

Project Summary

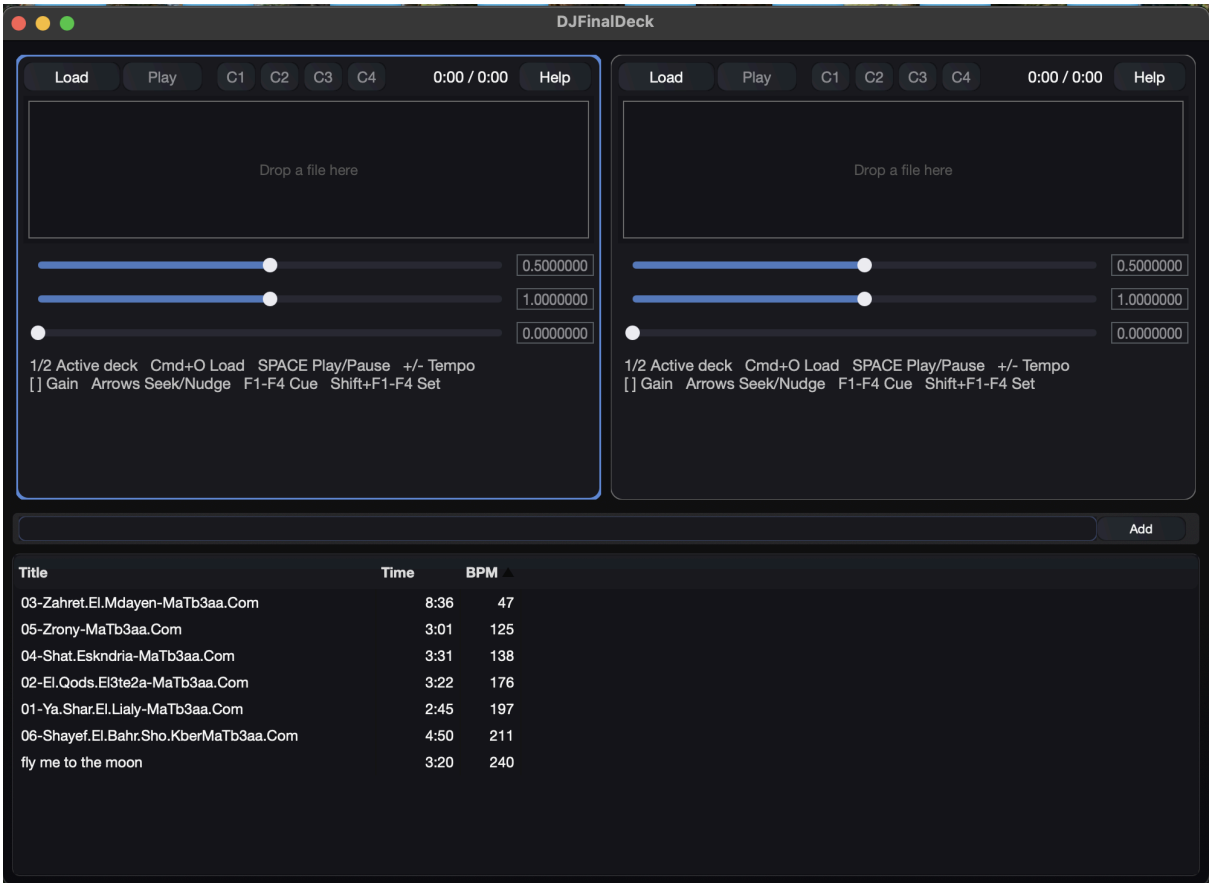
I implemented “DJFinalDeck” to meet the assessment requirements for this module by delivering a two-deck audio application with an integrated music library, a customised user interface, and one researched feature from industry software. The feature I selected is hot cues, inspired by VirtualDJ, and implemented end-to-end with keyboard control, mouse interaction, waveform markers and library persistence. I also added a lightweight Beats Per Minute (BPM) estimation pipeline to enrich the playlist. The primary target platform is macOS, with builds that run on Linux and Windows.

R1-A : App entry & launch from IDE

```
'''cpp (Main.cpp)
// In MainWindow ctor:
setContentOwned (new MainComponent(), true);

// At end of file:
START_JUCE_APPLICATION (DJFinalDeckApplication)

'''
```



Screenshot A: Overall application layout with two decks on top and playlist at the bottom

R1. How the basic program works

Architecture and audio flow:

The application hosts two deck UIs, each backed by a `DJAudioPlayer`, and mixes them through a `juce::MixerAudioSource`. The `MainComponent` wires the players into the mixer, starts audio, and lays out the UI with the decks above the playlist. The mixer pulls from both players to produce the final stereo output, which satisfies multi-track playback and volume-based mixing through per-deck gain.

R1-B : Audio I/O and mixer routing

```
'''cpp (MainComponent.cpp)
    // Audio device: 0 inputs, 2 outputs
    setAudioChannels (0, 2);

    // Mixer: combine both players
    mixer_.addInputSource (&player1_, false);
    mixer_.addInputSource (&player2_, false);
'''
```

Each deck exposes transport and performance controls:

- Load audio by file chooser, drag-and-drop from Finder, or internal drags from the playlist.
- Play and pause via the button and space bar.
- Gain, tempo and seek controls through sliders and keyboard shortcuts.

The deck UI owns a `WaveformDisplay` and drives it from the player's transport position. A high-frequency timer syncs the time readout and waveform cursor for responsive visual feedback.

File formats and loading:

`DJAudioPlayer::loadURL` uses JUCE's `AudioFormatManager` to read common formats; transport and resampler are prepared, and the deck UI is reset on successful load. Supported extensions include mp3, wav, aiff, flac, ogg and oga.

Playback, speed and gain:

The player wraps a `AudioTransportSource` for playback and a `ResamplingAudioSource` for tempo changes. Gain is applied on the transport, and tempo is mapped to the resampling ratio; the UI binds sliders and keyboard nudges to these methods.

Keyboard control:

I routed shortcuts through `MainComponent::keyPressed` so a single active deck consumes transport and performance keys, whereas active-deck selection uses 1 and 2. Space toggles play and pause. Left and right seek, while up and down nudge by 0.25 s. Square brackets trim gain, plus and minus change tempo.

