Project Report/Documentation

GUI based Audio Stream Player

A GUI based Audio Player that streams audio from an URI developed in Python

Group 3:

Prabhjit Kumar Dutta, Nishant Kumar, Govind Jaishi, Rohit Singh, Gargi Paul

Industry Guide:

Somnath Dutta

Project Link:

https://github.com/PrabhjitDutta/URL_Audio_Player

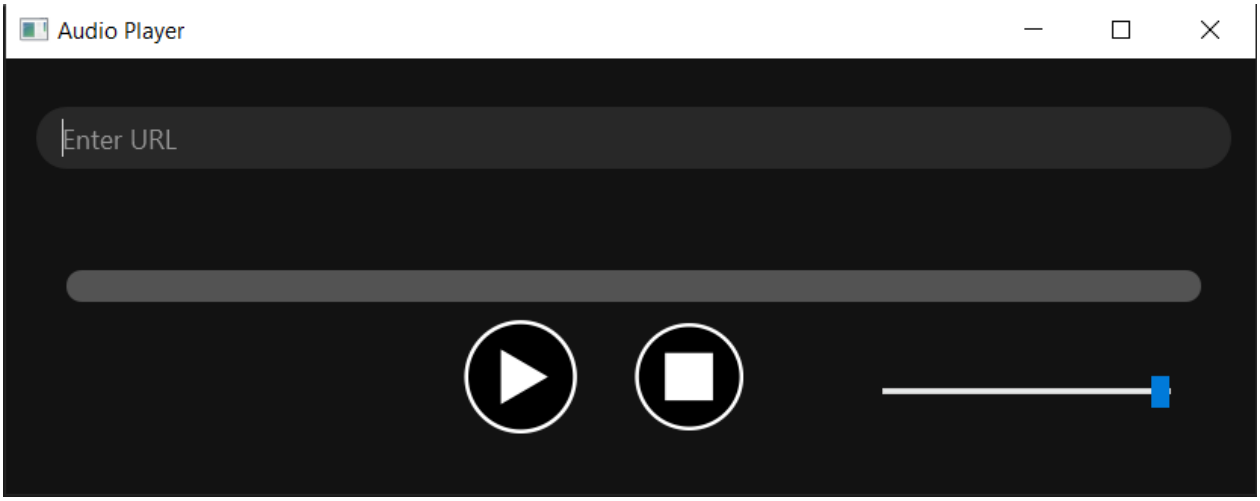
Abstract

The objective of the Project was to develop a GUI based Audio Player that streams audio from an URI in Python.

Using the Python Programming Language, we have developed a GUI based Audio Player which can play audio from an URL. The Player accepts an URL in an address bar and plays the Audio from the URL. The GUI for the application was developed in *PyQt* which is a Python binding for *QT*. The application includes a play/pause button and a stop button. The Application also includes a progress bar to check the progress of the Audio Stream and also a Volume control slider to control the application volume.

A windows frozen binary executable file was built in the Project which allows any user to execute the application without installing any dependencies. While a 'setup.py' and a 'setup.bat' file was developed to allow any user who want to execute the python file to download any dependencies required. After installing the dependencies, the user can execute the Python file.

The Audio Player can play wave and mp3 files.



Prerequisites

Note: you don't require to install any dependencies to use the windows binary executable the dependencies are only required to execute the Python file.

- Python 3
- *PyQt5*
- *pygame*
- *requests*
- *mutagen*

Installation

Any dependencies mentioned in the prerequisites section other than python 3 can be installed directly by executing the *'setup.bat'* file. You will need a working internet connection for the setup to proceed. Again, you don't require to install any dependencies to run the windows binary executable file so if to want to run the application from the file you can skip this step.

Modules Used

- *PyQt5*: To develop the GUI for the application.
- *pygame*: To manage the sound aspects of the application.
- *requests*: To make HTTP GET requests to the Audio URL.
- *io*: the convert the binary data received from the GET request into a BytesIO object.
- *threading*: to implement multithreading in the program.
- *time*: to implement sleep functions in the program.
- *mutagen*: to get the required audio metadata from the mp3 files.
- *wave*: to get the required audio metadata from the wave files.
- *contextlib*: for some Utilities for the with-statement context manager.
- *setuptools*: to implement the setup file.

