CS213: Programming II
Assignment 2 – Enhanced Audio Player
Dr. Mohammad El-Ramly
S15,S16

Eslam Ezzat Rasmy - 20242042

Mohamed Hanafy - 20242280

Mohamed saad - 20240498

Abdelrahman Waleed - 20240319

Work Breakdown Table

Task 1: Modularization (PlayerAudio/GUI)	**Collective Work (All Members)**
Task 2, 5, 8: Play/Pause, Metadata, Playlist	**Abdelrahman Waleed - 20240319**
Task 3, 6, 9: Mute, Speed Slider, Waveform	**Eslam Ezzat Rasmy - 20242042**
Task 4, 7, 10: Loop, Position Slider, A-B Loop	**Mohamed Hanafy - 20242280**
Task 12, 13, 14: 10s Jumps, Save Session, Markers	**Mohamed saad - 20240498**
Task 11: UI Redesign & Report Finalization	**Collective Work (All Members)**
Final Report Writing	**Eslam Ezzat Rasmy - 20242042**
Demo Video Production	**Mohamed Hanafy - 20242280**

Classes Description & Diagram

1.Classes Description

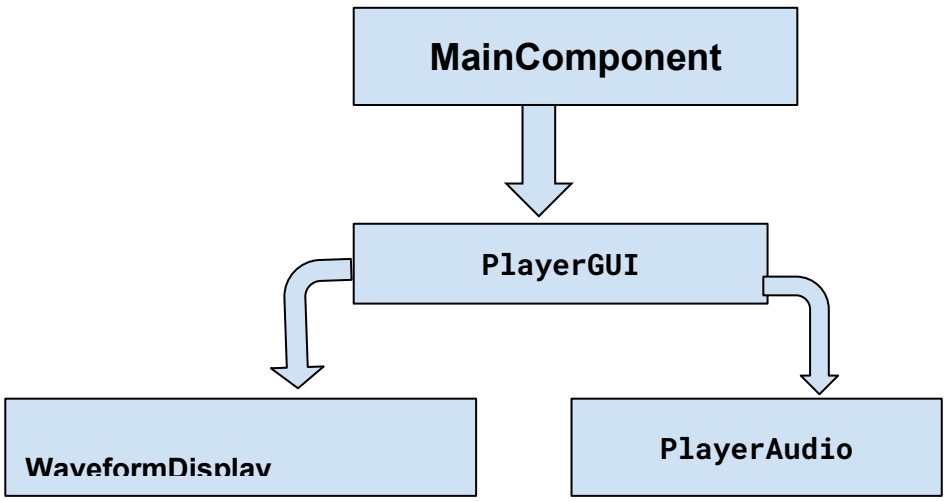
MainComponent: This class serves as the main application component, handling the overall setup and teardown of the audio player. It manages the `PlayerGUI` instance, sets up audio channels, and handles saving and loading the last session's playback state (e.g., the last played file and its position).

PlayerGUI: This class is responsible for the graphical user interface of the audio player. It manages all the visual elements like buttons (add files, play/pause, stop, mute, loop, jump forward/backward, A-B loop, markers), sliders (volume, position, speed), labels (time, metadata, speed), and the playlist box. It also handles user interactions with these UI elements and updates the `PlayerAudio` based on these interactions.

PlayerAudio: This class handles the core audio playback functionalities. It manages audio formats, loads audio files, controls playback (play, pause, stop, loop), adjusts gain (volume) and playback position, and provides functionality for jumping forward and backward in the audio. It also manages the audio transport source and resampling for speed control.

WaveformDisplay: While not a separate class file provided, the `PlayerGUI` class contains a `thumbnailComponent` which is initialized with an `AudioThumbnail`. This `thumbnailComponent` is responsible for displaying the waveform of the loaded audio file, allowing users to visually navigate the track.

2. Class Diagram



OOP Principles (A PIE) Application

Our project is built on the four fundamental principles of Object-Oriented Programming (OOP). Here is how each principle (Abstraction, Polymorphism, Inheritance, and Encapsulation) was applied using the JUCE framework and our custom classes.