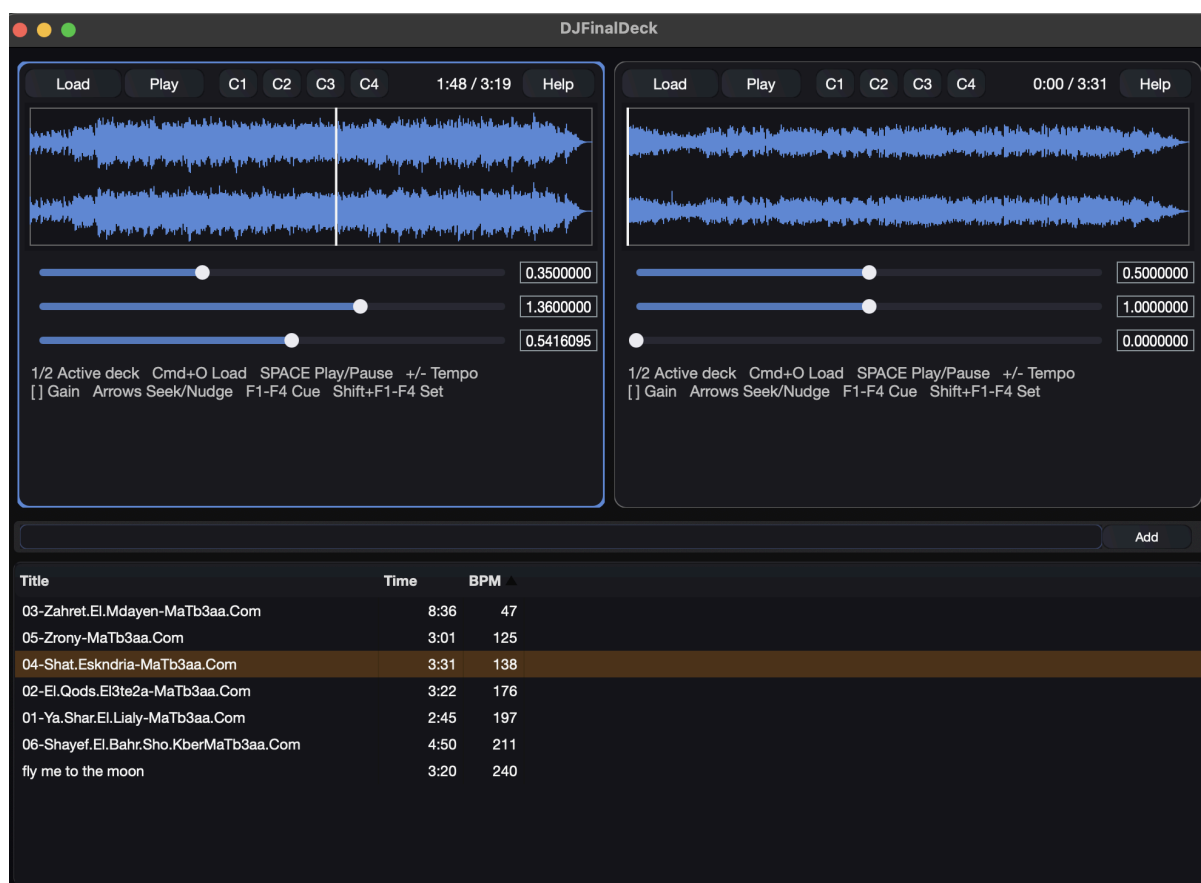
R1-C : GUI and audio wiring

```
'''cpp (MainComponent.cpp)
// Gain
gainSlider_.onValueChange      = [this] { player_.setGain      ((double)
gainSlider_.getValue()); };

// Speed (tempo)
speedSlider_.onValueChange     = [this] { player_.setSpeed     ((double)
speedSlider_.getValue()); };

// Position (0..1) + sync waveform
posSlider_.onValueChange      = [this]
{
    player_.setPositionRelative ((double) posSlider_.getValue());
    waveform_.setPositionRelative ((double) posSlider_.getValue());
};

'''
```



Screenshot B: A deck while playing, with time display, waveform and cue buttons

R2. Music library: implementation and persistence

Data model and persistence format:

The music library lives in `PlaylistComponent`. Each row is a `TrackItem` with fields for path, title, duration, BPM and four cue times in seconds. The in-memory vector is mirrored by a `juce::ValueTree` named `library`, which is serialised to a `PropertiesFile` as a compact XML string for durability across sessions.

The app stores data in `../DJFinalDeck/Data/library.properties` and also persists UI preferences, such as search text, sort order and column widths.

