Main.cpp

```cpp
/*
  ==============================================================================

    Main.cpp
    Created: 10 Aug 2025 8:28:23am
    Author:  joao

  ==============================================================================
*/

#include "../JuceLibraryCode/JuceHeader.h"
#include "MainComponent.h"

//==============================================================================
class OtoDecksApplication : public JUCEApplication {
public:
    //==============================================================================
    OtoDecksApplication() {}
    const String getApplicationName() override {
        return ProjectInfo::projectName;
    }
    const String getApplicationVersion() override {
        return ProjectInfo::versionString;
    }
    bool moreThanOneInstanceAllowed() override { return true; }

    //==============================================================================
    void initialise(const String &commandLine) override {
        // This method is where you should put your application's initialisation
        // code..

        mainWindow.reset(new MainWindow(getApplicationName()));
    }

    void shutdown() override {
        // Add your application's shutdown code here..

        mainWindow = nullptr; // (deletes our window)
    }

    //==============================================================================
    void systemRequestedQuit() override {
        // This is called when the app is being asked to quit: you can ignore this
        // request and let the app carry on running, or call quit() to allow the app
        // to close.
        quit();
    }

    void anotherInstanceStarted(const String &commandLine) override {
        // When another instance of the app is launched while this one is running,
        // this method is invoked, and the commandLine parameter tells you what
        // the other instance's command-line arguments were.
    }

    //==============================================================================
    /*
        This class implements the desktop window that contains an instance of
        our MainComponent class.
```

```cpp
    */
    class MainWindow : public DocumentWindow {
    public:
      MainWindow(String name)
          : DocumentWindow(
                name,
                Desktop::getInstance().getDefaultLookAndFeel().findColour(
                    ResizableWindow::backgroundColourId),
                DocumentWindow::allButtons) {
        setUsingNativeTitleBar(true);
        setContentOwned(new MainComponent(), true);

#if JUCE_IOS || JUCE_ANDROID
        setFullScreen(true);
#else
        setResizable(true, true);
        centreWithSize(getWidth(), getHeight());
#endif

        setVisible(true);
      }

      void closeButtonPressed() override {
        // This is called when the user tries to close this window. Here, we'll
        // just ask the app to quit when this happens, but you can change this to
        // do whatever you need.
        JUCEApplication::getInstance()->systemRequestedQuit();
      }

      /* Note: Be careful if you override any DocumentWindow methods - the base
         class uses a lot of them, so by overriding you might break its
         functionality. It's best to do all your work in your content component
         instead, but if you really have to override any DocumentWindow methods,
         make sure your subclass also calls the superclass's method.
      */

    private:
      JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(MainWindow)
    };

private:
  std::unique_ptr<MainWindow> mainWindow;
};

//==============================================================================
// This macro generates the main() routine that launches the app.
START_JUCE_APPLICATION(OtoDecksApplication)
```

MainComponent.h

```cpp
/*
  ==============================================================================

    MainComponent.h
    Created: 10 Aug 2025 8:28:35am
    Author:  joao

  ==============================================================================
```

```cpp
*/

#pragma once

#include "../JuceLibraryCode/JuceHeader.h"
#include "DJAudioPlayer.h"
#include "DeckGUI.h"
#include "MixerComponent.h"
#include "PlaylistComponent.h"


/**
 * Main component class that serves as the central audio application component
 * It handles audio routing, crossfading, and master volume control.
 */
class MainComponent : public AudioAppComponent {
public:
  /**
   * Constructor for MainComponent
   * Set application size
   * Request audio permissions
   * Add child components
   * Set deck references in playlist component
   * Setup mixer callbacks
   * Initialize mixer with default values to set proper initial gains
   * Start with both decks at equal volume
   * Start with 70% master volume
   */
  MainComponent();

  /**
   * Destructor for MainComponent
   * shuts down the audio device and clears the audio source.
   */
  ~MainComponent();

  /**
   * Prepares the main component for audio playback
   * @param samplesPerBlockExpected Expected number of samples per audio block
   * @param sampleRate The sample rate for audio processing
   */
  void prepareToPlay(int samplesPerBlockExpected, double sampleRate) override;

  /**
   * Fills the audio buffer with the next block of mixed audio data
   * @param bufferToFill Audio buffer to be filled with mixed audio samples
   */
  void getNextAudioBlock(const AudioSourceChannelInfo &bufferToFill) override;

  /**
   * Releases audio resources when the application is stopped
   */
  void releaseResources() override;

  /**
   * Handles the painting of the main component
   * @param g Graphics context for drawing operations
   */
  void paint(Graphics &g) override;
```

```cpp
    /**
     * Handles the resizing of the main component and its child components
     */
    void resized() override;

