# Faculty Of Computing and Artificial Intelligence

# Cairo University

## Object Oriented Programming

## Assignment 2 – HearMEOut Audio Player

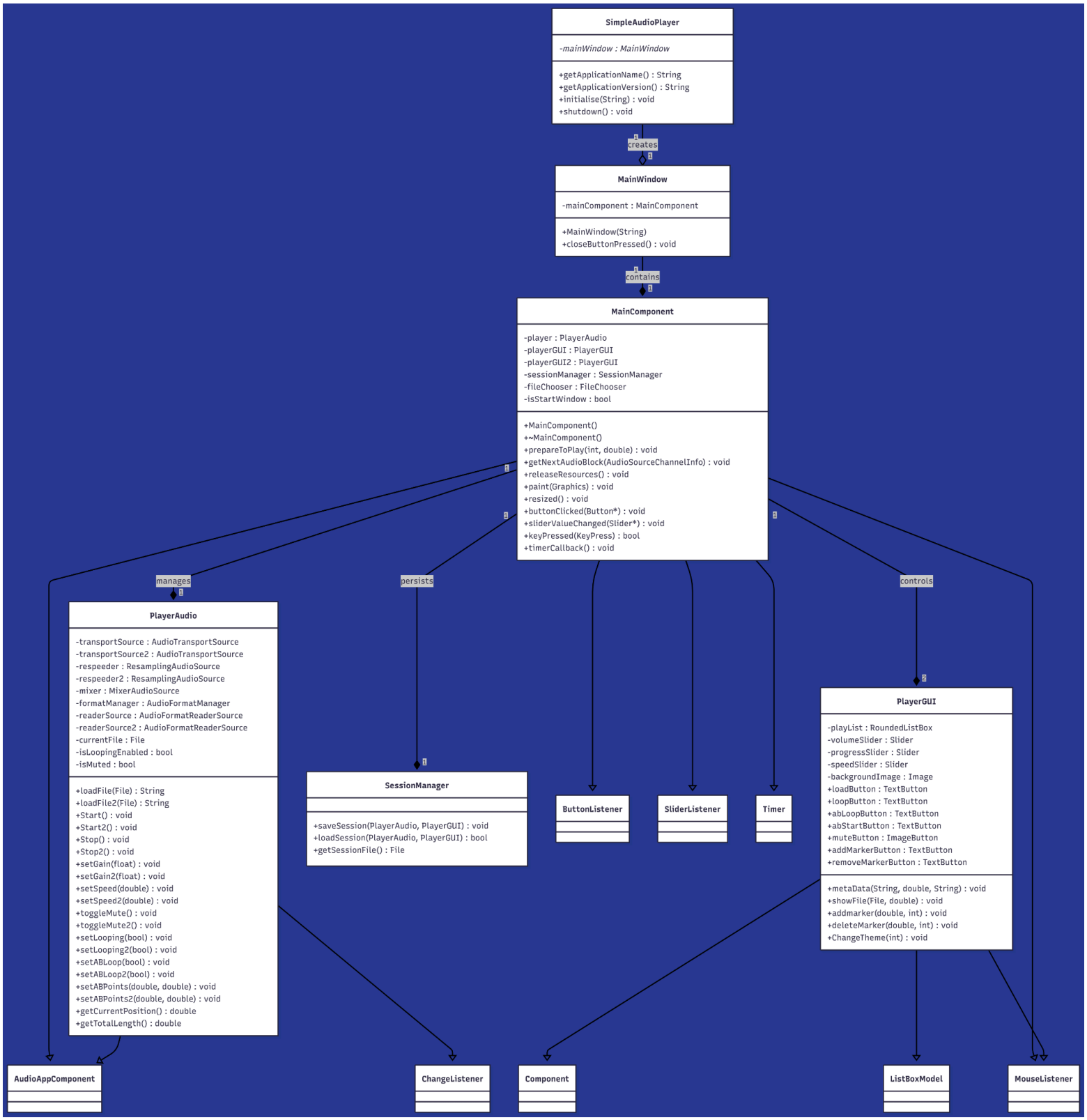
## Submitted to: *Dr. Mohammed Elramly*

**The project link on GitHub:** https://github.com/birdbox219/HearMEout.git

**Participants:**

| Name | ID | S | Work | Emails | Extra tasks |
|---|---|---|---|---|---|
| Mahmoud Elsayed musa | 20240548 | 17 | Task 4, 7, 10 | 20240548@stud.fci-cu.edu.eg | Task 13 ,Theme feature , UI Improvement and Keyboard Shortcuts |
| Amr Atif Abd El-Kareem | 20240398 | 17 | Task 3, 6, 9 | 20240398@stud.fci-cu.edu.eg | Task 12 ,Task 14 and Shuffle play feature |
| Alaa Mamdouh Hamed | 20240109 | 17 | Task 2, 5, 8 | 20240109@stud.fci-cu.edu.eg | Multiple players, Mixer and Keyboard Shortcuts |

# Table of Contents




# Class Diagram

## 1. SimpleAudioPlayer (Main Application Class)

- This is the main application entry point.
- Creates the main window of the program.

---

## 2. MainWindow

- Represents the main window shown to the user.
- Contains an instance of `MainComponent.`
- Handles closing the window and making sure the app exits safely.

---

## 3. MainComponent

- This is the brain of the application.
- Connects and controls the two main parts:
  `PlayerAudio` → sound engine
  `PlayerGUI` → user interface
- Handles:
  ✔ buttons
  ✔ keyboard shortcuts
  ✔ file selection
  ✔ timer for updating GUI (like progress bar)
- Sends data to PlayerAudio and updates GUI based on playback state.

---

## 4. PlayerAudio

- The audio engine of the project.

- Responsible for:
  ✔ Loading audio files
  ✔ Playing / Pausing / Stopping
  ✔ Volume control
  ✔ Speed and Resampling
  ✔ Getting track length and current position

- Uses JUCE audio classes like `AudioTransportSource`, `ResamplingAudioSource`, etc.
  In short: This is where sound is processed.

---

## 5. PlayerGUI

- Handles every visible thing the user interacts with:
  ✔ Buttons → Play, Pause, Stop
  ✔ Sliders → volume, speed
  ✔ Playlist display
  ✔ Progress bar

- Listens for button clicks and slider movements, and then tells `PlayerAudio` what to do.

---

## 6. SessionManager

- Handles saving and loading session data.

- Example: last played song, volume, speed… etc.

- Saves session to a file and can restore it when program opens again.

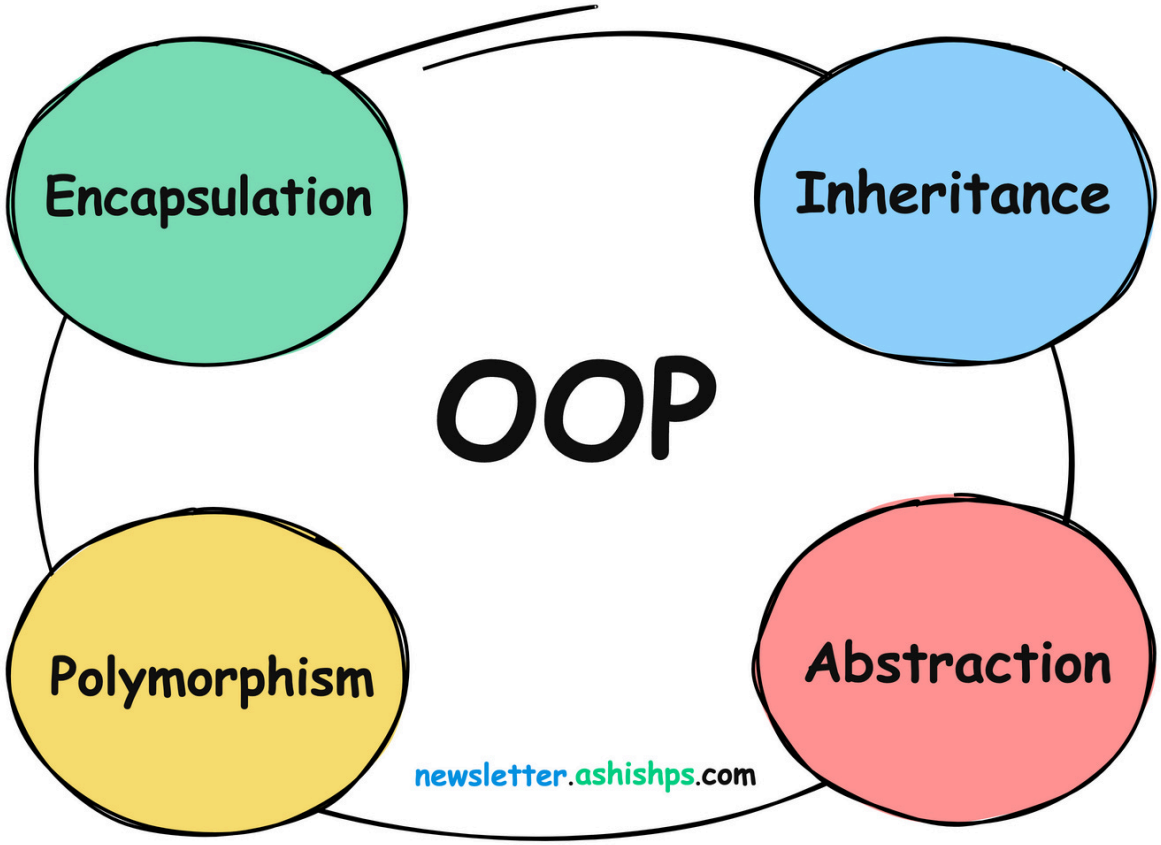
---

## 7. **ButtonListener / SliderListener / ChangeListener / MouseListener**

- Interfaces used to detect and respond to:
  ✔ button clicks
  ✔ slider value changes
  ✔ mouse events
  ✔ audio transport changes

- Allow communication between GUI and audio components.