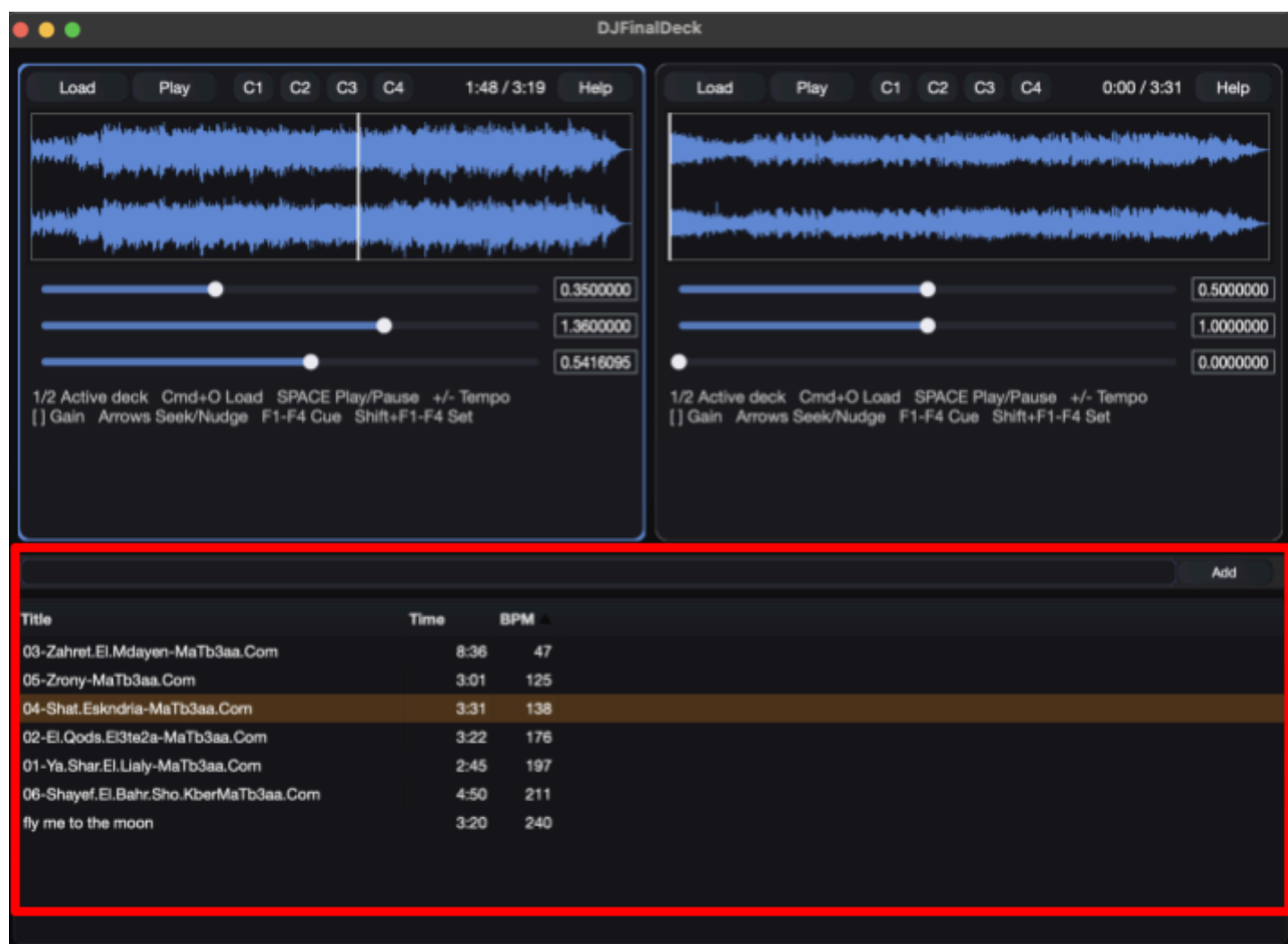
R2-A : Persistence location (on disk)

```
'''cpp (PlaylistComponent.cpp)
    dataDir          =          juce::File          (juce::File::getSpecialLocation
(juce::File::userHomeDirectory)
                    .getChildFile ("Downloads")
                    .getChildFile ("DJFinalDeck")
                    .getChildFile ("Data"));
    dataDir.createDirectory();

    const auto propsFile = dataDir.getChildFile ("library.properties");
    props          =          std::make_unique<juce::PropertiesFile>          (propsFile,
juce::PropertiesFile::Options{});

'''
```

On startup, the component creates the data directory, opens the properties file, rebuilds the library tree and table, and restores header sizes and saved search. On shutdown or after edits such as cue changes, it writes the tree back to disk so state persists.



Screenshot C: Playlist showing Title, Time and BPM columns with the search box

R2-B : Model schema & stored fields (including cues)

```
'''cpp (PlaylistComponent.cpp)
    juce::ValueTree node ("track");
    node.setProperty ("id",          t.id,          nullptr);
    node.setProperty ("file",        canonical,      nullptr);
    node.setProperty ("title",       t.title,        nullptr);
    node.setProperty ("durationSec", t.durationSeconds, nullptr);
    node.setProperty ("bpm",         t.bpm,          nullptr);
    node.setProperty ("cue1Sec",     t.cue1Sec,       nullptr);
    node.setProperty ("cue2Sec",     t.cue2Sec,       nullptr);
    node.setProperty ("cue3Sec",     t.cue3Sec,       nullptr);
    node.setProperty ("cue4Sec",     t.cue4Sec,       nullptr);
    libraryTree.addChild (node, -1, nullptr);

'''
```

Loading, browsing and sending tracks to decks:

- **Adding tracks:** The Add button launches a multi-select chooser for supported audio types, deduplicates, and inserts tracks into the model and tree. Double-clicking or pressing return loads the selected row to the active deck through a callback provided by `MainComponent`. Dragging rows internally sends the file path as the drag description so a deck can accept the drop. When a deck loads a file via any route, it signals the playlist to auto-add the track.

R2-C : Load from library into the active deck & app-level hook

```
'''cpp (PlaylistComponent.cpp)
    void PlaylistComponent::cellDoubleClicked (int    rowNumber,    int,    const
    juce::MouseEvent&)
    {
        loadRowToActiveDeck (rowNumber);
    }

'''

'''cpp (MainComponent.cpp)

    // Playlist double-click -> active deck
    playlist_.onRequestLoadToActiveDeck = [this](const juce::File& f) {
        activeDeck().loadFile (f); };

'''
```

- **Search and sort:** A search text editor filters the table and is persisted for convenience. Table header sorting is enabled for Title, Time and BPM; column widths persist using a header listener.