    /**
     * Callback function for when the crossfader value changes
     * This function will adjust each deck's gain based on the crossfader value
     * @param value Crossfader value
     */
    void onCrossfaderChanged(double value);

    /**
     * Callback function for when the master volume changes
     * @param volume Master volume value
     */
    void onMasterVolumeChanged(double volume);

private:
    AudioFormatManager formatManager;
    AudioThumbnailCache thumbCache{100};

    DJAudioPlayer player1{formatManager};
    DeckGUI deckGUI1{&player1, formatManager, thumbCache};

    DJAudioPlayer player2{formatManager};
    DeckGUI deckGUI2{&player2, formatManager, thumbCache};

    MixerAudioSource mixerSource;

    PlaylistComponent playlistComponent;
    MixerComponent mixerComponent;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(MainComponent)
};
```

MainComponent.cpp

```cpp
/*
  ==============================================================================

    MainComponent.cpp
    Created: 10 Aug 2025 8:28:35am
    Author:  joao

  ==============================================================================
*/

#include "MainComponent.h"

//==============================================================================
MainComponent::MainComponent() {
    setSize(1200, 600);

    if (RuntimePermissions::isRequired(RuntimePermissions::recordAudio) &&
        !RuntimePermissions::isGranted(RuntimePermissions::recordAudio)) {
      RuntimePermissions::request(RuntimePermissions::recordAudio,
                                  [&](bool granted) {
                                      if (granted)
```

```cpp
                                        setAudioChannels(2, 2);
                            });
  } else {
    setAudioChannels(0, 2);
  }

  addAndMakeVisible(deckGUI1);
  addAndMakeVisible(deckGUI2);
  addAndMakeVisible(playlistComponent);
  addAndMakeVisible(mixerComponent);

  playlistComponent.setDeckReferences(&deckGUI1, &deckGUI2);

  mixerComponent.setCrossfaderCallback(
      [this](double value) { onCrossfaderChanged(value); });
  mixerComponent.setMasterVolumeCallback(
      [this](double volume) { onMasterVolumeChanged(volume); });

  onCrossfaderChanged(0.5);
  onMasterVolumeChanged(0.7);

  formatManager.registerBasicFormats();
}

MainComponent::~MainComponent() { shutdownAudio(); }

//==============================================================================
void MainComponent::prepareToPlay(int samplesPerBlockExpected,
                                  double sampleRate) {
  player1.prepareToPlay(samplesPerBlockExpected, sampleRate);
  player2.prepareToPlay(samplesPerBlockExpected, sampleRate);

  mixerSource.prepareToPlay(samplesPerBlockExpected, sampleRate);

  mixerSource.addInputSource(&player1, false);
  mixerSource.addInputSource(&player2, false);
}
void MainComponent::getNextAudioBlock(
    const AudioSourceChannelInfo &bufferToFill) {
  mixerSource.getNextAudioBlock(bufferToFill);
}

void MainComponent::releaseResources() {
  player1.releaseResources();
  player2.releaseResources();
  mixerSource.releaseResources();
}

//==============================================================================
void MainComponent::paint(Graphics &g) {
  g.fillAll(getLookAndFeel().findColour(ResizableWindow::backgroundColourId));
}

void MainComponent::resized() {
  auto area = getLocalBounds();

  int deckWidth = area.getWidth() / 3;
  int centerWidth = area.getWidth() / 3;
  int topHeight = area.getHeight() / 2;
  int bottomHeight = area.getHeight() / 2;
```

```
    deckGUI1.setBounds(0, 0, deckWidth, area.getHeight());
    playlistComponent.setBounds(deckWidth, 0, centerWidth, topHeight);
    deckGUI2.setBounds(deckWidth + centerWidth, 0, deckWidth, area.getHeight());

    mixerComponent.setBounds(deckWidth, topHeight, centerWidth, bottomHeight);
}

/**
 * This function was written by me without assistance based on the code provided
 * during the module and the JUCE documentation
 */
void MainComponent::onCrossfaderChanged(double value) {
    double deck1Gain, deck2Gain;

    if (value <= 0.5) {
        deck1Gain = 1.0;
        deck2Gain = value * 2.0;
    } else {
        deck1Gain = 2.0 * (1.0 - value);
        deck2Gain = 1.0;
    }

    player1.setGain(deck1Gain);
    player2.setGain(deck2Gain);
}

/**
 * This function was written by me without assistance based on the code provided
 * during the module and the JUCE documentation
 */
void MainComponent::onMasterVolumeChanged(double volume) {
    double crossfaderValue = mixerComponent.getCrossfaderValue();

    double deck1Gain, deck2Gain;

    if (crossfaderValue <= 0.5) {
        deck1Gain = volume;
        deck2Gain = volume * crossfaderValue * 2.0;
    } else {
        deck1Gain = volume * 2.0 * (1.0 - crossfaderValue);
        deck2Gain = volume;
    }

    player1.setGain(deck1Gain);
    player2.setGain(deck2Gain);
}
```

DeckGUI.h

```
/*
  ==============================================================================

    DeckGUI.h
    Created: 10 Aug 2025 8:34:24am
    Author:  joao


  ==============================================================================
*/
```

```cpp
#pragma once

#include "../JuceLibraryCode/JuceHeader.h"
#include "DJAudioPlayer.h"
#include "WaveformDisplay.h"


//==============================================================================
/*
 */
class DeckGUI : public Component,
                public Button::Listener,
                public Slider::Listener,
                public Timer {
public:
  /**
   * Constructor: Initializes the deck GUI with audio player and format
   * management components
   * @param player Pointer to the DJAudioPlayer that handles audio playback
   * @param formatManagerToUse Reference to audio format manager for loading
   * different audio formats
   * @param cacheToUse Reference to thumbnail cache for waveform generation