# A PIE (OOP 4 principles)




## 1. Abstraction

**Meaning:**
Abstraction hides complex code and shows only what the user or programmer needs.

**How it was used in the project:**

- The user interface (PlayerGUI) does not deal with low-level audio processing.
- When the user presses Play, GUI just calls a simple function like:**player.start();**

## 2. Polymorphism

We used polymorphism in the `PlayerGUI` class.
`PlayerGUI` inherits from `juce::ListBoxModel` and overrides its virtual functions like `getNumRows()` and `paintListBoxItem()`.

In the constructor, we set `playList.setModel(this);`

So the ListBox only sees a ListBoxModel, but at runtime it actually calls our own implementations inside PlayerGUI.

This is polymorphism — the base class pointer calls the derived class functions.

## 3. Inheritance

**Meaning:**
A class can take (inherit) functions and variables from another class.

**How it was used:**

- **`PlayerGUI`** inherits from

  juce::Component , juce::ListBoxModel and juce::MouseListener

- **`MainComponent`** inherits from
  `juce::AudioAppComponent`, `juce::Button::Listener`, `juce::Slider::Listener`, `juce::Timer and juce::MouseListener`

- **`PlayerAudio`** inherits from

  `juce::AudioAppComponent and juce::ChangeListener`

This allowed our components to gain existing functionality without rewriting it from scratch.

## 4. Encapsulation

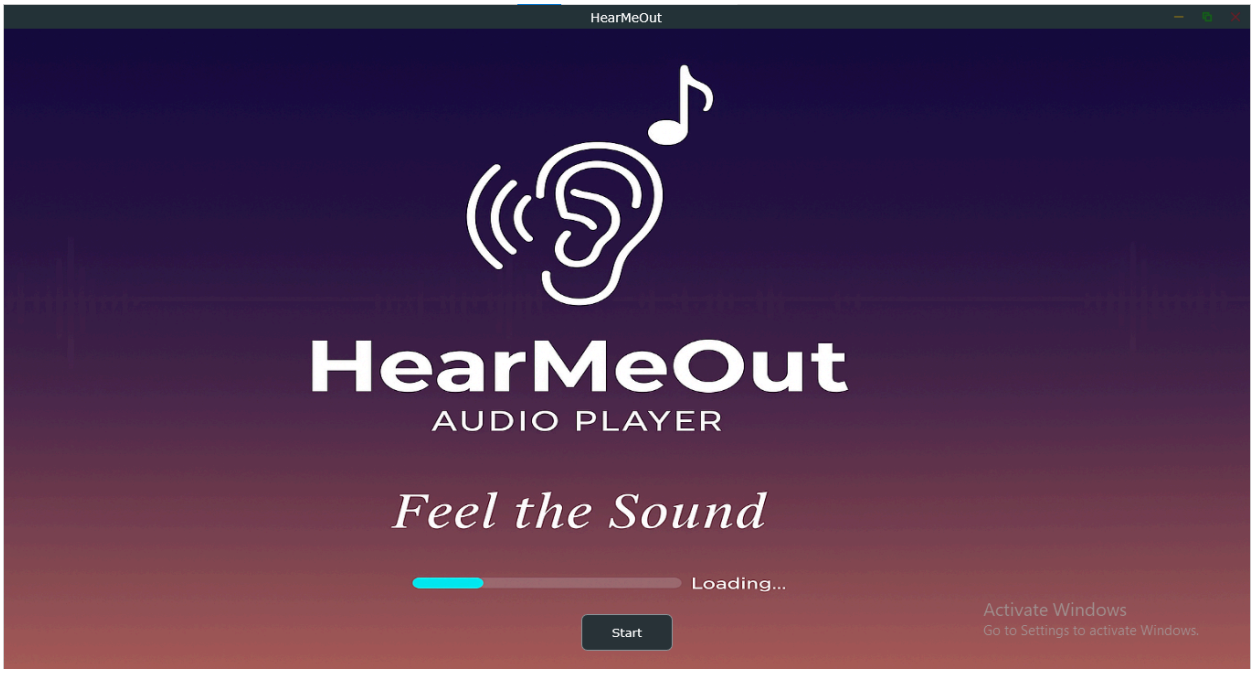
**Meaning:**
Each class contains its own data and protects it from being changed directly.