R2-D : Column width persistence (UI behaviour)

```
'''cpp (PlaylistComponent.cpp)
    table.getHeader().addListener (this);
    restoreColumnWidths();

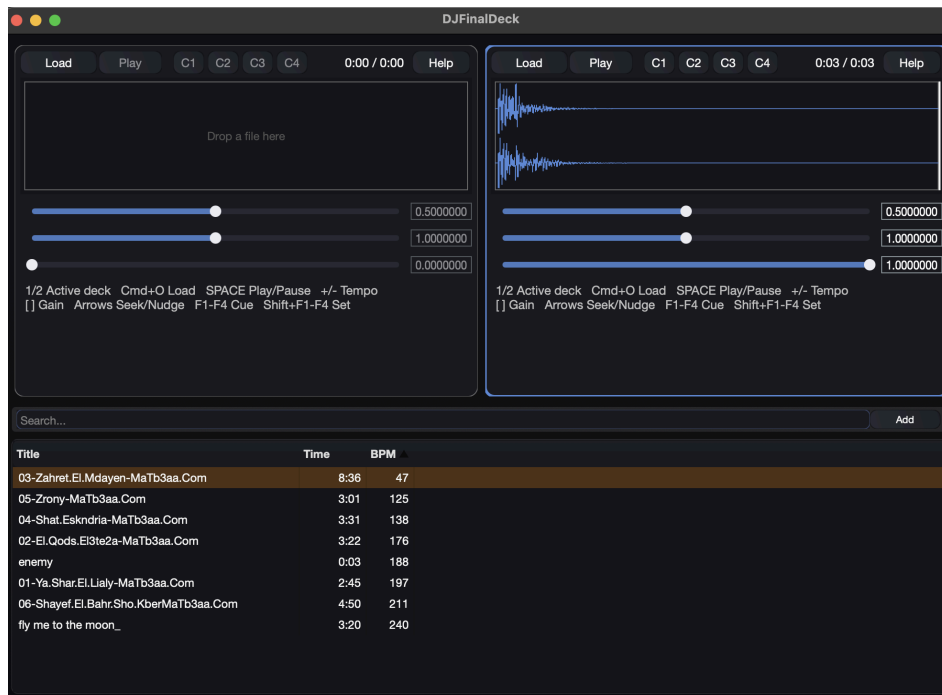
    // Persist on resize:
    void PlaylistComponent::tableColumnsResized (juce::TableHeaderComponent* header)
    {
        persistColumnWidth (colTitle, header->getColumnWidth (colTitle));
    }
```

```

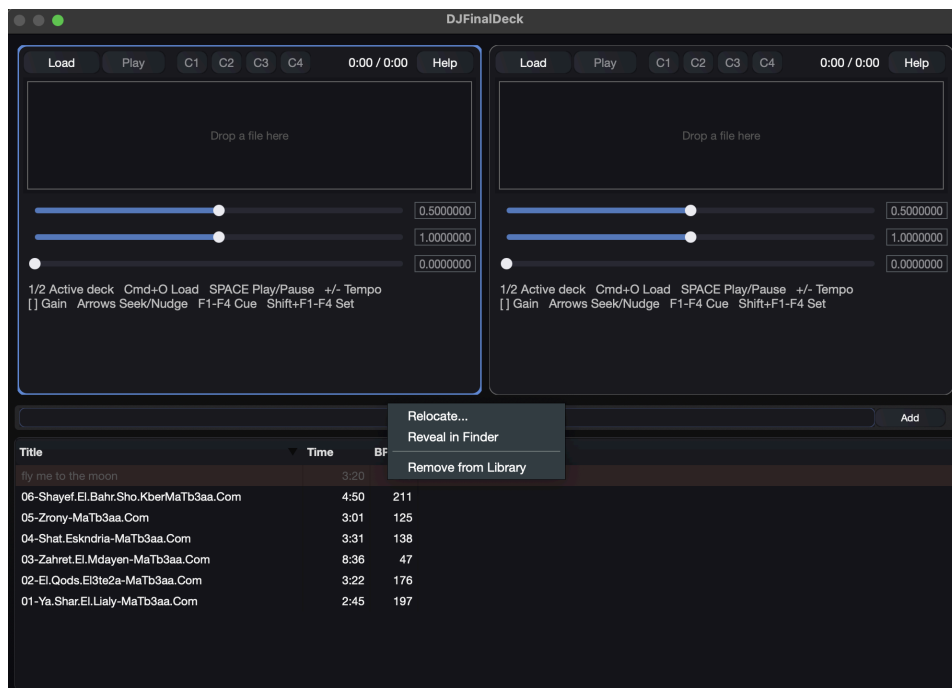
    persistColumnWidth (colTime, header->getColumnWidth (colTime));
    persistColumnWidth (colBPM, header->getColumnWidth (colBPM));
}

```

...



- **Deleting and repairing rows:** The context menu supports Reveal in Finder, Relocate, and Remove. Missing files are flagged on load and rendered distinctly in the list.