   */
  DeckGUI(DJAudioPlayer *player, AudioFormatManager &formatManagerToUse,
          AudioThumbnailCache &cacheToUse);

  /**
   * Destructor: Cleans up resources when the DeckGUI is destroyed
   */
  ~DeckGUI();

  /**
   * Renders the deck GUI components to the screen
   * @param g Graphics context used for drawing the component
   */
  void paint(Graphics &) override;

  /**
   * Handles layout and positioning of child components when the deck is resized
   */
  void resized() override;

  /**
   * Responds to button click events (play, pause, stop buttons)
   * @param button Pointer to the button that was clicked
   */
  void buttonClicked(Button *) override;

  /**
   * Responds to slider value changes (volume, speed, position sliders)
   * @param slider Pointer to the slider whose value changed
   */
  void sliderValueChanged(Slider *slider) override;

  /**
   * Called periodically by timer to update the position slider and waveform
   * display Updates the GUI to reflect current playback position
   */
  void timerCallback() override;
```

```cpp
    /**
     * Loads an audio track into the deck for playback
     * @param audioFile Reference to the audio file to be loaded
     */
    void loadTrack(const juce::File &audioFile);

private:
    juce::FileChooser fChooser{"Select a file..."};

    TextButton playButton{"PLAY"};
    TextButton pauseButton{"PAUSE"};
    TextButton stopButton{"STOP"};

    Slider volSlider;
    Slider speedSlider;
    Slider posSlider;

    WaveformDisplay waveformDisplay;

    DJAudioPlayer *player;

    bool isUpdatingPosition;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(DeckGUI)
};
```

DeckGUI.cpp

```cpp
/*
  ==============================================================================

    DeckGUI.cpp
    Created: 10 Aug 2025 8:34:24am
    Author:  joao

  ==============================================================================
*/

#include "DeckGUI.h"
#include "../JuceLibraryCode/JuceHeader.h"
#include <cmath>


//==============================================================================
DeckGUI::DeckGUI(DJAudioPlayer *_player, AudioFormatManager &formatManagerToUse,
                 AudioThumbnailCache &cacheToUse)
    : player(_player), waveformDisplay(formatManagerToUse, cacheToUse),
      isUpdatingPosition(false) {

  addAndMakeVisible(playButton);
  addAndMakeVisible(pauseButton);
  addAndMakeVisible(stopButton);

  addAndMakeVisible(volSlider);
  addAndMakeVisible(speedSlider);
  addAndMakeVisible(posSlider);

  addAndMakeVisible(waveformDisplay);
```

```cpp
    playButton.addListener(this);
    pauseButton.addListener(this);
    stopButton.addListener(this);

    volSlider.addListener(this);
    speedSlider.addListener(this);
    posSlider.addListener(this);

    volSlider.setSliderStyle(Slider::LinearVertical);
    volSlider.setRange(0.0, 1.0);
    volSlider.setValue(0.5);

    speedSlider.setSliderStyle(Slider::LinearVertical);
    speedSlider.setRange(0.1, 3.0);
    speedSlider.setValue(1.0);

    posSlider.setSliderStyle(Slider::LinearHorizontal);
    posSlider.setRange(0.0, 1.0);
    posSlider.setValue(0.0);

    playButton.setButtonText("PLAY");
    pauseButton.setButtonText("PAUSE");
    stopButton.setButtonText("STOP");

    startTimer(500);
}

DeckGUI::~DeckGUI() { stopTimer(); }

void DeckGUI::paint(Graphics &g) {
    g.fillAll(getLookAndFeel().findColour(ResizableWindow::backgroundColourId));

    g.setColour(Colours::grey);
    g.drawRect(getLocalBounds(), 1);

    g.setColour(Colours::white);
    g.setFont(14.0f);
    g.drawText("DeckGUI", getLocalBounds(), Justification::centred, true);
}

/**
 * This function was written by me based on the code provided in the lecture,
 * the JUCE documentation related to FlexBox
 * and the JUCE examples and tutorials (mentioned in the report pdf)
 */
void DeckGUI::resized() {
    auto area = getLocalBounds();
    area.removeFromTop(30);

    FlexBox mainLayout;
    mainLayout.flexDirection = FlexBox::Direction::column;

    FlexBox row1Layout;
    row1Layout.flexDirection = FlexBox::Direction::column;

    FlexItem waveformItem(waveformDisplay);
    waveformItem.height = 120;
    waveformItem.minHeight = 120;
    waveformItem.maxHeight = 120;
```

```cpp
FlexItem positionItem(posSlider);
positionItem.height = 40;
positionItem.minHeight = 40;
positionItem.maxHeight = 40;

row1Layout.items.add(waveformItem);
row1Layout.items.add(positionItem);

FlexBox row2Layout;
row2Layout.flexDirection = FlexBox::Direction::row;

FlexItem emptyColumn;
emptyColumn.flexGrow = 0.6f;

FlexItem speedItem(speedSlider);
speedItem.flexGrow = 0.2f;

FlexItem volumeItem(volSlider);
volumeItem.flexGrow = 0.2f;

row2Layout.items.add(emptyColumn);
row2Layout.items.add(speedItem);
row2Layout.items.add(volumeItem);

FlexBox row3Layout;
row3Layout.flexDirection = FlexBox::Direction::row;
row3Layout.justifyContent = FlexBox::JustifyContent::spaceBetween;

FlexItem playItem(playButton);
playItem.flexGrow = 1.0f;
playItem.height = 40;
playItem.minHeight = 40;
playItem.maxHeight = 40;
playItem.margin = FlexItem::Margin(0, 5, 0, 0);

FlexItem pauseItem(pauseButton);
pauseItem.flexGrow = 1.0f;
pauseItem.height = 40;
pauseItem.minHeight = 40;
pauseItem.maxHeight = 40;
pauseItem.margin = FlexItem::Margin(0, 5, 0, 5);

FlexItem stopItem(stopButton);
stopItem.flexGrow = 1.0f;
stopItem.height = 40;
stopItem.minHeight = 40;
stopItem.maxHeight = 40;
stopItem.margin = FlexItem::Margin(0, 0, 0, 5);

row3Layout.items.add(playItem);
row3Layout.items.add(pauseItem);
row3Layout.items.add(stopItem);

FlexItem row1Item;
row1Item.associatedFlexBox = &row1Layout;
row1Item.height = 160;
row1Item.minHeight = 160;
row1Item.maxHeight = 160;

FlexItem row2Item;
```

```cpp
    row2Item.associatedFlexBox = &row2Layout;
    row2Item.flexGrow = 1.0f;

    FlexItem row3Item;