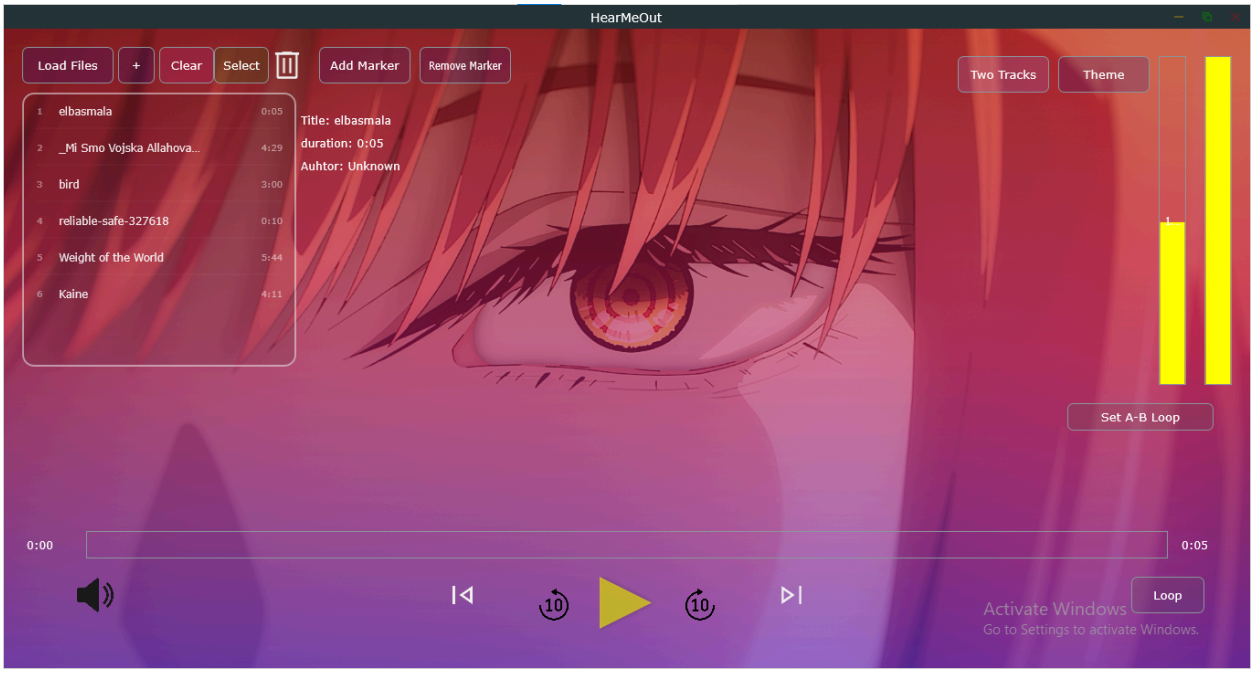
**How it was used:**

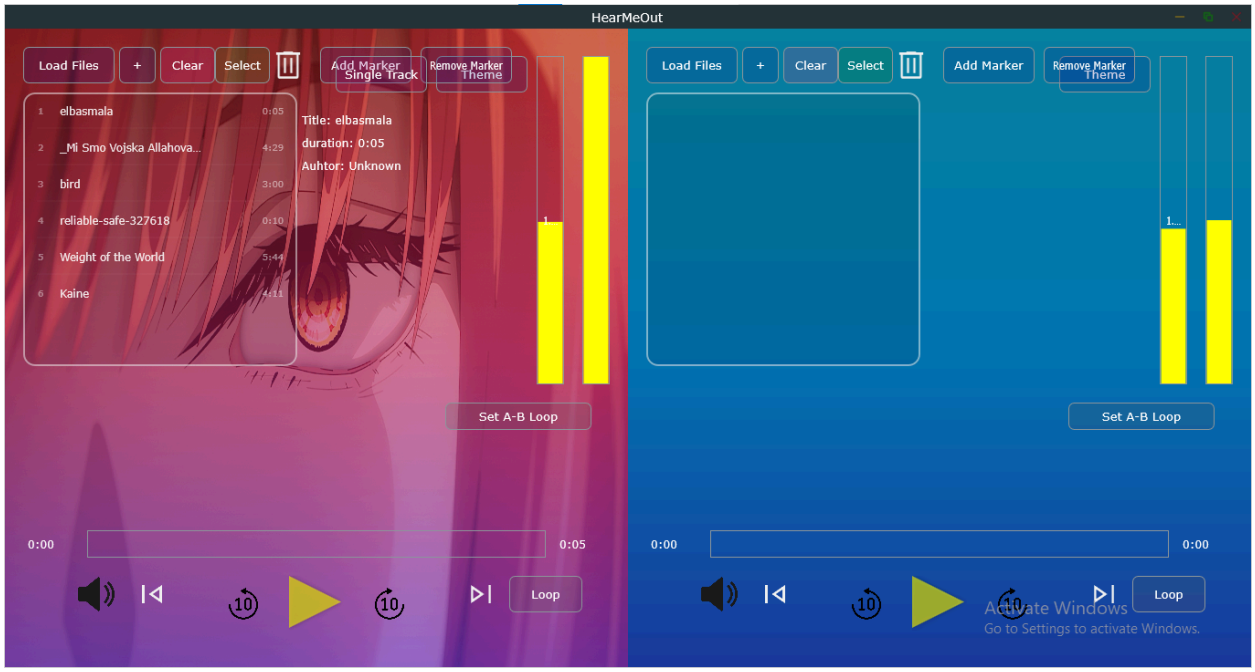
- Volume, speed, file path, and playback position are **private** inside `PlayerAudio`.
- `PlayerGUI` cannot change them directly.
- It only calls safe functions like: player.setVolume(0.5);

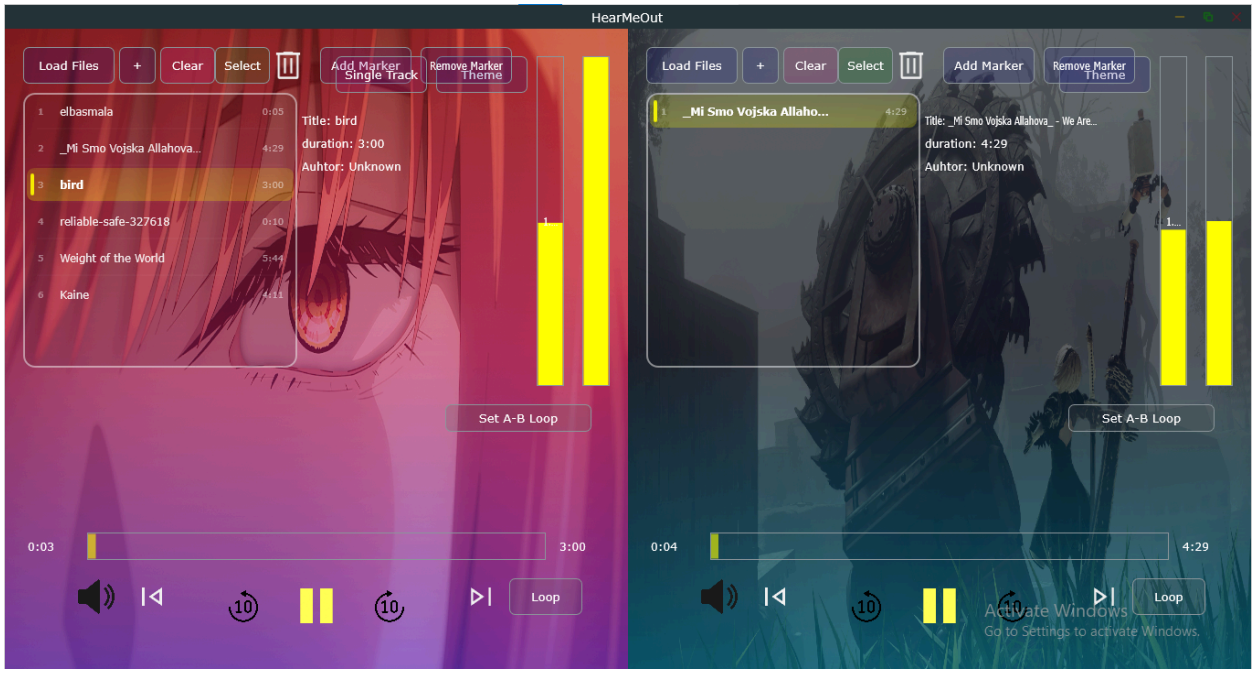
# ScreenShots

## Our Program

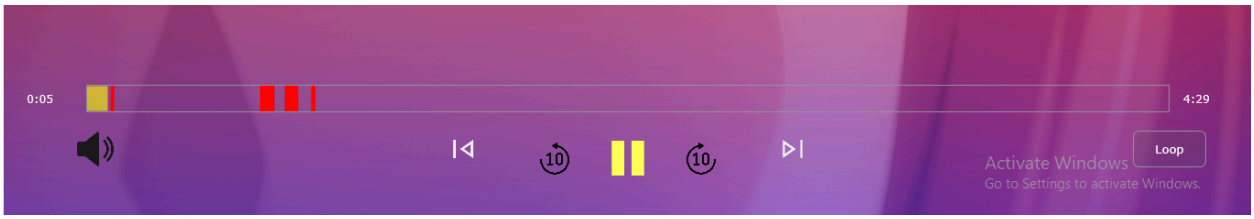


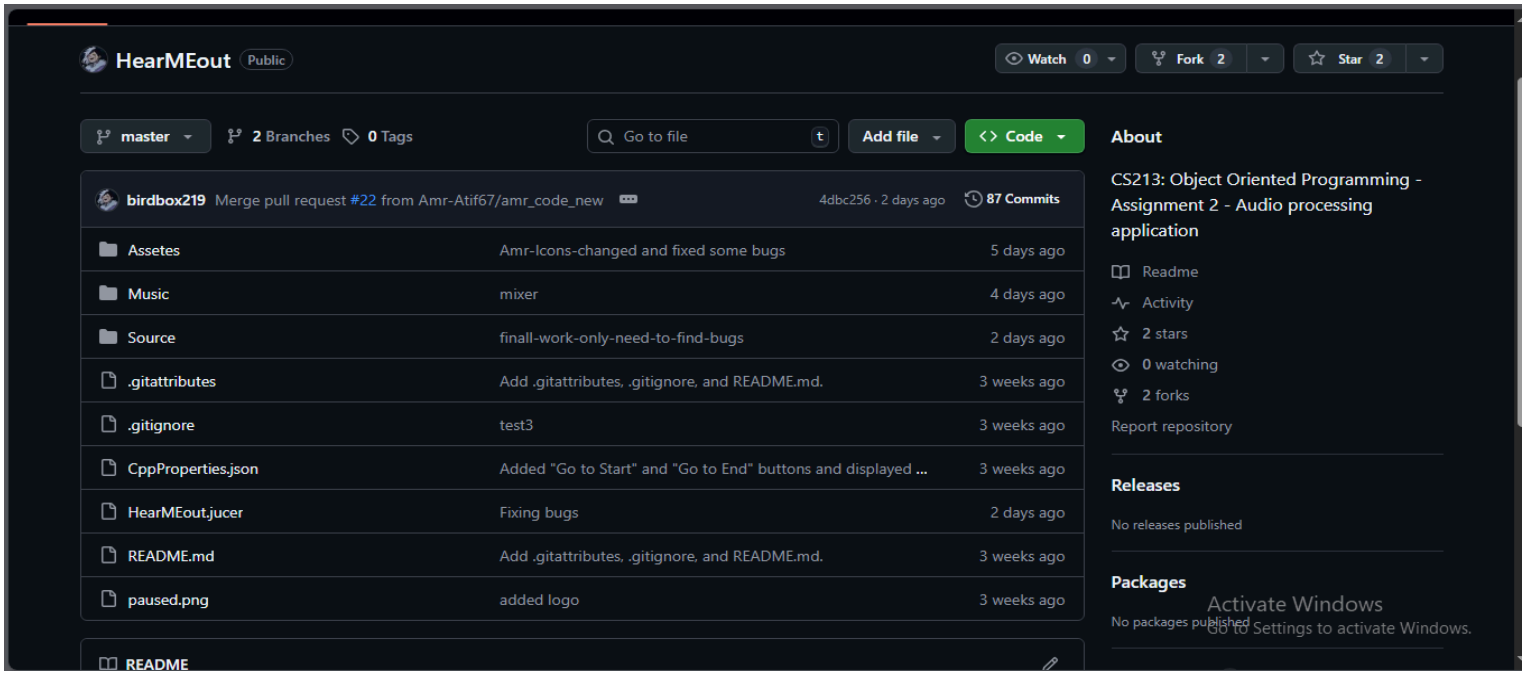

HearMeOut
Load Files
+
Clear
Select
Add Marker
Remove Marker
Single Track
Theme
1 elbasmala 0:05
2 _Mi Smo Vojska Allahova... 4:29
3 bird 3:00
4 reliable-safe-327618 0:10
5 Weight of the World 5:44
6 Kaine 4:11
Title: elbasmala
duration: 0:05
Auhtor: Unknown
Set A-B Loop
0:00
0:05
Loop
Load Files
+
Clear
Select
Add Marker
Remove Marker
Theme
Set A-B Loop
0:00
0:00
Loop
Activate Windows
Go to Settings to activate Windows.