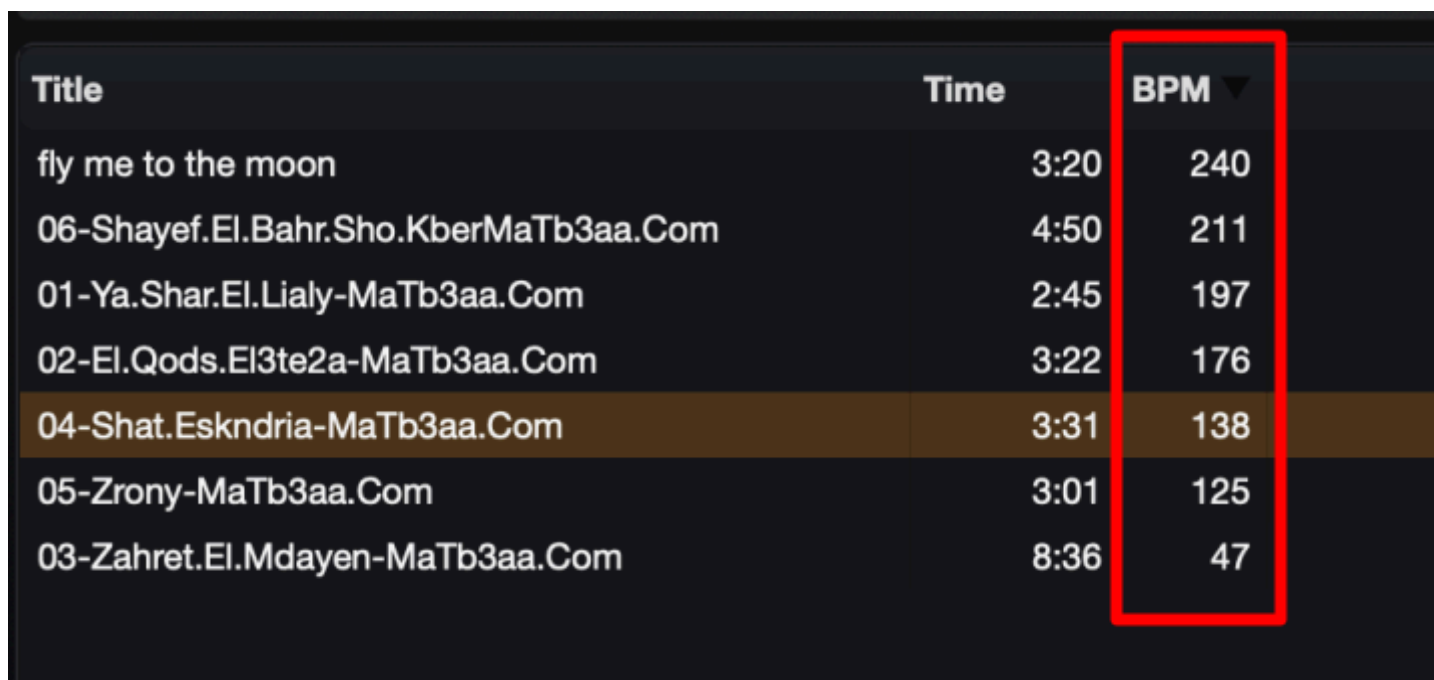


- **Tooltips and status:** BPM cells show informative tooltips that explain whether analysis is pending, unavailable due to unreadable audio or missing duration, or complete with a numeric value.

- **Keyboard forwarding from search:** While the search box has focus, a small `KeyListener` forwards global deck shortcuts, such as 1, 2, space, arrows, plus, minus and command-O, to `MainComponent`. This keeps typing natural while preserving performance control.

BPM estimation pipeline:

I implemented a lightweight, offline BPM estimate per track. The pipeline reads up to one minute of audio, downsamples the rectified envelope to a 200 Hz target rate, emphasises onsets with a first-difference high-pass, then performs a simple autocorrelation over lags that map to 40 through 240 BPM. The best lag converts to BPM and is folded into the display window. Additionally, cells report status while work is pending.



Title	Time	BPM ▼
fly me to the moon	3:20	240
06-Shayef.El.Bahr.Sho.KberMaTb3aa.Com	4:50	211
01-Ya.Shar.El.Lialy-MaTb3aa.Com	2:45	197
02-El.Qods.El3te2a-MaTb3aa.Com	3:22	176
04-Shat.Eskndria-MaTb3aa.Com	3:31	138
05-Zrony-MaTb3aa.Com	3:01	125
03-Zahret.El.Mdayen-MaTb3aa.Com	8:36	47

Screenshot D: Playlist with BPM values filled for analysed tracks

R3. User interface customisation

I delivered a cohesive dark theme and a new layout that differs from the classroom baseline.

Look and Feel: I created `AppLookAndFeel`, which sets dark tokens for windows, text editors, sliders, buttons and tables; draws custom button backgrounds with rounded gradients; renders sliders with a filled track and circular thumb; provides a focused outline for text editors; and draws a rounded, shaded header background in the table. `MainComponent` sets this as the app default.

Layout and visual design: Each deck is styled as a card with a subtle active-deck border and a compact top row of controls, followed by a waveform and three aligned sliders. A two-line legend summarises shortcuts, and a time label shows elapsed and total. The overall window places decks on the top half and the library below with gentle gradients and consistent spacing.

Event listener beyond the baseline: The `SearchKeyForwarder` described in R2 is an extra event listener that improves focus behaviour and satisfies the requirement to add new event-driven UI logic.

R3-A : Custom LookAndFeel (non-stock visuals)

```
'''cpp (MainComponent.h)
class AppLookAndFeel : public juce::LookAndFeel_V4 {
public:
    AppLookAndFeel() {
        using namespace juce;
        setColour (ResizableWindow::backgroundColourId,   Colour::fromRGB
(18,18,22));
        setColour (Slider::backgroundColourId,             Colour::fromRGBA
(40,44,52,255));
        setColour (Slider::trackColourId,                  Colour::fromRGBA
(85,140,220,210));
        setColour (Slider::thumbColourId,                  Colour::fromRGBA
(235,235,245,255));
        setColour (TableHeaderComponent::textColourId,    Colours::gainsboro);
    }
    void drawLinearSlider (juce::Graphics& g, int x, int y, int w, int h,
        float sliderPos, float, float,
        const juce::Slider::SliderStyle, juce::Slider& s)
override {
        auto track = juce::Rectangle<float> ((float)x, (float)(y + h/2 - 3),
(float)w, 6.0f);
        g.setColour (s.findColour (juce::Slider::backgroundColourId));
g.fillRoundedRectangle (track, 3.0f);
        auto fill = juce::Rectangle<float> ((float)x, track.getY(), sliderPos - x,
track.getHeight());
        g.setColour (s.findColour (juce::Slider::trackColourId));
g.fillRoundedRectangle (fill, 3.0f);
        g.setColour (s.findColour (juce::Slider::thumbColourId)); g.fillEllipse
(sliderPos - 6.0f, track.getCentreY() - 6.0f, 12.0f, 12.0f);
    }
};

...
'''
```

R3-B : Layout (two decks over playlist)

```
'''cpp (MainComponent.h)
void MainComponent::resized()
{
    auto r = getLocalBounds().reduced (8);

    auto top = r.removeFromTop (r.proportionOfHeight (0.55f));
    deck1_.setBounds (top.removeFromLeft (top.getWidth() / 2).reduced (4));
    deck2_.setBounds (top.reduced (4));

    r.removeFromTop (6);
    playlist_.setBounds (r);
}

...
'''
```

