“Tunepal Nua” ABC Importer: Indexing Corpus of Traditional Irish Music on an SQLite Database in Godot

Table of Contents

Table of Figures

Table of Tables

Glossary of Terms

ABC Importer: A subsystem developed for Tunepal Nua that converts ABC notation files into database entries, making them searchable by audio transcription

ABC notation: Plaintext musical notation used as a standard for traditional and folk music

ABC parser: A component which proceses raw ABC files, extracting tune information and filtering extraneous text or characters

API (Application Programming Interface) a set of protocols allowing software applications to communicate with one another

Fast Fourier Transform (FFT): mathematical algorithm used in pitch detection

Finite Impulse Response (FIR): A digital filter used in note onset detection

GDExtension: a Godot-specific technology that lets the engine interact with native shared libraries at runtime.

GDScript: Built-in scripting language in the Godot engine

JSON (JavaScript Object Notation): a lightweight text-based data-interchange format

MATT2: original transcription algorithm for Tunepal developed by Dr. Bryan Duggan

MIDI (Musical Instrument Digital Interface): A technical standard used in digital representation of music

ML (Machine Learning): A branch of artificial intelligence that enables systems to learn from data.

Node: Fundamental building blocks used in Godot Engine

ONNXRuntime: Inference and Training accelerator used in running machine learning models

Parsons code: Notation system that represents the melodic contour of a piece of music

PWA (Progressive Web Application): A web application that can be installed on a device and behave like a native application

QBP (Query-by-playing): search method in which a recorded snippet of music played on an instrument is used to search and retrieve a piece of music

SCons: Software construction tool used for C++ compilation

Scrum: An agile methodology used for managing complex projects

UI (User Interface): The visual elements and interactive components through which users interact with the application

Abstract

Tunepal is a popular query-by-playing search engine for Irish Traditional Music. This project involved onboarding to a development team, developing a next-generation Tunepal: *Tunepal Nua*. This work herein addresses some of new the new features which will set *Tunepal Nua* apart from its predecessor.

This work details the successful development of a system for indexing traditional tune collections from a digital plaintext form of music notation called “ABC notation” to an SQLite database in the Godot Game Engine. The system developed in this work transforms the tune’s notation into midi sequence and search keys that can be matched to notes which are transcribed from audio. It also scrapes tune metadata and inserts these data to a database along with the search keys. This work also illustrates work carried out on professionalising the workflow within the development team using Agile methodologies. Some early-stage investigatory work on an alternative Machine Learning (ML)-enabled transcription engine is documented.

1. Introduction

Tunepal is an application designed for musicians, learners, enthusiasts, scholars and teachers of Irish traditional music. It is a searching tool which instantly connects users with vast collections of traditional music from Ireland, as well as other related folk repertoires including Scottish, Welsh, Breton and North American amongst others.

It can be described in short as the "Shazam" of Irish music: its most defining feature is it's "query-by-playing" (QBP) search of vast collections of traditional music which enables a user to record a snippet of audio, from which it returns a list of matches with a confidence score. It also features a keyword search which combines various online collections and archives of traditional tunes: essentially an all-in-one traditional music search engine. Tunepal also offers other useful features such as displaying the musical score, midi playback, random tune and other useful features.

The application was originally created in 2009 by Dr. Bryan Duggan of Technological University Dublin (TUD), as a project which stemmed from the body of work undertaken for his PhD on MATT2. This is the original Java-based QBP system at the core of Tunepal (Duggan, 2009). It has since gained widespread use and a reputation for being an entirely novel, and revolutionary tool for Irish musicians. Until recently, it was the only such application of its kind and is still much-loved in the Irish music community.

For a traditional musician, or a learner of traditional music, this is an incredibly powerful and useful tool, as it enables anyone to accurately identify the name or even particular version of a traditional tune (of which there are many thousands in existence) from a snippet of audio.

The creator of the application has begun building a new version of the Tunepal: "Tunepal Nua" with a modern cutting-edge design, intended to be free, open source and with scope for further development. The work outlined in this report documents contributions made to the subsystems of this new Tunepal. The primary focus of the work is the "ABC Importer": a program which enables building of a database containing the corpus of traditional music from scratch on an SQLite database on device, and its development towards becoming an important resource for the open-source community.

2. Research

The initial project proposal for this work was focused on developing a new user interface (UI) for the new version of Tunepal using the original Tunepal API. Progressive Web Application (PWA) technologies such as Flutter, and React/Next.js were at first considered for this project (Suárez, 2024) (nextjs.org, 2024).

All the above are effective technologies for modern cross-platform application development. However, as the creators have chosen the game engine Godot as the technology of choice for building this new version. Godot was designed for building 2D

and 3D games, however it is also highly capable of building fast cross-platform applications and user interfaces (Alessandrelli, 2021) (Oesterle, 2024).

Godot is a less conventional method for building cross-platform web applications and could be considered overkill for many use cases for which conventional technologies such as JavaScript frameworks are adequate. However, it presents a cutting-edge solution and arguably better alternative for some use cases in UI development (Alessandrelli, 2021). Such a use case is Tunepal.

Following research into using conventional Javascript frameworks and PWA technologies for this work, it was determined that it would entail a significant challenge in interfacing a PWA technology (*i.e* Flutter) with Godot. Incorporating such a PWA technology with Godot would be excessively complex and would not align with the goals of the Tunepal project.

The Godot engine is extremely versatile. Besides being originally intended for game development, it also has the capability to host applications compiled for and run in web browsers, mobile devices and desktop. Godot also presents a promising means of building PWAs (Alessandrelli, 2021). It is also very easy to build and export Godot projects to run natively on any device or operating system (Godot Documentation, 2024).