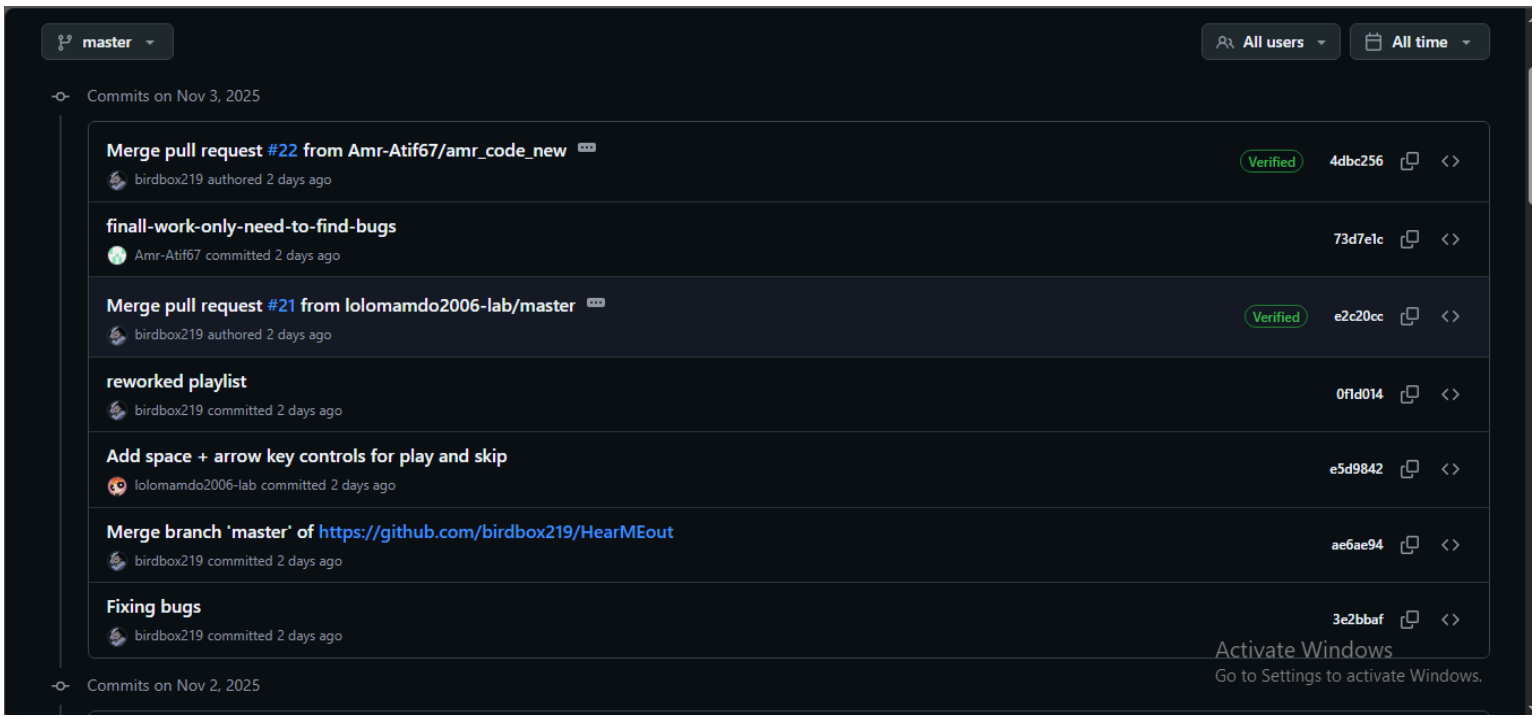

HearMeOut
Load Files
+
Clear
Select
Add Marker
Remove Marker
Single Track
Theme
1 elbasmala 0:05
2 _Mi Smo Vojska Allahova... 4:29
3 bird 3:00
4 reliable-safe-327618 0:10
5 Weight of the World 5:44
6 Kaine 4:11
Title: bird
duration: 3:00
Auhtor: Unknown
Set A-B Loop
0:03
3:00
Loop
Load Files
+
Clear
Select
Add Marker
Remove Marker
Theme
1 _Mi Smo Vojska Allaho... 4:29
Title: _Mi Smo Vojska Allahova_ - We Are...
duration: 4:29
Auhtor: Unknown
Set A-B Loop
0:04
4:29
Loop
Activate Windows
Go to Settings to activate Windows.

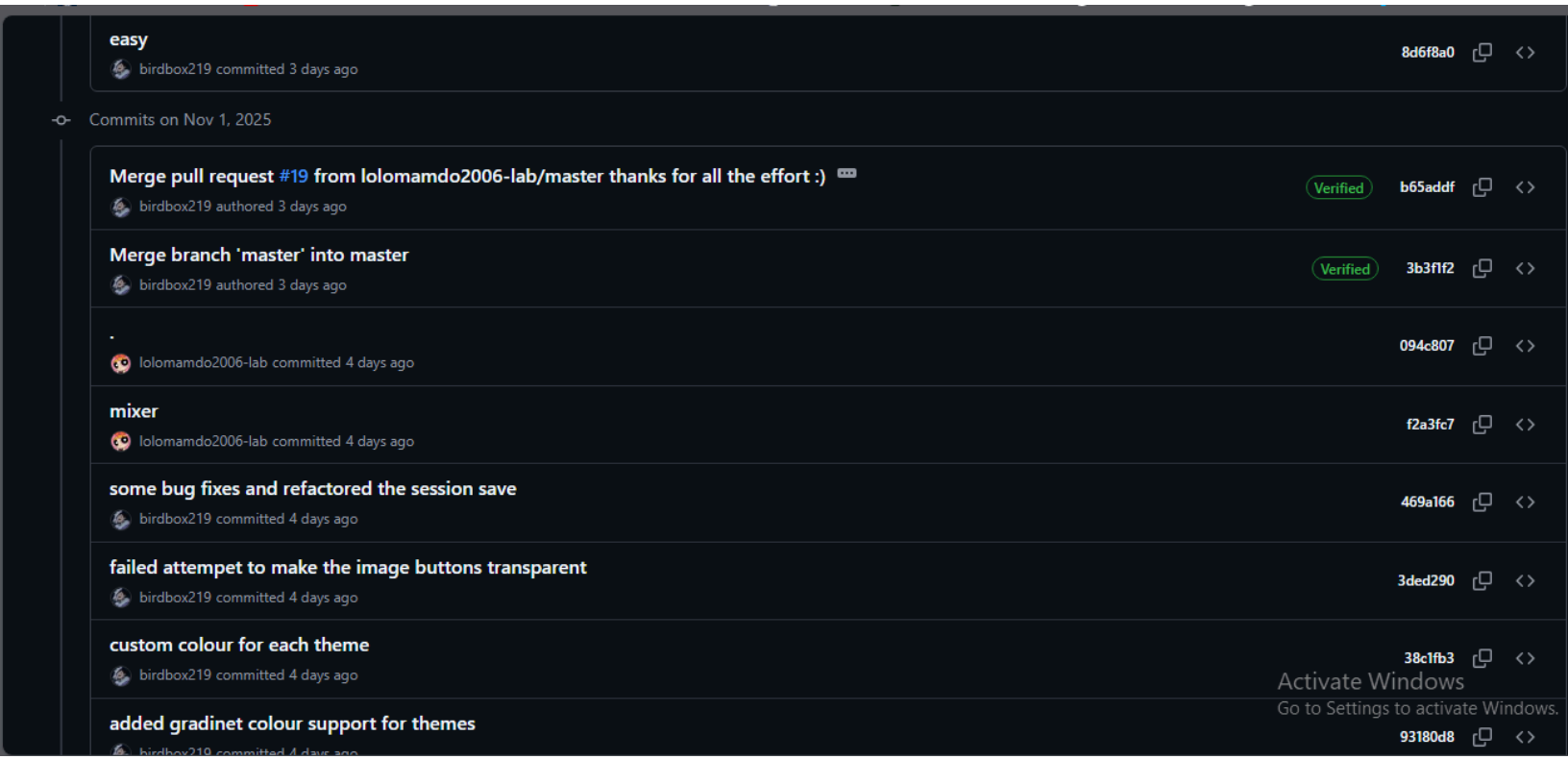

Hide A-B
Start: 3:00
End: 4:29
Start A-B
3:21
4:29
Loop
Activate Windows
Go to Settings to activate Windows.