    row3Item.associatedFlexBox = &row3Layout;
    row3Item.height = 50;
    row3Item.minHeight = 50;
    row3Item.maxHeight = 50;
    row3Item.margin = FlexItem::Margin(10, 15, 10, 15);

    mainLayout.items.add(row1Item);
    mainLayout.items.add(row2Item);
    mainLayout.items.add(row3Item);

    mainLayout.performLayout(area);
}

void DeckGUI::buttonClicked(Button *button) {
    if (button == &playButton) {
        std::cout << "Play button was clicked " << std::endl;
        player->start();
    }
    if (button == &pauseButton) {
        std::cout << "Pause button was clicked " << std::endl;
        player->stop();
    }
    if (button == &stopButton) {
        std::cout << "Stop button was clicked " << std::endl;
        player->stop();
        player->setPositionRelative(0.0);
    }
}

void DeckGUI::sliderValueChanged(Slider *slider) {
    if (slider == &volSlider) {
        player->setGain(slider->getValue());
    }

    if (slider == &speedSlider) {
        player->setSpeed(slider->getValue());
    }

    if (slider == &posSlider) {
        if (!isUpdatingPosition) {
            player->setPositionRelative(slider->getValue());
        }
    }
}

void DeckGUI::timerCallback() {
    if (player == nullptr)
        return;

    double currentPosition = player->getPositionRelative();

    if (std::isnan(currentPosition) || std::isinf(currentPosition) ||
        currentPosition < 0.0 || currentPosition > 1.0) {
        return;
    }
```

```cpp
    waveformDisplay.setPositionRelative(currentPosition);

    isUpdatingPosition = true;
    posSlider.setValue(currentPosition, dontSendNotification);
    isUpdatingPosition = false;

    if (currentPosition >= 0.99) {
      player->stop();
      player->setPositionRelative(0.0);

      std::cout << "Track finished - stopped and reset to beginning" << std::endl;
    }
}

void DeckGUI::loadTrack(const juce::File &audioFile) {
  if (audioFile.existsAsFile()) {
    player->loadURL(URL{audioFile});
    waveformDisplay.loadURL(URL{audioFile});
    std::cout << "Loaded track: " << audioFile.getFileName() << std::endl;
  }
}
```

DJAudioPlayer.h

```cpp
/*
  ==============================================================================

    DJAudioPlayer.h
    Created: 10 Aug 2025 8:35:47am
    Author:  joao

  ==============================================================================
*/

#pragma once

#include "../JuceLibraryCode/JuceHeader.h"

/**
 * Audio player class that handles audio file playback, transport control, and
 * audio processing This class inherits from AudioSource and provides
 * functionality for loading, playing, stopping, and controlling audio playback
 * with features like gain control, speed adjustment, and position seeking.
 */
class DJAudioPlayer : public AudioSource {
public:
  /**
   * Constructor for DJAudioPlayer
   * @param _formatManager Reference to the audio format manager for handling
   * different audio formats
   */
  DJAudioPlayer(AudioFormatManager &_formatManager);

  /**
   * Destructor for DJAudioPlayer
   */
  ~DJAudioPlayer();

  /**
```

```cpp
 * Prepares the audio player for playback
 * @param samplesPerBlockExpected Expected number of samples per audio block
 * @param sampleRate The sample rate for audio processing
 */
void prepareToPlay(int samplesPerBlockExpected, double sampleRate) override;

/**
 * Fills the audio buffer with the next block of audio data
 * @param bufferToFill Audio buffer to be filled with audio samples
 */
void getNextAudioBlock(const AudioSourceChannelInfo &bufferToFill) override;

/**
 * Releases audio resources when playback is stopped
 */
void releaseResources() override;

/**
 * Loads an audio file from the specified URL
 * @param audioURL URL pointing to the audio file to load
 */
void loadURL(URL audioURL);

/**
 * Sets the gain (volume) of the audio player
 * @param gain Gain value
 */
void setGain(double gain);

/**
 * Sets the playback speed ratio
 * @param ratio Speed ratio
 */
void setSpeed(double ratio);

/**
 * Sets the playback position in seconds
 * @param posInSecs Position in seconds from the start of the audio file
 */
void setPosition(double posInSecs);

/**
 * Sets the playback position as a relative value
 * @param pos Relative position
 */
void setPositionRelative(double pos);

/**
 * Starts audio playback
 */
void start();

/**
 * Stops audio playback
 */
void stop();

/**
 * Gets the current playback position as a relative value
 * @return Relative position
```

```
      */
  double getPositionRelative();

private:
  AudioFormatManager &formatManager;
  std::unique_ptr<AudioFormatReaderSource> readerSource;
  AudioTransportSource transportSource;
  ResamplingAudioSource resampleSource{&transportSource, false, 2};
};
```

DJAudioPlayer.cpp

```
/*
  ==============================================================================

    DJAudioPlayer.cpp
    Created: 10 Aug 2025 8:35:47am
    Author:  joao

  ==============================================================================
*/

#include "DJAudioPlayer.h"

DJAudioPlayer::DJAudioPlayer(AudioFormatManager &_formatManager)
    : formatManager(_formatManager) {}

DJAudioPlayer::~DJAudioPlayer() {}

void DJAudioPlayer::prepareToPlay(int samplesPerBlockExpected,
                                  double sampleRate) {
  transportSource.prepareToPlay(samplesPerBlockExpected, sampleRate);
  resampleSource.prepareToPlay(samplesPerBlockExpected, sampleRate);
}

void DJAudioPlayer::getNextAudioBlock(
    const AudioSourceChannelInfo &bufferToFill) {
  resampleSource.getNextAudioBlock(bufferToFill);
}

void DJAudioPlayer::releaseResources() {
  transportSource.releaseResources();
  resampleSource.releaseResources();
}

void DJAudioPlayer::loadURL(URL audioURL) {
  auto *reader =
      formatManager.createReaderFor(audioURL.createInputStream(false));
  if (reader != nullptr) // good file!
