.

# **GIT Website**

<https://github.com/jovannovarian1117/SpeechToAction_BinusFinalProject_AOA>