0:05
4:29
Loop
Activate Windows
Go to Settings to activate Windows.

GitHub


HearMEout
Public
Watch 0
Fork 2
Star 2
master
2 Branches
0 Tags
Go to file
t
Add file
Code
birdbox219 Merge pull request #22 from Amr-Atif67/amr_code_new
4dbc256 · 2 days ago
87 Commits
Assetes Amr-Icons-changed and fixed some bugs 5 days ago
Music mixer 4 days ago
Source finall-work-only-need-to-find-bugs 2 days ago
.gitattributes Add .gitattributes, .gitignore, and README.md. 3 weeks ago
.gitignore test3 3 weeks ago
CppProperties.json Added "Go to Start" and "Go to End" buttons and displayed ... 3 weeks ago
HearMEout.jucer Fixing bugs 2 days ago
README.md Add .gitattributes, .gitignore, and README.md. 3 weeks ago
paused.png added logo 3 weeks ago
README
About
CS213: Object Oriented Programming - Assignment 2 - Audio processing application
Readme
Activity
2 stars
0 watching
2 forks
Report repository
Releases
No releases published
Packages
No packages published
Activate Windows
Go to Settings to activate Windows.


master
All users
All time
Commits on Nov 3, 2025
Merge pull request #22 from Amr-Atif67/amr_code_new
birdbox219 authored 2 days ago
Verified
4dbc256
finall-work-only-need-to-find-bugs
Amr-Atif67 committed 2 days ago
73d7e1c
Merge pull request #21 from lolomamdo2006-lab/master
birdbox219 authored 2 days ago
Verified
e2c20cc
reworked playlist
birdbox219 committed 2 days ago
0f1d014
Add space + arrow key controls for play and skip
lolomamdo2006-lab committed 2 days ago
e5d9842
Merge branch 'master' of https://github.com/birdbox219/HearMEout
birdbox219 committed 2 days ago
ae6ae94
Fixing bugs
birdbox219 committed 2 days ago
3e2bbaf
Commits on Nov 2, 2025
Activate Windows
Go to Settings to activate Windows.


easy
birdbox219 committed 3 days ago
8d6f8a0
Commits on Nov 1, 2025
Merge pull request #19 from lolomamdo2006-lab/master thanks for all the effort :)
birdbox219 authored 3 days ago
Verified
b65addf
Merge branch 'master' into master
birdbox219 authored 3 days ago
Verified
3b3f1f2
.
lolomamdo2006-lab committed 4 days ago
094c807
mixer
lolomamdo2006-lab committed 4 days ago
f2a3fc7
some bug fixes and refactored the session save
birdbox219 committed 4 days ago
469a166
failed attempet to make the image buttons transparent
birdbox219 committed 4 days ago
3ded290
custom colour for each theme
birdbox219 committed 4 days ago
38c1fb3
Activate Windows
Go to Settings to activate Windows.
added gradinet colour support for themes
93180d8

# Explanation of Some Implemented Features

Add pause | |  play ► buttons and “go to start” |◄ / “end” ►| buttons.

**Button Declaration(`PlayerGUI.h`):**

The `startIcon, stopButtonIcon, goStartButton and goEndButton` was first declared inside the GUI header file.

```
juce::ArrowButton startIcon{ "Play" , 0.0f , juce::Colours::yellow.withAlpha(0.6f)};
//juce::TextButton stopButton{ "Stop" };
```

```
juce::ImageButton goStartButton;
juce::ImageButton goEndButton;
juce::ImageButton stopButtonIcon;
```

Also, we defined the images that will be used for the buttons.

```
//Icon Buttons
juce::Image stopImageIcon;
juce::Image stopImageOverIcon;
juce::Image stopImageDownIcon;
juce::Image goEnd;
juce::Image goStart;
juce::Image remove;
juce::Image forward_10;
juce::Image backward_10;
juce::Image muteimage;
juce::Image unmuteimage;
```

**Button Design & UI (`PlayerGUI.cpp`):**

we made the button visible by `addAndMakeVisible(goStartButton)` and loaded the image that will be used as the icon

```
// Icon Manager
stopImageIcon = juce::ImageFileFormat::loadFrom(BinaryData::pauseyellow_png, BinaryData::pauseyellow_pngSize
goEnd = juce::ImageFileFormat::loadFrom(BinaryData::goEnd_png, BinaryData::goEnd_pngSize);
goStart = juce::ImageFileFormat::loadFrom(BinaryData::goStart_png, BinaryData::goStart_pngSize);
remove = juce::ImageFileFormat::loadFrom(BinaryData::delete_png, BinaryData::delete_pngSize);
forward_10 = juce::ImageFileFormat::loadFrom(BinaryData::ten_for_png,BinaryData::ten_for_pngSize);
backward_10 = juce::ImageFileFormat::loadFrom(BinaryData::ten_back_png,BinaryData::ten_back_pngSize);
muteimage = juce::ImageFileFormat::loadFrom(BinaryData::volume_png,BinaryData::volume_pngSize);
unmuteimage = juce::ImageFileFormat::loadFrom(BinaryData::no_volume_png,BinaryData::no_volume_pngSize);
```

### Audio Logic (`PlayerAudio.h`),(`PlayerAudio.cpp`):

We first declared the functions in the `.h` file, and then we implemented their code in the `.cpp` file.

```
void PlayerAudio::Start()
{
    transportSource.start();
}
void PlayerAudio::Start2() { ... }

void PlayerAudio::Stop()
{
    transportSource.stop();
}
void PlayerAudio::Stop2() { ... }

void PlayerAudio::goStart() {
    transportSource.setPosition(0.0);
}
void PlayerAudio::goStart2() { ... }

void PlayerAudio::goEnd() {
  transportSource.setPosition(transportSource.getLengthInSeconds());

}
void PlayerAudio::goEnd2() { ... }
```

```
void goStart();
void goStart2();

void goEnd();
void goEnd2();
```

**Connecting UI to Audio (MainComponent.cpp):**

We also added listeners to our buttons by `playerGUI.startIcon.addListener(this);` so that the system can detect clicks.

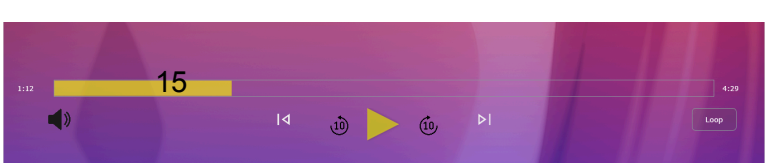
We used an `if` statement inside the `buttonClicked()` function to check which button was pressed.

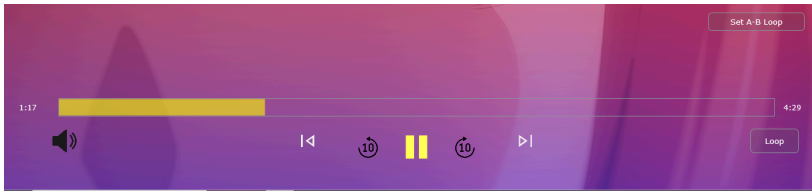
Each condition handles a different button and performs its specific action (Play, Stop, Load, etc.).

For example:

- If the user clicked the **Start** button, we started audio playback and changed the button icons.
- If the user clicked the **Stop** button, we stopped the audio and switched the icons back.
- If the user clicked **Go to Start** or **Go to End**, we moved the audio position and reset the slider.

After implementing the button logic and connecting all components together, this is the final UI and how the audio player looks while running.