  {
    std::unique_ptr<AudioFormatReaderSource> newSource(
        new AudioFormatReaderSource(reader, true));
    transportSource.setSource(newSource.get(), 0, nullptr, reader->sampleRate);
    readerSource.reset(newSource.release());
  }
}

void DJAudioPlayer::setGain(double gain) {
  if (gain < 0 || gain > 1.0) {
```

```cpp
      std::cout << "DJAudioPlayer::setGain gain should be between 0 and 1"
                << std::endl;
  } else {
    transportSource.setGain(gain);
  }
}

void DJAudioPlayer::setSpeed(double ratio) {
  if (ratio < 0 || ratio > 100.0) {
    std::cout << "DJAudioPlayer::setSpeed ratio should be between 0 and 100"
                << std::endl;
  } else {
    resampleSource.setResamplingRatio(ratio);
  }
}

void DJAudioPlayer::setPosition(double posInSecs) {
  transportSource.setPosition(posInSecs);
}

void DJAudioPlayer::setPositionRelative(double pos) {
  if (pos < 0 || pos > 1.0) {
    std::cout
        << "DJAudioPlayer::setPositionRelative pos should be between 0 and 1"
        << std::endl;
  } else {
    double posInSecs = transportSource.getLengthInSeconds() * pos;
    setPosition(posInSecs);
  }
}

void DJAudioPlayer::start() { transportSource.start(); }

void DJAudioPlayer::stop() { transportSource.stop(); }

double DJAudioPlayer::getPositionRelative() {
  double lengthInSeconds = transportSource.getLengthInSeconds();

  // Prevent division by zero and ensure valid values
  if (lengthInSeconds <= 0.0) {
    return 0.0;
  }

  double currentPosition = transportSource.getCurrentPosition();
  double relativePosition = currentPosition / lengthInSeconds;

  // Clamp the result to valid range [0.0, 1.0]
  if (relativePosition < 0.0)
    return 0.0;
  if (relativePosition > 1.0)
    return 1.0;

  return relativePosition;
}
```

MixerComponent.h

```
/*
  ==============================================================================
```

```cpp
    MixerComponent.h
    Created: 19 Aug 2025 9:05:39am
    Author:  joao


  ==============================================================================
*/

#pragma once

#include "../JuceLibraryCode/JuceHeader.h"
#include <functional>

//==============================================================================

/**
 * This class was written by me without assistance based on the code provided in
 * the module, the juce documentation and tutorials about how to use the juce
 * library.
 */

/**
 * MixerComponent - Mixer interface with crossfader and master controls
 * This mixes the audio from the two decks and allows the user to control
 * the master volume.
 * Uses callback pattern to communicate mixer changes to the main audio
 * component
 */
class MixerComponent : public Component, public Slider::Listener {
public:
  /**
   * Constructor - Initializes the mixer component with default values
   * Sets up the crossfader and master volume sliders with appropriate ranges
   * and labels for the DJ mixing interface
   */
  MixerComponent();

  /**
   * Destructor
   */
  ~MixerComponent();

  /**
   * Paint method - Renders the mixer component's visual representation
   * @param g - Graphics context for drawing operations
   */
  void paint(Graphics &g) override;

  /**
   * Resized method - Handles component size changes and repositioning
   * Arranges the crossfader and master volume controls when the component
   * dimensions change.
   */
  void resized() override;

  /**
   * Slider value change handler - Responds to user interactions with sliders.
   * Handles crossfader and master volume slider movements and trigger
   * appropriate callbacks
   * @param slider - Pointer to the slider that triggered the change
```

```cpp
 */
void sliderValueChanged(Slider *slider) override;

/**
 * Sets the crossfader value
 * @param value - Crossfader position
 */
void setCrossfaderValue(double value);

/**
 * Retrieves the current crossfader position value
 * @return Current crossfader value
 */
double getCrossfaderValue() const;

/**
 * Sets the master volume level for the entire mix
 * @param volume - Master volume level
 */
void setMasterVolume(double volume);

/**
 * Retrieves the current master volume level
 * @return Current master volume value
 */
double getMasterVolume() const;

/**
 * Registers a callback function for crossfader value changes.
 * @param callback - Function to be called when crossfader value changes
 */
void setCrossfaderCallback(std::function<void(double)> callback);

/**
 * Registers a callback function for master volume changes.
 * @param callback - Function to be called when master volume changes
 */
void setMasterVolumeCallback(std::function<void(double)> callback);

private:
  Slider crossfader;
  Slider masterVolume;

  Label crossfaderLabel;
  Label masterVolumeLabel;

  // Callback functions for mixer control changes
  std::function<void(double)> crossfaderCallback;
  std::function<void(double)> masterVolumeCallback;

  JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(MixerComponent)
};
```

MixerComponent.cpp

```cpp
/*
  ==============================================================================

    MixerComponent.cpp
```

```
    Created: 19 Aug 2025 9:05:39am
    Author:  joao

  ==============================================================================
*/

#include "MixerComponent.h"

/**
 * This class was written by me without assistance based on the code provided in
 * the module, the juce documentation and tutorials about how to use the juce
 * library.