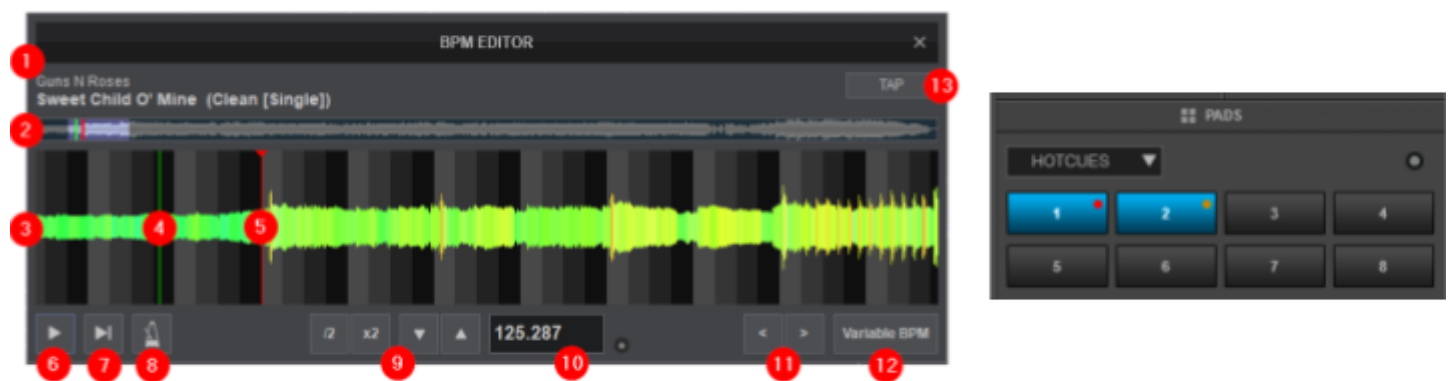

R4. Researched feature: hot cues

Research:

I investigated hot cues in VirtualDJ 8.0.8. The manual describes numbered hot cues that can be set when empty and triggered when present, with extended views offering multiple pads and the ability to manage them in a Points Of Interest (POI) editor. Official documentation also states that VirtualDJ supports a large number of cue points per track. ([VirtualDJ Website](#))

Controller documentation and the on-screen pads show a consistent interaction model: press a pad to assign or jump to cues 1 through 8, with a common workflow that uses the Shift modifier to delete an assigned hot cue. ([VirtualDJ Website](#), [VirtualDJ Website](#))

I also reviewed VirtualDJ's BPM analysis facilities. The manual and knowledge base describe automatic analysis with an editor and batch operations in the browser, which informed the scope and affordances for my lighter-weight estimate. ([VirtualDJ Website](#))



Screenshot E: VirtualDJ hotcues pads and POI editor

Analysis and rationale

My goal was a minimal, keyboard-friendly subset that fits a compact two-deck UI and complements coursework constraints. I selected four persistent hot cues per track and mapped them to F1 through F4 to jump. I added a set mode via Shift to write the current playhead to the selected slot. The deck shows C1 through C4 buttons with tooltips and a waveform overlay to anchor the cues visually. The library stores cue locations per track, so the same cues appear when the file is reloaded or used on the other deck.

This model mirrors VirtualDJ's numbered hot cues and pad-style interaction while using a keyboard mapping that suits a laptop without a controller. The Shift semantics match industry practice for set or delete operations and keep accidental editing under control. ([VirtualDJ Website](#), [VirtualDJ Website](#))

R4-A : Hot Cue logic (set vs jump)

```
'''cpp (DeckGUI.cpp)
void DeckGUI::handleCueClick (int cueIndex, bool setMode)
{
    if (! hasTrack_ || cueIndex < 1 || cueIndex > 4) return;

    const double len = player_.getLengthInSeconds();
    if (len <= 0.0) return;

    if (setMode)
    {
        const double curSec = player_.getPositionSeconds();
```

```

        cueSec_[(size_t) (cueIndex - 1)] = curSec;

        if (commitCueForFile && currentFile_.existsAsFile())
            commitCueForFile (currentFile_, cueIndex, curSec);

        waveform_.setCueMarkers (cueSec_);
        repaint();
    }
    else
    {
        const double sec = cueSec_[(size_t) (cueIndex - 1)];
        if (sec >= 0.0 && sec <= len)
        {
            const double rel = (len > 0.0 ? sec / len : 0.0);
            posSlider_.setValue (juce::jlimit (0.0, 1.0, rel));
        }
    }
}

'''

```

Implementation:

Data flow: `MainComponent` wires the deck cue APIs to the library. When a deck requests cues for a newly loaded file, the playlist returns the four times from memory or the persisted tree; when a user sets a cue, the deck calls back to commit the change, which updates both the in-memory model and the `ValueTree`, then writes to disk.

Deck behaviour: Clicking C1 to C4 jumps to an existing cue; holding Shift while clicking sets the cue to the current position and updates markers. Keyboard shortcuts F1 to F4 act the same; holding Shift switches into set mode. The timer keeps the waveform cursor and time label in sync during playback.

R4-B : Keyboard shortcuts for cues

```

'''cpp (MainComponent.cpp)
// ===== Cue shortcuts =====
const bool setMode = key.getModifiers().isShiftDown();
const int kc = key.getKeyCode();
if (kc == KeyPress::F1Key) { if (! ready) return true;
activeDeck().triggerCueShortcut (1, setMode); return true; }
if (kc == KeyPress::F2Key) { if (! ready) return true;
activeDeck().triggerCueShortcut (2, setMode); return true; }
if (kc == KeyPress::F3Key) { if (! ready) return true;
activeDeck().triggerCueShortcut (3, setMode); return true; }
// (F4 identical to F1-F3)

'''

```

Waveform integration: The deck requests cues after a successful load and forwards them to the waveform to render markers. Markers update immediately when a cue is set.

R4-C : Waveform cue markers (visual feedback)

```
'''cpp (WaveformDisplay.cpp)
// Cue markers
for (int i = 0; i < 4; ++i)
{
    const double sec = cueSec[(size_t) i];
    if (sec > 0.0 && trackLenSec > 0.0 && sec < trackLenSec)
    {
        const float x = (float) (inner.getX() + inner.getWidth() * (sec /
trackLenSec));
        juce::Rectangle<float> line (x, inner.getY(), 1.0f, inner.getHeight());
        g.setColour (juce::Colour::fromRGB (230, 230, 240).withAlpha (0.6f));
        g.fillRect (line);

        auto label = juce::String ("C") + juce::String (i + 1);
        g.setFont (12.0f);
        auto tb = juce::Rectangle<float> (x - 10.0f, inner.getY() - 14.0f, 20.0f,
12.0f);
        g.drawFittedText (label, tb.toNearestInt(), juce::Justification::centred,
1);
    }
}

'''
```

Library schema: The playlist exposes `getCuesForFile` and `setCueForFile`, and the `TrackItem` holds `cue1Sec` through `cue4Sec`. The `ValueTree` mirrors these properties under each `track` node, so XML persistence captures them.