For more information on the modules check out their documentation.

Installation files implementation

The Installation files were implemented using two files namely the *'setup.py'* file and the *'setup.bat'* file. The *'setup.py'* file was developed using the *'setuptools'* module which installs all the dependencies on your pc. On the other hand, the *'setup.bat'* file runs the pip command on the *'setup.py'* file and starts the installation.

Note: In order to make the setup file work you have to have to pip command assigned to your environment variable PATH. You will require a working internet connection for the installation to proceed.

The Program

The Program is implemented in 1 Parent Thread and 3 Child Threads using the threading module. This division into multiple threads is done in order for the GUI and the audio, the progress bar and the volume slider to work simultaneously and seamlessly.

The Parent Thread

The Parent Thread handles the execution of the GUI and the execution of the Child Threads. The parent thread also handles some global variables which help in communication and message passing between the various child threads. The global variables also act as flags for various events.

The GUI

The GUI was developed using the PyQt5 module which is a Python binding for QT5. The GUI is running on the parent thread. The GUI includes:

- Address bar: implemented using *QLineEdit* class.
- Progress bar: implemented using *QProgressBar* class.
- Play/Pause and Stop buttons: implemented using *QPushButton* class.
- Volume Control Slider: implemented using *QSlider* class

Some elements of the GUI are modified using CSS. The UI is a dark themed UI aimed to be cool and easy on the eyes. The GUI also has a closing *'x'* button which stops the execution of the entire program.

The Click Events for the buttons are handled by the *'click_play()'* and *'click_stop()'* functions.

click_start()

The *'click_start()'* function is called by the GUI to handle the play/pause button. The functions start the *'PlayThread'* child threads and also manages and changes some of the flags to send signals to the child threads. Specifically, it sends signals to other threads telling them whether the Audio is paused or playing.

click_stop()

The *'click_stop()'* function is called by the GUI to handle the stop button. It sends a signal to the child threads to tell them the playing has stopped.

The *'PlayThread'* Child Thread

The *'PlayThread'* Child Thread handles the Playing of the Audio.

- First it accepts an URL from the address bar.
- And makes a HTTP GET request to the URL using the *requests* module while checking if the URL is valid.
- If the URL is valid, we get a HTTP response or if it's not valid it asks for a valid URL from the user.
- Then it gets the byte data from the response and converts its into a *BytesIO* object using the *BytesIO* function from the *io* module.
- Then it checks for metadata like audio length and sample rate using the *wave* or the *mutagen* module depending on the audio format. It also checks whether the audio is an mp3 or wave file and if it's not asks the user for a valid URL.
- Then it initializes the audio player with the sample audio rate using the *'pygame.mixer.init()'* function and moves on to load the player and start playing.

- The *'PlayThread'* Child Thread also control the pause/play and stopping of the audio tracks. This is implemented using signals received from the play/pause and stopped button events of the Parent Thread.
- It also sends the audio length and the audio status to the *'Progress_Thread'* which handles the Progress Bar.

The *'Progress_Thread'* Child Thread

The *'Progress_Thread'* Child Thread handles the functioning of the GUI element, the Progress Bar.

- It receives the audio length and audio status from the *'PlayThread'* Child Thread. It also receives player status signals like pause, play, and stop from the main thread.
- From the audio length and status, it calculated the audio progress and updates it in the Progress Bar.
- It resets the Progress Bar if the Audio is stopped.

The *'Volume_Rocker'* Child Thread

The *'Volume_Rocker'* Child Thread handles the functioning of the GUI element, the Volume Slider.

- This thread checks the Volume Slider element value and changes the volume of the application using the function *'pygame.mixer.music.set_volume()'*.

The *'close'* Function

The *'close'* function is present in the parent thread and is called when the *'x'* button is clicked. It sends signals to all the threads asking them to stop.