 */

//==============================================================================
MixerComponent::MixerComponent() {
  // Setup crossfader
  addAndMakeVisible(crossfader);
  crossfader.setSliderStyle(Slider::LinearHorizontal);
  crossfader.setRange(0.0, 1.0, 0.01);
  crossfader.setValue(0.5); // Center position
  crossfader.addListener(this);

  // Setup master volume
  addAndMakeVisible(masterVolume);
  masterVolume.setSliderStyle(Slider::LinearVertical);
  masterVolume.setRange(0.0, 1.0, 0.01);
  masterVolume.setValue(0.7);
  masterVolume.addListener(this);

  // Setup crossfader label
  addAndMakeVisible(crossfaderLabel);
  crossfaderLabel.setText("Crossfader", dontSendNotification);
  crossfaderLabel.setJustificationType(Justification::centred);

  // Setup master volume label
  addAndMakeVisible(masterVolumeLabel);
  masterVolumeLabel.setText("Master", dontSendNotification);
  masterVolumeLabel.setJustificationType(Justification::centred);
}

MixerComponent::~MixerComponent() {}

void MixerComponent::paint(Graphics &g) {
  g.fillAll(Colours::darkgrey);

  g.setColour(Colours::white);
  g.drawRect(getLocalBounds(), 2);

  g.setColour(Colours::lightgrey);
  g.setFont(16.0f);
  g.drawText("MIXER", getLocalBounds().removeFromTop(30),
             Justification::centred, true);
}

void MixerComponent::resized() {
  auto area = getLocalBounds();
  area.removeFromTop(30);

  auto crossfaderArea = area.removeFromTop(area.getHeight() / 2);
```

```
    crossfaderLabel.setBounds(crossfaderArea.removeFromTop(20));
    crossfader.setBounds(crossfaderArea.reduced(10));

    masterVolumeLabel.setBounds(area.removeFromTop(20));
    masterVolume.setBounds(area.reduced(20, 10));
}

void MixerComponent::sliderValueChanged(Slider *slider) {
    if (slider == &crossfader) {
        if (crossfaderCallback) {
            crossfaderCallback(crossfader.getValue());
        }
    } else if (slider == &masterVolume) {
        if (masterVolumeCallback) {
            masterVolumeCallback(masterVolume.getValue());
        }
    }
}

void MixerComponent::setCrossfaderValue(double value) {
    crossfader.setValue(value, dontSendNotification);
}

double MixerComponent::getCrossfaderValue() const {
    return crossfader.getValue();
}

void MixerComponent::setMasterVolume(double volume) {
    masterVolume.setValue(volume, dontSendNotification);
}

double MixerComponent::getMasterVolume() const {
    return masterVolume.getValue();
}

void MixerComponent::setCrossfaderCallback(
    std::function<void(double)> callback) {
    crossfaderCallback = callback;
}

void MixerComponent::setMasterVolumeCallback(
    std::function<void(double)> callback) {
    masterVolumeCallback = callback;
}
```

PlaylistComponent.h

```
/*
  ==============================================================================

    PlaylistComponent.h
    Created: 15 Aug 2025 8:40:03am
    Author:  joao

  ==============================================================================
*/

#pragma once
```

```cpp
#include "DeckGUI.h"
#include <JuceHeader.h>
#include <string>
#include <vector>


//================================================================================
/*
 */


/**
 * This class was written by me without assistance based on the code provided in
 * the module, the juce documentation and tutorials about how to use the juce
 * library. I adjusted the PlaylistComponent that have been built during the
 * module and added the functionality to load and save the playlist to
 * persistent storage. I also added the buttons to load the tracks to the decks.
 */


/**
 * PlaylistComponent - A component that displays a playlist of audio tracks
 * and allows the user to load and assign them to the decks
 */
class PlaylistComponent : public juce::Component,
                          public juce::TableListBoxModel,
                          public juce::Button::Listener {
public:
  /**
   * Constructor
   * Initializes the playlist component and sets up the table, load button
   * and application properties
   */
  PlaylistComponent();

  /**
   * Destructor
   * Saves the current playlist to persistent storage
   */
  ~PlaylistComponent() override;

  /**
   * Paints the playlist component
   * @param Graphics - The graphics context to paint on
   */
  void paint(juce::Graphics &) override;

  /**
   * Handles the resizing of the playlist component
   */
  void resized() override;

  /**
   * Returns the number of rows in the playlist table
   * @return The number of rows in the playlist table
   */
  int getNumRows() override;

  /**
   * Paints the background of a specific row in the playlist table
   * @param Graphics - The graphics context to paint on
   * @param rowNumber - The row number of the cell
```

```cpp
 * @param width - The width of the cell
 * @param height - The height of the cell
 * @param rowIsSelected - Whether the row is selected
 */
void paintRowBackground(Graphics &, int rowNumber, int width, int height,
                        bool rowIsSelected) override;


/**
 * Paints the content of a specific cell in the playlist table
 * @param Graphics - The graphics context to paint on
 * @param rowNumber - The row number of the cell
 * @param columnId - The column ID of the cell
 * @param width - The width of the cell
 * @param height - The height of the cell
 * @param rowIsSelected - Whether the row is selected
 */
void paintCell(Graphics &, int rowNumber, int columnId, int width, int height,
               bool rowIsSelected) override;


/**
 * Refreshes the component for a specific cell in the playlist table
 * @param rowNumber - The row number of the cell
 * @param columnId - The column ID of the cell
 * @param isRowSelected - Whether the row is selected
 * @param existingComponentToUpdate - The existing component to update
 * @return The refreshed component
 */
Component *
refreshComponentForCell(int rowNumber, int columnId, bool isRowSelected,
                        Component *existingComponentToUpdate) override;


/**
 * Handles button click events from the playlist interface
 * @param button - The button that was clicked
 */
void buttonClicked(Button *button) override;


/**
 * Sets references to the two deck GUI components for track loading
 * @param deck1 - The first deck GUI component
 * @param deck2 - The second deck GUI component
 */
void setDeckReferences(DeckGUI *deck1, DeckGUI *deck2);


/**
 * Adds a new audio track to the playlist
 * @param audioFile - The audio file to add to the playlist
 */
void addTrack(const juce::File &audioFile);


/**
 * Clears all tracks from the playlist
 */
void clearTracks();


/**
 * Saves the current playlist to persistent storage
 */
void savePlaylist();
```

```cpp
    /**
     * Loads a previously saved playlist from persistent storage
     */
    void loadPlaylist();

private:
    TableListBox tableComponent;
    TextButton loadButton{"Load Track"};
    juce::FileChooser fileChooser{
        "Select an audio file...",
        juce::File::getSpecialLocation(juce::File::userMusicDirectory),
        "*.mp3;*.wav;"};

    struct TrackInfo {
        std::string title;
        juce::File filePath;
    };

    std::vector<TrackInfo> tracks;
    DeckGUI *deckGUI1;
    DeckGUI *deckGUI2;
    std::unique_ptr<ApplicationProperties> appProperties;

    JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(PlaylistComponent)
};
```

PlaylistComponent.cpp

```cpp
/*
  ==============================================================================

    PlaylistComponent.cpp
    Created: 15 Aug 2025 8:40:03am
    Author:  joao

  ==============================================================================
*/

#include "PlaylistComponent.h"
#include <JuceHeader.h>



//==============================================================================
/**
 * This class was written by me without assistance based on the code provided in
 * the module, the juce documentation and tutorials about how to use the juce
 * library. I adjusted the PlaylistComponent that have been built during the
 * module and added the functionality to load and save the playlist to
 * persistent storage. I also added the buttons to load the tracks to the decks.