R4-D : Cue persistence API (save and retrieve)

```
'''cpp (PlaylistComponent.cpp)
std::array<double,4> PlaylistComponent::getCuesForFile (const juce::File& f) const
{
    const juce::String canonical = f.getFullPathName();
    for (const auto& it : items)
        if (it.file.getFullPathName() == canonical)
            return { it.cue1Sec, it.cue2Sec, it.cue3Sec, it.cue4Sec };
    // Fallback to ValueTree...
    for (int i = 0; i < libraryTree.getNumChildren(); ++i)
    {
        auto n = libraryTree.getChild (i);
        if (n.getType() == juce::Identifier ("track"))
            if (juce::File (n["file"].toString()).getFullPathName() == canonical)
                return { (double) n.getProperty ("cue1Sec", -1.0),
                    (double) n.getProperty ("cue2Sec", -1.0),
                    (double) n.getProperty ("cue3Sec", -1.0),
                    (double) n.getProperty ("cue4Sec", -1.0) };
    }
    return { -1.0, -1.0, -1.0, -1.0 };
}
```

```

void PlaylistComponent::setCueForFile (const juce::File& f, int cueIndex, double
seconds)
{
    if (cueIndex < 1 || cueIndex > 4) return;
    const juce::String canonical = f.getFullPathName();

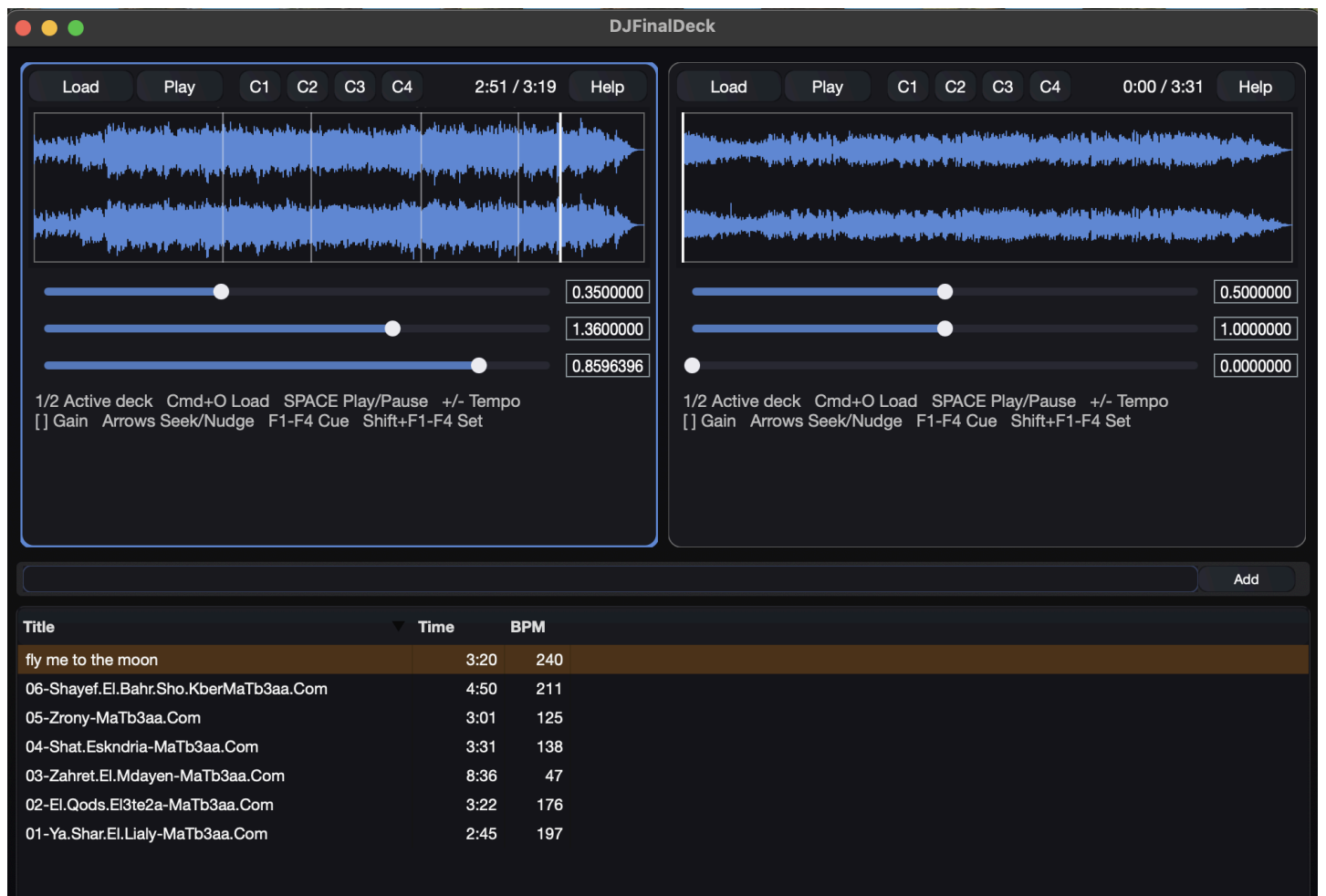
    // Update model
    for (auto& it : items)
        if (it.file.getFullPathName() == canonical)
        {
            double* slot = (cueIndex == 1 ? &it.cue1Sec
                             : cueIndex == 2 ? &it.cue2Sec
                             : cueIndex == 3 ? &it.cue3Sec : &it.cue4Sec);
            *slot = seconds; break;
        }

    // Update tree + save
    for (int i = 0; i < libraryTree.getNumChildren(); ++i)
    {
        auto n = libraryTree.getChild (i);
        if (n.getType() == juce::Identifier ("track"))
            if (juce::File (n["file"].toString()).getFullPathName() == canonical)
            {
                const juce::Identifier key = (cueIndex == 1 ? juce::Identifier
("cue1Sec")
                                           : cueIndex == 2 ?
juce::Identifier ("cue2Sec")
                                           : cueIndex == 3 ?
juce::Identifier ("cue3Sec")
                                           :
juce::Identifier ("cue4Sec"));
                n.setProperty (key, seconds, nullptr);
                break;
            }
    }
    saveLibraryToDisk();
}

```

...