A - Abstraction (التجريد)

Abstraction was applied by separating the high-level user interface logic from the complex, low-level audio processing details.

- **Example:** The `PlayerGUI` class (the "frontend") does not need to know *how* audio is processed, resampled, or read from a file. All this complexity is hidden (abstracted) away inside the `PlayerAudio` class (the "backend").
- When a user clicks the volume slider, `PlayerGUI` simply calls a high-level, abstract function like `playerAudio.setGain((float)slider->getValue())`. It doesn't know about the underlying `juce::AudioTransportSource` that is actually handling the gain.

P - Polymorphism (تعدد الأشكال)

Polymorphism ("many forms") was achieved by overriding virtual functions from the base JUCE classes that we inherited.

- **Example:** Our `PlayerGUI` class inherits from multiple `Listener` classes. To make these listeners work, we provide our own specific implementations (we override) their functions:
    - `void buttonClicked(juce::Button* button) override;`: This function is implemented in `PlayerGUI.cpp` to check *which* button was pressed (e.g., Play, Stop, Add Marker) and perform a specific action.
    - `void sliderValueChanged(juce::Slider* slider) override;`: This is overridden to react specifically to the `volumeSlider` or `positionSlider`.
    - `void paint(juce::Graphics& g) override;`: This is overridden to draw our component's custom background.

I - Inheritance (الوراثة)

Inheritance was the primary mechanism used to integrate our code with the JUCE framework and gain powerful built-in functionalities.

- **Example:** Our `PlayerGUI` class "is-a" `Component` because it inherits from `public juce::Component`. This allows it to be drawn on the screen, resized, and manage child components.
- It also inherits from:
    - `public juce::Button::Listener` (to respond to button clicks).
    - `public juce::Slider::Listener` (to respond to slider movements).
    - `public juce::Timer` (to update the position slider and time label).
    - `public juce::ListBoxModel` (to manage the playlist).

E - Encapsulation (التغليف)

Encapsulation was applied in all our classes by making member variables `private` and providing `public` functions (methods) to interact with that data safely.

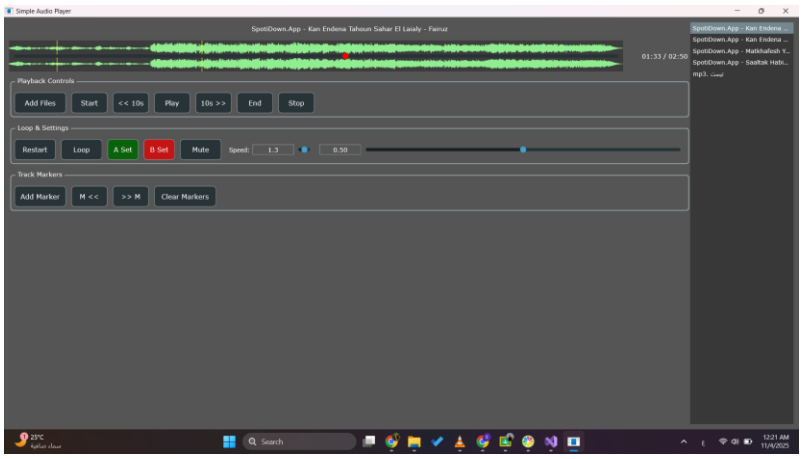
- **Example:** In the `PlayerAudio.h` file, the core audio engines like `juce::AudioTransportSource transportSource` and `std::unique_ptr<juce::ResamplingAudioSource> resamplingSource` are all declared as `private`.
- No outside class (like `PlayerGUI`) can access or modify them directly. To change the audio, `PlayerGUI` *must* use the `public` functions provided, such as `setPosition(double pos)`, `setGain(float gain)`, or `setSpeed(double speed)`. This protects the audio engine from being corrupted and ensures data integrity.

Screenshots & Demo Video Link

1. Demo Video Link

https://youtu.be/Jrp07Gz0H_4?si=eXwpiPDk_zJ3Zj_m

2. Application Screenshots



3.Github

https://github.com/IsLam122005/audio-player2