 */
PlaylistComponent::PlaylistComponent() : deckGUI1(nullptr), deckGUI2(nullptr) {
    PropertiesFile::Options options;
    options.applicationName = "OtoDecks";
    options.filenameSuffix = ".settings";
    options.osxLibrarySubFolder = "Application Support";

    appProperties.reset(new ApplicationProperties());
    appProperties->setStorageParameters(options);
```

```cpp
  tableComponent.getHeader().addColumn("Deck 1", 1, 80);
  tableComponent.getHeader().addColumn("Track Title", 2, 240);
  tableComponent.getHeader().addColumn("Deck 2", 3, 80);

  tableComponent.setModel(this);
  addAndMakeVisible(tableComponent);

  addAndMakeVisible(loadButton);
  loadButton.addListener(this);

  loadPlaylist();
}

PlaylistComponent::~PlaylistComponent() { savePlaylist(); }

void PlaylistComponent::paint(juce::Graphics &g) {
  g.fillAll(
      getLookAndFeel().findColour(juce::ResizableWindow::backgroundColourId));

  g.setColour(juce::Colours::grey);
  g.drawRect(getLocalBounds(), 1);
}

void PlaylistComponent::resized() {
  auto area = getLocalBounds();

  auto buttonArea = area.removeFromTop(30);
  loadButton.setBounds(buttonArea.reduced(5));

  tableComponent.setBounds(area);
}

int PlaylistComponent::getNumRows() { return tracks.size(); }

void PlaylistComponent::paintRowBackground(Graphics &g, int rowNumber,
                                           int width, int height,
                                           bool rowIsSelected) {
  if (rowIsSelected) {
    g.fillAll(Colours::orange);
  } else {
    g.fillAll(Colours::darkgrey);
  }
}

void PlaylistComponent::paintCell(Graphics &g, int rowNumber, int columnId,
                                  int width, int height, bool rowIsSelected) {
  if (columnId == 2) {
    g.drawText(tracks[rowNumber].title, 2, 0, width - 4, height,
               Justification::centredLeft);
  }
}

Component *PlaylistComponent::refreshComponentForCell(
    int rowNumber, int columnId, bool isRowSelected,
    Component *existingComponentToUpdate) {
  if (columnId == 1 || columnId == 3) {
    Component *container = existingComponentToUpdate;
    TextButton *button = nullptr;

    if (container == nullptr) {
```

```cpp
      container = new Component();
      button = new TextButton();
      button->addListener(this);
      container->addAndMakeVisible(button);
    } else {
      button = dynamic_cast<TextButton *>(container->getChildComponent(0));
    }

    if (button != nullptr) {
      if (columnId == 1) {
        button->setButtonText("Load");
        button->setComponentID(String(rowNumber) + "_left");
      } else {
        button->setButtonText("Load");
        button->setComponentID(String(rowNumber) + "_right");
      }

      auto containerBounds =
          Rectangle<int>(0, 0, 80, tableComponent.getRowHeight());
      container->setSize(containerBounds.getWidth(),
                         containerBounds.getHeight());
      button->setBounds(containerBounds.reduced(1));
    }

    return container;
  }

  return nullptr;
}

void PlaylistComponent::buttonClicked(Button *button) {
  if (button == &loadButton) {
    fileChooser.launchAsync(FileBrowserComponent::openMode |
                                FileBrowserComponent::canSelectFiles,
                            [this](const FileChooser &chooser) {
                              auto results = chooser.getResults();
                              if (results.size() > 0) {
                                addTrack(results[0]);
                              }
                            });
    return;
  }

  String componentId = button->getComponentID();
  int rowNumber =
      componentId.upToFirstOccurrenceOf("_", false, false).getIntValue();
  bool isLeftDeck = componentId.contains("left");

  if (rowNumber >= 0 && rowNumber < tracks.size()) {
    if (isLeftDeck && deckGUI1 != nullptr) {
      if (tracks[rowNumber].filePath.existsAsFile()) {
        deckGUI1->loadTrack(tracks[rowNumber].filePath);
        DBG("Loading " + tracks[rowNumber].title + " into left deck");
      } else {
        DBG("Track file does not exist: " + tracks[rowNumber].title);
      }
    } else if (!isLeftDeck && deckGUI2 != nullptr) {
      if (tracks[rowNumber].filePath.existsAsFile()) {
        deckGUI2->loadTrack(tracks[rowNumber].filePath);
        DBG("Loading " + tracks[rowNumber].title + " into right deck");
```

```cpp
    } else {
      DBG("Track file does not exist: " + tracks[rowNumber].title);
    }
  }
 }
}

void PlaylistComponent::setDeckReferences(DeckGUI *deck1, DeckGUI *deck2) {
  deckGUI1 = deck1;
  deckGUI2 = deck2;
}

void PlaylistComponent::addTrack(const juce::File &audioFile) {
  if (audioFile.existsAsFile()) {
    TrackInfo newTrack;
    newTrack.title = audioFile.getFileNameWithoutExtension().toStdString();
    newTrack.filePath = audioFile;
    tracks.push_back(newTrack);
    tableComponent.updateContent();

    savePlaylist();
  }
}

void PlaylistComponent::clearTracks() {
  tracks.clear();
  tableComponent.updateContent();

  savePlaylist();
}

void PlaylistComponent::savePlaylist() {
  if (appProperties != nullptr) {
    auto *userSettings = appProperties->getUserSettings();
    if (userSettings != nullptr) {
      int currentPlaylistSize = userSettings->getIntValue("playlistSize", 0);