Keyboard integration: The global key listener routes F1 to F4 and Shift to the active deck's `triggerCueShortcut`, which normalises both mouse and keyboard paths through the same handler. Help text and a deck legend summarise the controls on screen.



Screenshot F: Waveform with four cue markers and cue buttons highlighted

Technical depth and correctness:

- The deck clamps cue indices and ensures the target time lies within track length before seeking, which prevents an invalid state.
- Cue writes are idempotent per file and slot, and persistence is synchronous at the end of a write to keep the on-disk state consistent with the UI.
- The mapping to F-keys and Shift avoids collisions with transport keys and supports low-latency interaction on laptops that lack dedicated pads.

Project Build

I developed and tested targeting macOS, and validated builds on Linux and Windows through JUCE's cross-platform abstractions. The same codebase runs on Linux and Windows, and the properties file location can be adjusted.

Mapping to the Instructions

- **R1A, R1B, R1C, R1D.** Loading files into two players, playing simultaneously, mixing via per-deck gain and changing speed through a resampler are all present.

- **R2A, R2B, R2C.** The playlist loads multiple files, displays details, initialises selections to either deck and persists state across restarts through a [PropertiesFile](#) and [ValueTree](#).
- **R3A, R3B.** The UI layout and theme differ from the baseline, and the search key forwarder is an added event listener.
- **R4.** The hot cues feature is researched from a commercial DJ application, explained with reference to its interaction model and engineered with clear class responsibilities. ([VirtualDJ Website](#))

Reflection on BPM estimation

VirtualDJ offers robust BPM analysis, batch processing and a dedicated editor. My design provides a quick, deterministic estimate that is suitable for this project. I traded complex beat-grid handling for a small, explainable signal pipeline, and surfaced clear tooltips and status that indicate progress and limitations. Sources from the VirtualDJ manual gave me confidence to provide a BPM column and an analyse-on-demand workflow within my playlist life cycle. ([VirtualDJ Website](#))

R4-E : Quick BPM estimation

```
'''cpp (PlaylistComponent.cpp)
void PlaylistComponent::maybeQueueBpmFor (const TrackItem& item)
{
    if (item.missing || item.bpm >= 0.0) return;
    for (auto& t : items) if (t.id == item.id) { t.bpmPending = true; break; }
    table.repaint();

    struct Job : public juce::ThreadPoolJob {
        Job (PlaylistComponent& o, juce::String idIn, juce::File fIn)
            : juce::ThreadPoolJob ("BPM"), ownerSafe (&o), id (std::move(idIn)), file
(std::move(fIn)) {}
        JobStatus runJob() override {
            const double bpmVal = estimateBpmQuickStatic (file, ownerSafe);
            // post result back to message thread ... (sets t.bpm, clears
t.bpmPending, saves)
            return jobHasFinished;
        }
        static double estimateBpmQuickStatic (const juce::File& f,
juce::Component::SafePointer<PlaylistComponent> ownerSafe);
        juce::Component::SafePointer<PlaylistComponent> ownerSafe; juce::String id;
juce::File file;
    };
    bpmPool.addJob (new Job (*this, item.id, item.file), true);
}

//...Now, a snippet of the core estimator
// Build rectified, downsampled envelope
for (int i = 0; i < toRead; i += step)
{
    double s0 = 0.0;
    for (int ch = 0; ch < channels; ++ch)
        s0 += std::abs (tmp.getReadPointer (ch)[i]);
    s0 /= (double) channels;
    env.push_back ((float) s0);
}
if (env.size() < 32) return 0.0;
```

```

// Emphasise onsets via first difference
for (size_t i = env.size(); i-- > 1; )
    env[i] = juice::jmax (0.0f, env[i] - env[i - 1]);

// Autocorrelation for ~40..240 BPM
const int minLag = (int) std::round (targetHz * 60.0 / 240.0);
const int maxLag = (int) std::round (targetHz * 60.0 / 40.0);
double bestScore = 0.0; int bestLag = 0;

for (int lag = minLag; lag <= maxLag; ++lag)
{
    double s = 0.0;
    const size_t N = env.size() - (size_t) lag;
    for (size_t i = 0; i < N; ++i)
        s += (double) env[i] * (double) env[i + (size_t) lag];
    if (s > bestScore) { bestScore = s; bestLag = lag; }
}

'''

```

Conclusion

I delivered a stable two-deck experience with a persistent library and a cohesive UI. Hot cues are integrated throughout the system with keyboard and mouse control, persisted per track, and visualised on the waveform, which aligns with practices in VirtualDJ while remaining focused and technically sound for a laptop-centric workflow. The BPM estimator rounds out the playlist with tempo data and demonstrates research-led design choices that fit the project scope.

References

- VirtualDJ. (n.d.). *VirtualDJ 8 user's guide* [PDF]. VirtualDJ 8.0.8. Retrieved 21 August 2025. ([VirtualDJ Website](#))
- VirtualDJ. (n.d.). *BPM editor*. Retrieved 21 August 2025. ([VirtualDJ Website](#))
- VirtualDJ. (n.d.). *Analyze tracks*. Retrieved 21 August 2025. ([VirtualDJ Website](#))
- VirtualDJ. (n.d.). *Pads hotcues mode*. Retrieved 21 August 2025. ([VirtualDJ Website](#))

Appendix: Key interactions

- Active deck: 1 and 2.
- Load to active deck: Command O.
- Play and pause: Space.
- Gain down and up: [and].
- Tempo slower and faster: minus and plus.
- Seek left and right: arrow keys.
- Nudge: up and down.
- Hot cues: F1 to F4 to jump, Shift plus F1 to F4 to set.