# Display metadata (title, author, duration, etc.).

We also applied the same structure to the metadata labels (Title, Author, and Duration).
First, we created the labels in the `PlayerGUI.h` file:
Then, in the `PlayerGUI.cpp` file, we made them visible and styled them:

```
addAndMakeVisible(title);
addAndMakeVisible(time);
addAndMakeVisible(author);
```

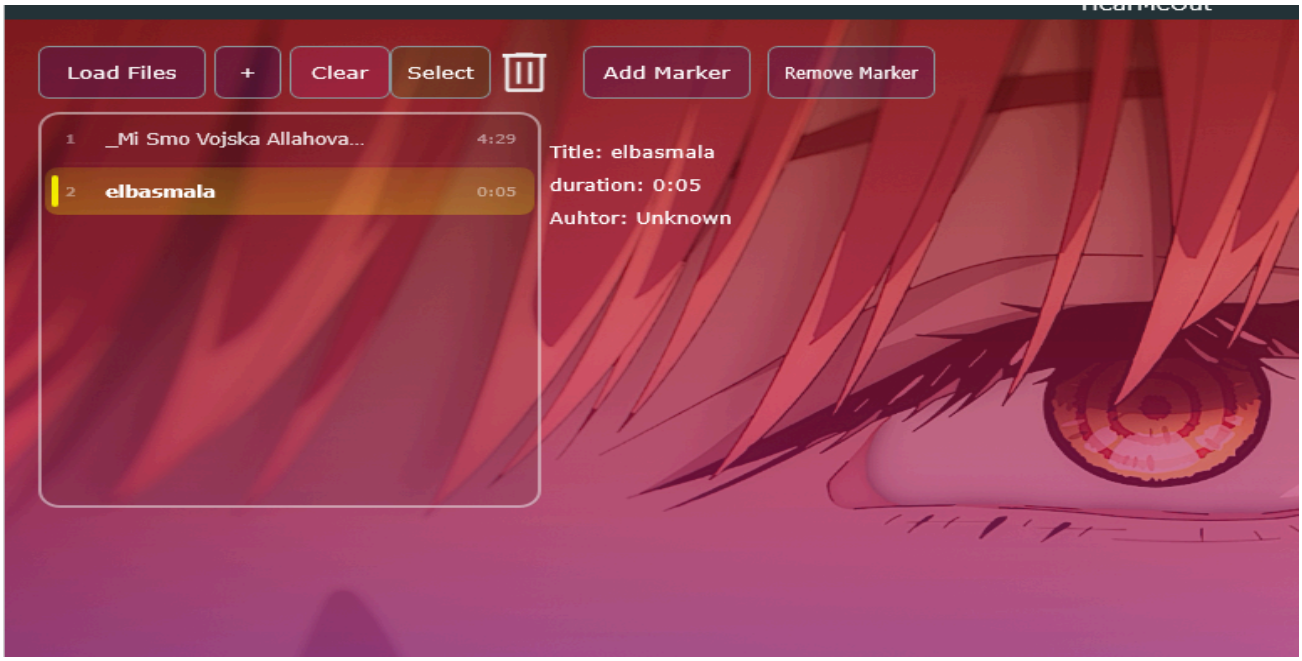
```
// Lebels
juce::Slider speedSlider;
juce::Label title;
juce::Label time;
juce::Label author;

juce::Label currentTimeLabel;
juce::Label TotalTimeLabel;
```

We created a function called `metaData()` that updates the labels with the file name, author name, and total audio duration:

```
void PlayerGUI::metaData(juce::String& fileName, double& totalTime, juce::String& authorName)
{
    title.setText("Title: " + fileName, juce::dontSendNotification);
    author.setText("Auhtor: " + authorName, juce::dontSendNotification);

    int h = 0;
    int m = 0;
    int s = 0;
    m = totalTime / 60;
    s = int(totalTime) % 60;
    if (m >= 60) {
        h = m / 60; m = m % 60;
        time.setText("duration: " + juce::String(h) + ":" + juce::String(m) + ":" + juce::String(s), juce::dontSendNotification);
    }
    //added if file less than 60min to will dispaly only mins
    else
    {
        time.setText("duration: " + juce::String(m) + ":" + juce::String(s).paddedLeft('0', 2), juce::dontSendNotification);
    }
```

After loading the audio file, we call the `metaData()` function to update the UI.

```
(button == &playerGUI.loadButton)

 fileChooser = std::make_unique<juce::FileChooser>(
     "Select an audio file...", juce::File{}, "*.wav;*.mp3");

 fileChooser->launchAsync(
     juce::FileBrowserComponent::openMode | juce::FileBrowserComponent::canSelectFiles,
     [this](const juce::FileChooser& fc)
     {
         auto file = fc.getResult();

         authorName = player.loadFile(file);
         double totalTime = player.getTotalLength();
         playerGUI.TotalTimeLabel.setText(formatTime(totalTime), juce::dontSendNotifica
         juce::String fileName = file.getFileNameWithoutExtension();
         playerGUI.metaData(fileName, totalTime, authorName);
         player.addToList(file);                    //fixing some issues and bugs-amr
         playerGUI.showFile(file, totalTime);   //same i added those lines.
```


Load Files
+
Clear
Select
Add Marker
Remove Marker
1 _Mi Smo Vojska Allahova... 4:29
2 elbasmala 0:05
Title: elbasmala
duration: 0:05
Auhtor: Unknown

## Add playlist support to load multiple files and select which one to play.

We also created a playlist feature that allows the user to add, select, and remove audio files.

**1. Declaring the Playlist Components (in PlayerGUI.h)**
We defined the buttons and the playlist structure:

`addToListButton` → adds a new audio file
`resetButton` → clears the playlist
`selectButton` → loads a selected track
`removeButton` → removes a track
`RoundedListBox playList` → displays all added tracks
`std::vector<fileInfo> files` → stores the audio files and their duration

We also declared required functions:

`getNumRows()`→returns number of tracks in the playlist
`paintListBoxItem()`→ custom UI for each row in the playlist
`showFile()`→store file name and duration in our vector
`selectedRowsChanged()`→ triggered when user selects a row
`listBoxItemDoubleClicked()`→ double-click to auto-load and play a track

```
// Playlist-related components
juce::TextButton addToListButton{"+"};
juce::TextButton resetButton{ "Clear" };
juce::TextButton selectButton{ "Select" };
juce::ImageButton removeButton;
RoundedListBox playList;
struct fileInfo {
    juce::File file;
    double time = 0.0;//changes to float to get exact point for my markers MUST STAY AS IT'S.
    std::vector<double> markersTime;
};
std::vector<fileInfo> files;
int getNumRows() override;
void paintListBoxItem(int rowNumber, juce::Graphics& g, int width, int height, bool rowIsSelecte
void showFile(juce::File& file,double time);
void selectedRowsChanged(int lastRowSelected);
void listBoxItemDoubleClicked(int row, const juce::MouseEvent&) override;

void mouseMove(const juce::MouseEvent& e) override;
void mouseExit(const juce::MouseEvent& e) override;
```

## 2. Implementing the Functions (in PlayerGUI.cpp)

### ✔ showFile()

This function checks if the file already exists in the playlist.
If not, it adds the file name and duration to the vector, then refreshes the list:

```
void PlayerGUI::showFile(juce::File& file,double time) {
    juce::File f;
    bool exist{};
    if (!files.empty()) {
        for (auto f : files) {

            if (file.getFileNameWithoutExtension() == f.file.getFileNameWithoutExtens
                exist = 1;
            }
        }
    }
    if (!exist) {
        fileInfo info;
        info.file = file;
        info.time = time;
        files.push_back(info);


    }
    playList.updateContent();
    playList.repaint();
}
```

### ✔ getNumRows()

Returns the number of items in the playlist:

```
}
int PlayerGUI::getNumRows()
{
    return files.size();
}
```

### ✔ paintListBoxItem()

This function draws each playlist row with custom UI:

- Rounded design
- Highlight on hover
- Yellow highlight on selection
- Track number and duration

```
void PlayerGUI::paintListBoxItem(int rowNumber, juce::Graphics& g,
    int width, int height, bool rowIsSelected)
{
if (rowNumber < files.size())
{
    bool isHovered = (rowNumber == hoveredRow);
    if (rowIsSelected)
    {
        // Gradient for selected item
        juce::ColourGradient selectedGradient(
            juce::Colours::yellow.withAlpha(0.15f), 4, 2,
            juce::Colours::yellow.withAlpha(0.25f), 4, height - 2,
            false
        );
        g.setGradientFill(selectedGradient);
        g.fillRoundedRectangle(4, 2, width - 8, height - 4, 8.0f);

        // Add a glowing accent line on the left
        g.setColour(juce::Colours::yellow.withAlpha(0.9f));
        g.fillRoundedRectangle(8, 8, 4, height - 16, 2.0f);
    }
```

```
if (rowNumber < files.size())
{
    bool isHovered = (rowNumber == hoveredRow);
    if (rowIsSelected) { ... }
    // Hover effect
    else if (isHovered)
    {
        g.setColour(juce::Colours::white.withAlpha(0.08f));
        g.fillRoundedRectangle(4, 2, width - 8, height - 4, 8.0f);

        // Subtle left accent on hover
        g.setColour(juce::Colours::yellow.withAlpha(0.3f));
        g.fillRoundedRectangle(8, 10, 2, height - 20, 1.0f);
    }

    // Draw track number
    g.setColour(juce::Colours::white.withAlpha(0.4f));
    g.setFont(juce::Font(11.0f, juce::Font::bold));
    g.drawText(juce::String(rowNumber + 1),
        16, 0, 20, height,
        juce::Justification::centredLeft, true);

    // Song name with better spacing
    g.setColour(rowIsSelected ? juce::Colours::white : juce::Colours::white.withAlpha(0.9f));
    g.setFont(juce::Font(14.0f, rowIsSelected ? juce::Font::bold : juce::Font::plain));

    juce::String songName = files[rowNumber].file.getFileNameWithoutExtension();
    // Truncate long names
    if (songName.length() > 30)
        songName = songName.substring(0, 27) + "...";
```