      userSettings->removeValue("playlistSize");
      for (int i = 0; i < currentPlaylistSize; ++i) {
        userSettings->removeValue("track_" + String(i) + "_title");
        userSettings->removeValue("track_" + String(i) + "_path");
      }

      userSettings->setValue("playlistSize", (int)tracks.size());
      for (int i = 0; i < tracks.size(); ++i) {
        userSettings->setValue("track_" + String(i) + "_title",
                               String(tracks[i].title));
        userSettings->setValue("track_" + String(i) + "_path",
                               tracks[i].filePath.getFullPathName());
      }

      appProperties->saveIfNeeded();
      DBG("Playlist saved with " + String(tracks.size()) + " tracks");
    }
  }
}

void PlaylistComponent::loadPlaylist() {
  if (appProperties != nullptr) {
    auto *userSettings = appProperties->getUserSettings();
```

```cpp
  if (userSettings != nullptr) {
    tracks.clear();

    int playlistSize = userSettings->getIntValue("playlistSize", 0);
    for (int i = 0; i < playlistSize; ++i) {
      String title = userSettings->getValue("track_" + String(i) + "_title");
      String path = userSettings->getValue("track_" + String(i) + "_path");

      if (title.isNotEmpty() && path.isNotEmpty()) {
        File audioFile(path);
        if (audioFile.existsAsFile()) {
          TrackInfo track;
          track.title = title.toStdString();
          track.filePath = audioFile;
          tracks.push_back(track);
        } else {
          DBG("Saved track file no longer exists: " + path);
        }
      }
    }

    tableComponent.updateContent();
    DBG("Playlist loaded with " + String(tracks.size()) + " tracks");
  }
}
}
```

WaveformDisplay.h

```cpp
/*
  ==============================================================================

    WaveformDisplay.h
    Created: 10 Aug 2025 8:37:18am
    Author:  joao

  ==============================================================================
*/

#pragma once

#include "../JuceLibraryCode/JuceHeader.h"

//==============================================================================
/*
 */
class WaveformDisplay : public Component, public ChangeListener {
public:
  WaveformDisplay(AudioFormatManager &formatManagerToUse,
                  AudioThumbnailCache &cacheToUse);
  ~WaveformDisplay();

  void paint(Graphics &) override;
  void resized() override;

  void changeListenerCallback(ChangeBroadcaster *source) override;

  void loadURL(URL audioURL);
```

```cpp
  /** set the relative position of the playhead*/
  void setPositionRelative(double pos);

private:
  AudioThumbnail audioThumb;
  bool fileLoaded;
  double position;

  JUCE_DECLARE_NON_COPYABLE_WITH_LEAK_DETECTOR(WaveformDisplay)
};
```

WaveformDisplay.cpp

```cpp
/*
  ==============================================================================

    WaveformDisplay.cpp
    Created: 10 Aug 2025 8:37:18am
    Author:  joao


  ==============================================================================
*/

#include "WaveformDisplay.h"
#include "../JuceLibraryCode/JuceHeader.h"


//==============================================================================
WaveformDisplay::WaveformDisplay(AudioFormatManager &formatManagerToUse,
                                 AudioThumbnailCache &cacheToUse)
    : audioThumb(1000, formatManagerToUse, cacheToUse), fileLoaded(false),
      position(0)

{
  // In your constructor, you should add any child components, and
  // initialise any special settings that your component needs.

  audioThumb.addChangeListener(this);
}

WaveformDisplay::~WaveformDisplay() {}

void WaveformDisplay::paint(Graphics &g) {
  /* This demo code just fills the component's background and
     draws some placeholder text to get you started.

     You should replace everything in this method with your own
     drawing code..
  */

  g.fillAll(getLookAndFeel().findColour(
      ResizableWindow::backgroundColourId)); // clear the background

  g.setColour(Colours::grey);
  g.drawRect(getLocalBounds(), 1); // draw an outline around the component

  g.setColour(Colours::orange);
  if (fileLoaded) {
    audioThumb.drawChannel(g, getLocalBounds(), 0, audioThumb.getTotalLength(),
```

```cpp
                          0, 1.0f);
    g.setColour(Colours::lightgreen);
    g.drawRect(position * getWidth(), 0, getWidth() / 20, getHeight());
  } else {
    g.setFont(20.0f);
    g.drawText("File not loaded...", getLocalBounds(), Justification::centred,
               true); // draw some placeholder text
  }
}

void WaveformDisplay::resized() {
  // This method is where you should set the bounds of any child
  // components that your component contains..
}

void WaveformDisplay::loadURL(URL audioURL) {
  audioThumb.clear();
  fileLoaded = audioThumb.setSource(new URLInputSource(audioURL));
  if (fileLoaded) {
    std::cout << "wfd: loaded! " << std::endl;
    repaint();
  } else {
    std::cout << "wfd: not loaded! " << std::endl;
  }
}

void WaveformDisplay::changeListenerCallback(ChangeBroadcaster *source) {
  std::cout << "wfd: change received! " << std::endl;

  repaint();
}

void WaveformDisplay::setPositionRelative(double pos) {
  if (pos != position) {
    position = pos;
    repaint();
  }
}
```