Speech recognition:

The application uses python’s speech recognition module, using their support on the Google API. A Microphone object is created in order to record user’s voice input, which uses a default rate of samples per second that are grouped in multiple audio chunks. Then, a Recognizer object is used to turn the audio input into spoken words by sending the audio chunks to the API and receiving the best guess of what was said.

Background listening:

The user can input a voice commands to the application only while holding down the “RCtrl” key. The purpose of this functionality is to stop the program from listening in the background when the user does not wish to send a new command and also to make it easy for it to separate commands from one another. This is achieved using the “pynput.keyboard” package inside the “pynput” module, that has a built in Listener which can pause the program until the user presses a certain key and also an identifier for which key has been pressed. The Listener is a thread which can detect through calling the set “on\_press” and “on\_release” functions, whether a key has been pressed or released and how should the program behave in these situations. A global variable will make sure that the microphone will be recorded only while pressing a certain key. Using speech recognition, user’s microphone will be listened to in the background while he is holding down the “RCtrl” key, recording which will stop and will call on the recognizer when the user releases the key, resulting in the output text that was said during the listening.

The “Voice” class:

A class named “Voice” will handle the vocal input from the user. Upon initialization, an object of this class will allocate memory for a “command” variable that will store the result of speech recognizing, an “executed” variable that will determine whether the command has been recognized and has started or not its execution and a “stop\_listening” variable that will identify the start and the end of the background listening. The “listen” function will initialize a Recognizer and a Microphone and will start the background listening, having the “recognize” function as a callback for when the listening stops. The “stop” function will stop the background listening, while the “recognize” function will set the “command” variable to the text result obtained from the API.

The keyboard listener:

Upon initialization, the Listener will have set its “on\_press” and “on\_release” functions and start a thread which will pause the program until the user presses a key. When that happens, the “on\_press” function is called, which checks whether the pressed key is the required one (“RCtrl”), sets a global variable “pressed” to let the program know that the correct key has been pressed and calls the “listen” function inside the “Voice” object to start listening in the background. The “on\_release” function will be called when the user releases the pressed key and will check using the “pressed” variable if the required key has been released. It will pause the program for a little in order to receive the last bits of the audio input and then call the “stop” function of the “Voice” object to stop the background listening and compute the text result.

Application flow:

On startup, the application will initialize a Browser, Voice and Controller object and start an infinite loop which will initialize a Listener object at each iteration in order to wait for the user to input a command. After the user does so, a variable will be used in order to determine that the command just given by the user has been recognized, started its execution and should not be executed again until the user inputs another command. Then, the recognized command will be compared with the commands supported by the application using Levenshtein’s ratio in order to bypass a certain degree of error between what the user said and what the speech recognizer has guessed and to determine which command should be performed, if any. If a command has been identified, it will be separated in two parts, one that represents the identifiers for that command and another that is the user’s input to that command (if the command requires any). The Browser created object will then call the function which handles the command with the additional input if necessary and after the command has been performed, the application will continue looping and request a new command from the user. The program will stop when the user inputs the “exit” command, which will close the browser and exit the main loop.