## ✔ selectedRowsChanged()

Stores the file selected by the user:

```
void PlayerGUI::selectedRowsChanged(int lastRowSelected)
{
    if (lastRowSelected >= 0)
    {
        auto file = files[lastRowSelected];

        sendFile = file.file;
        sendRow = lastRowSelected;
    }
}
```

## ✔ listBoxItemDoubleClicked()

When user double-clicks, it auto-selects and triggers the Select button

```
void PlayerGUI::listBoxItemDoubleClicked(int row, const juce::MouseEvent&)
{
    if (row >= 0 && row < files.size())
    {
        sendFile = files[row].file;
        sendRow = row;

        selectButton.triggerClick();
    }
}
```

## ✔mouseMove(const juce::MouseEvent& e)

- This function detects when the mouse moves inside the playlist.
- It calculates which row the mouse is currently hovering on.
- If the hovered row changes, it repaints the old row and the new row.
- This is used to create the “hover effect” when the cursor is over a song in the list.

## ✔ mouseExit(const juce::MouseEvent&)

- This function runs when the mouse leaves the playlist area.
- It removes the hover effect from the last hovered row.
- Then it resets `hoveredRow` to `-1` meaning “nothing is hovered”.

```cpp
    auto posInList = playList.getLocalPoint(e.eventComponent, e.getPosition());
    int row = playList.getRowContainingPosition(posInList.x, posInList.y);

    if (row != hoveredRow)
    {
        DBG("Hover changed to row " << row);

        // repaint old and new rows
        if (hoveredRow >= 0) playList.repaintRow(hoveredRow);
        hoveredRow = row;
        if (hoveredRow >= 0) playList.repaintRow(hoveredRow);
    }
}

void PlayerGUI::mouseExit(const juce::MouseEvent&)
{
    if (hoveredRow >= 0)
    {
        playList.repaintRow(hoveredRow);
        hoveredRow = -1;
    }
}
```

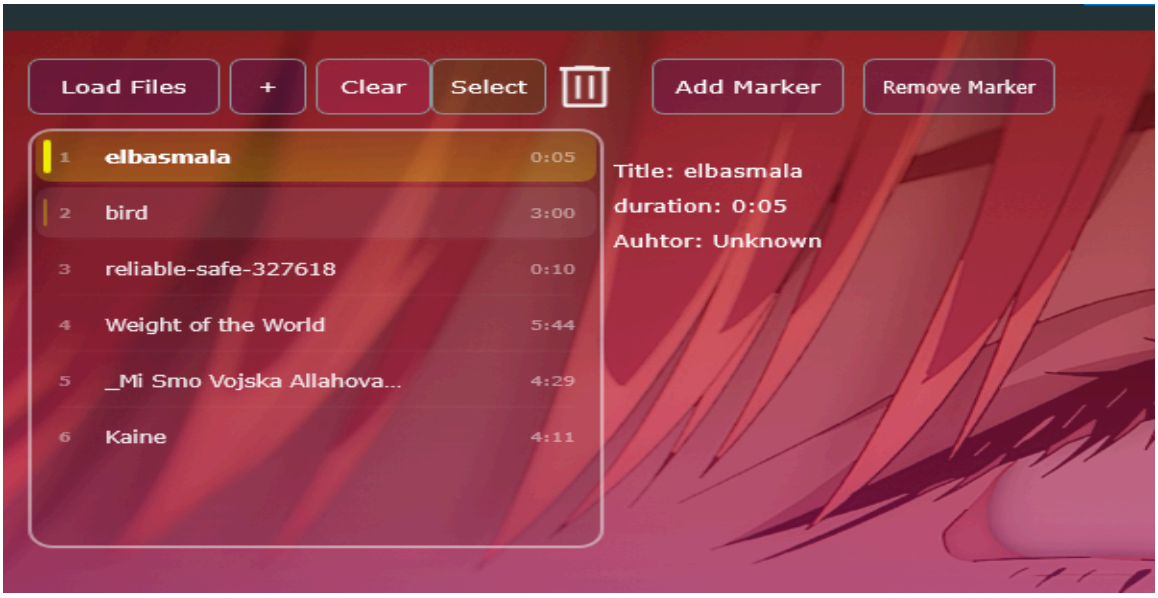
## 3. Connecting Playlist Logic in MainComponent.cpp

Inside `buttonClicked()`, we connected the playlist buttons:

- Add button → opens file chooser and adds the file to the playlist
- Reset button → clears playlist
- Select button → loads the selected file

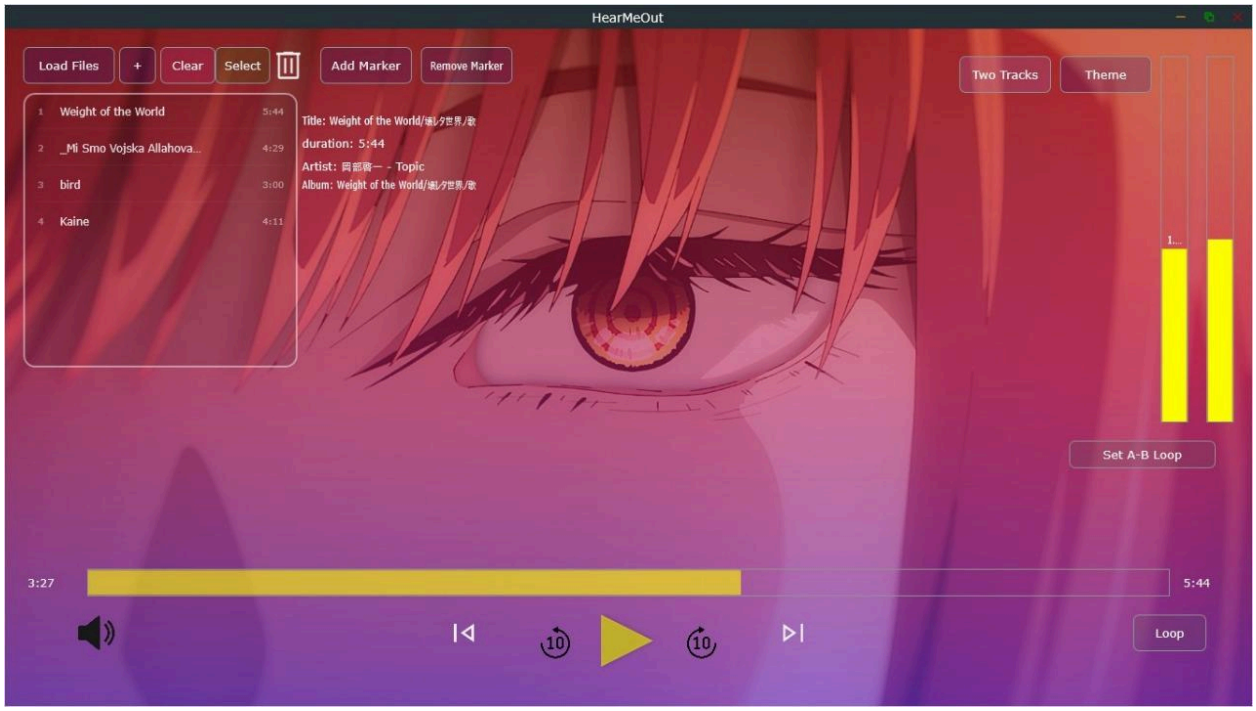
```cpp
else if (button == &playerGUI.addToListButton) { ... }
else if (button == &playerGUI.resetButton) { ... }
else if (button == &playerGUI.selectButton) { ... }
else if (button == &playerGUI.removeButton) { ... }
//==========
```

- Remove button → deletes the selected row



## **TagLib** library

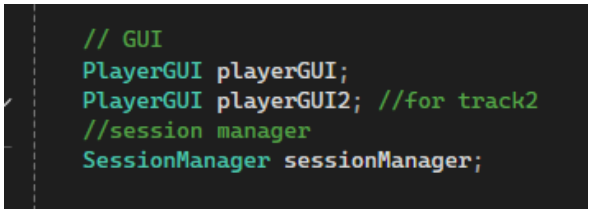
We used the **TagLib** library to read and display the metadata of each audio file, such as the title, artist name, album, and duration.



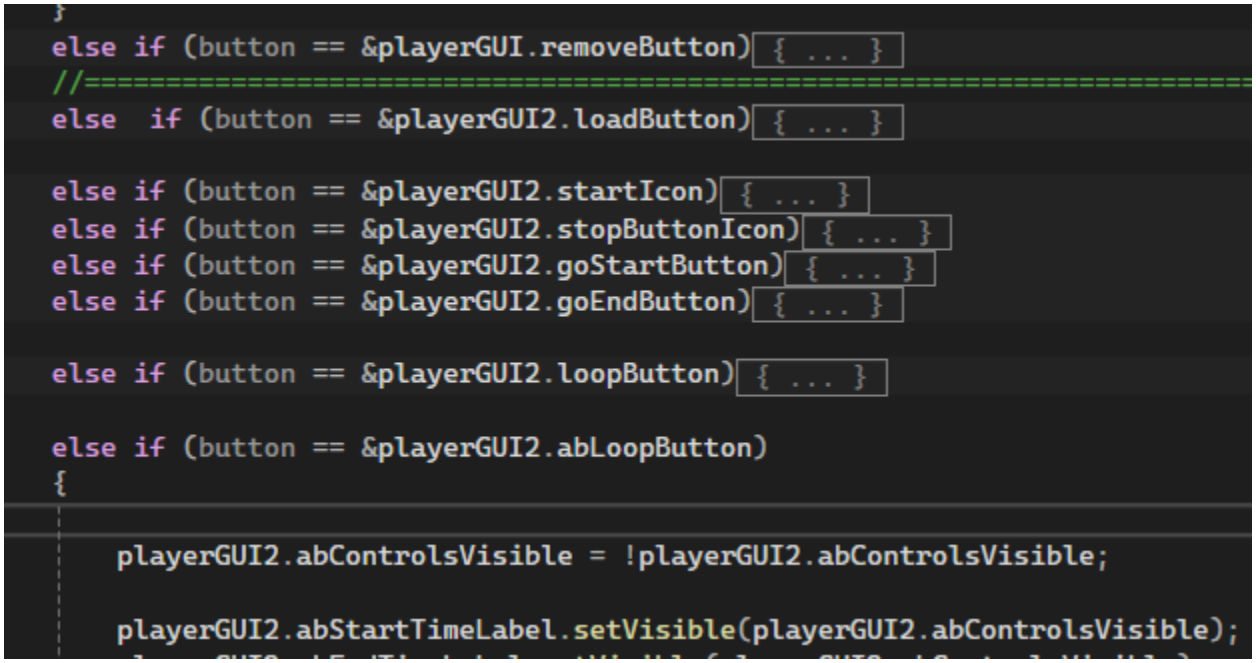
## Multiple players and Mixer

### The player can play two tracks. Player can mix the two tracks by varying each of their volumes and playing their mix.

To support multiple audio players in the application, I created another object from the `PlayerGUI` class.
This allowed me to manage two separate players on the same interface.




I duplicated the button-handling code inside `MainComponent` so each player has its own controls.




To display both players properly, I divided the main window into two separate areas:

```
if (playerGUI2.isVisible())
{
    // Two tracks mode - split the area
    auto leftArea = bounds.removeFromLeft(bounds.getWidth() / 2);
    auto rightArea = bounds;

    playerGUI.setBounds(leftArea);
    playerGUI2.setBounds(rightArea);
    playerGUI2.resized();

    // Position buttons at the top of each track area
    singleTrackButton.setBounds(leftArea.getWidth() - 320, 30, 100, 40);
}
else
```

Since each player uses its own audio transport, the two audio tracks cannot play together by default.

JUCE's `AudioTransportSource` can only output one stream at a time, so to enable simultaneous playback, I added a `juce::MixerAudioSource`.

```
void checkABLoop2();
juce::MixerAudioSource mixer; //To mix the two sounds together
juce::AudioFormatManager formatManager;
std::unique_ptr<juce::AudioFormatReaderSource> readerSource;
std::unique_ptr<juce::AudioFormatReaderSource> readerSource2; //for track2
juce::AudioTransportSource transportSource;

juce::ResamplingAudioSource respeeder { &transportSource, false, 2 };//for speed.
```

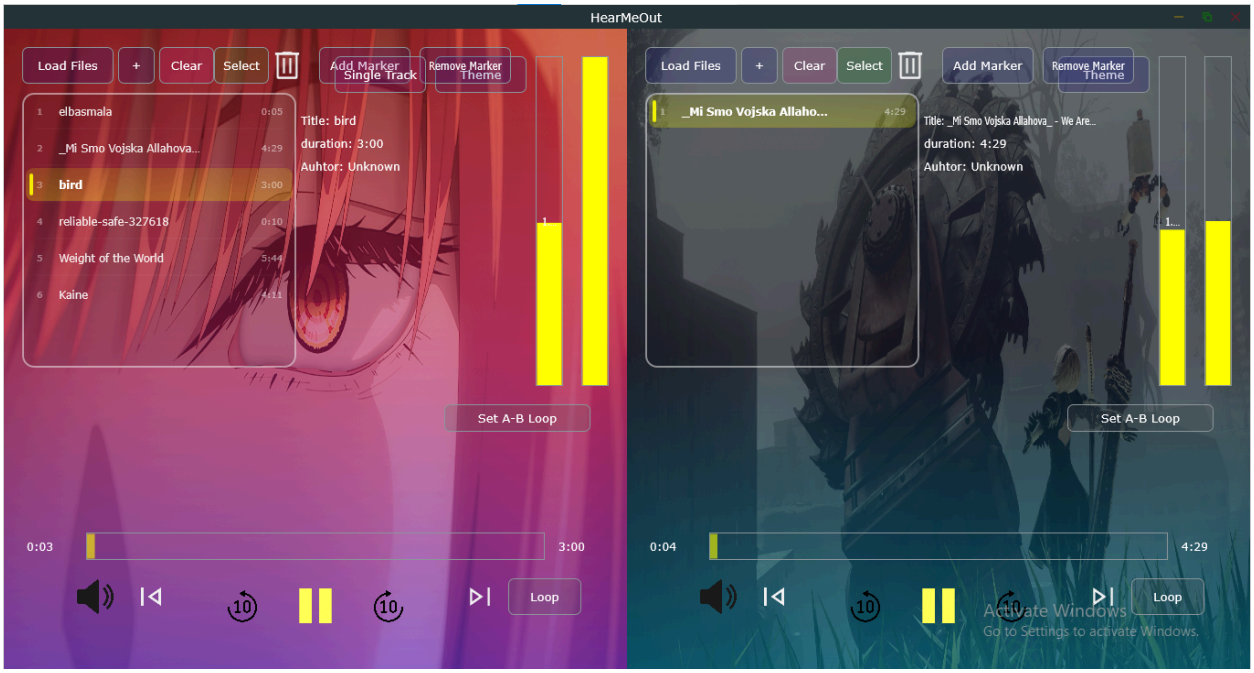
The `formatManager` is used to load different audio formats.
`transportSource` handles playback control (start, stop, pause…)
`ResamplingAudioSource` enables playback speed adjustments.

After registering the audio formats, the two audio sources are added into the mixer

To control each track independently, I implemented two versions of every main function—one for the first player and one for the second:

```
{
    formatManager.registerBasicFormats();
    transportSource.addChangeListener(this);

    respeeder.setResamplingRatio(1.0);
    respeeder2.setResamplingRatio(1.0);

    mixer.addInputSource(&respeeder, false);
    mixer.addInputSource(&respeeder2, false);

    //setAudioChannels(0, 2); // no inputs, 2 outputs
```

```
void PlayerAudio::Start()
{
    transportSource.start();
}
void PlayerAudio::Start2()
{
    transportSource2.start();
}

void PlayerAudio::Stop()
{
    transportSource.stop();
}
void PlayerAudio::Stop2()
{
    transportSource2.stop();
```


HearMeOut
Load Files
+
Clear
Select
Add Marker
Remove Marker
Single Track
Theme
1 elbasmala 0:05
2 _Mi Smo Vojska Allahova... 4:29
3 bird 3:00
4 reliable-safe-327618 0:10
5 Weight of the World 5:44
6 Kaine 4:11
Title: bird
duration: 3:00
Auhtor: Unknown
Set A-B Loop
0:03
3:00
Loop
Load Files
+
Clear
Select
Add Marker
Remove Marker
Theme
1 _Mi Smo Vojska Allaho... 4:29
Title: _Mi Smo Vojska Allahova_ - We Are...
duration: 4:29
Auhtor: Unknown
Set A-B Loop
0:04
4:29
Loop
Activate Windows
Go to Settings to activate Windows.

# Demo

You can watch the full demo of the project here:
👉 Hearmeout App Audio Player