The creators had carried out prior work on this and had ported much of the basic UI layout and functionality of the original application to Godot/C++ in a private repository. However the core functionality was only partly implemented. These were determined as the goals for the project and will be discussed in a later section. This repository was subsequently made public, enabling others to contribute to development of *Tunepal Nua*. This marked the kick off point for the contributions towards *Tunepal Nua* carried out as a part of this project. This was also the first step towards achieving the creator's goal of making the new Tunepal into a community-driven, open-source project. The benefits of being an open-source project are clear: any new feature development is subject to scrutiny within the community affording greater robustness in design. In addition, bug-fixes and feature development are enhanced and accelerated by widespread collaboration. Other benefits of being open source include enhanced security, performance and flexibility of use (Lorraine Morgan, 2007).

2.1. Godot Engine

Godot is a free, open-source game engine; however, it is also a highly capable and powerful UI engine. One popular example of a UI created in Godot is the "Bosca Ceoil" music sequencer (Sizov, 2024). Other examples include the Godot editor itself, "Material Maker", "Pixelorama", and "CozyBlanket" (GameFromScratch.com, 2024).

Godot has its own scripting language called GDScript, but also natively supports C#. It also supports C++ through the GDExtension API. A GDExtension is a binding for various programming languages in Godot. GDExtensions exists for a wide range of programming

languages including Python, Go, Rust, and TypeScript. Since Godot itself is written in C++, the C++ GDExtension binding found at godot-cpp is one of the best choices for writing custom nodes and functionality (Snopek, 2024).

The Godot engine uses “scenes” which are a collection or tree of elements (nodes and other scenes) which constitute a functional unit of a game or application (Godot Documentation, 2024). Nodes serve various functions: displaying images, handling logic or handling input (DragonflyDB, 2024). Nodes together form a tree, making it easy to organise projects. It is possible to run a “scene” to test out the app in the Godot editor.

Godot has some drawbacks too for creating UI and web applications: one major flaw for this is the lack of screen reader support (El-Kouly, 2022), as well as not supporting deep system integration (Oesterle, 2024).

Advantages of using Godot for UIs are that it is open source, it can be exported to any major platform, and it is easy to create visual effects that are extremely advanced compared to what is possible with traditional UI engines. Prototyping is extremely fast and easy in Godot. Working UIs can be assembled visually in the Godot editor can also be used instead of other tools such as Figma for prototyping (Oesterle 2024).

Using a game engine for a standard software application also has some drawbacks and advantages. Drawing a new frame at the specified frame rate is a waste of CPU resources. However, Godot can be run in low processor mode which reduces the demand on CPU. While not particularly suitable for many Games, it is suitable for most standard UI applications. All the UI features such as tabs, buttons, panels etc… are typically built using “control nodes”.

Godot has its own midi player and audio stream recorder which are very easily implemented, making hardware interfacing simple. This offers a huge potential benefit to Tunepal: namely is can record an audio clip that is subsequently transcribed. This requires significantly less work in Godot than it would if using more conventional JavaScript frameworks or other equivalent tools. Godot has plugins intended for game development that are easy to implement. For instance, Tunepal V2 uses Godot’s AudioStreamPlayer and AudioEffectRecord plugins, which enable utilisation of a device’s audio bus.

In the context of the Tunepal app, each separate menu such as keyword search, record tune, random tune etc… represents its own scene. The main app itself is the main scene.

3. Analysis

In recent years, an alternative tune identification application has appeared on the market. It is a free web-based application, very similar to Tunepal, named “Folkfriend” (Wyllie, 2025; Wyllie, 2020). Though less widely known and with fewer features, it is a noteworthy competitor to Tunepal as it also has a query-by-playing search feature. It can

easily be installed on a mobile device from the browser and does a reasonably effective job of matching tunes, though it's database may differ.

Tunepal's Android and iOS applications have become increasingly dated over the years since their initial release. The recent web release is the best performing available version currently available (found at https://tunepal.org/index.html#!/record). It is largely pared back for simplicity compared to what was initially developed: Other previous additions to Tunepal such as viewing geo-locations of tunes (by using the geographic coordinates as metadata (Duggan, 2010)) have largely been left out of the more up-to-date browser-based version for this reason.

Tunepal has had seen some minor bug fixes and updates throughout the years, but by enlarge has remained the same. On the other hand, technology and the online world have advanced, grown and transformed enormously in the decade since Tunepal's initial release. With cutting-edge technologies such as AI driving modern applications presenting huge potential for use within an Irish music search tool. Without taking these new advances into consideration, Tunepal runs the risk of being left behind and another more modern alternative taking its place. Some drawbacks of Tunepal are still evident from the beginning and have been sought after by its users for a long time: its inability to work offline being one notable example which this work addresses.

The existing publicly available Tunepal is closed source, and it is in several different codebases: for example the web version is in Java and JavaScript, with the MySQL database at the backend hosted at tunepal.org. The android and iOS versions each have their own codebases.

3.1. Use-Case Scenario

Tunepal users are typically traditional musicians and enthusiasts of Irish Traditional music. There are many possible scenarios for the typical end user of Tunepal. To keep this concise, one such use-case scenario will be described below. This hypothetical end user, Gráinne, is an individual who is attending workshops at an Irish music festival:

Gráinne is attending fiddle workshops at the *Scoil Samhraidh Willie Clancy* in Milltown Malbay. Irish traditional music is typically learned "by ear": *i.e.* tunes are picked up from listening to other musicians playing. As a result, many teachers will still teach tunes by ear and without musical notation. Gráinne's teacher is fiddle player Sorcha, who will teach two to three tunes per day, and write some notation on a chalkboard. However, Gráinne chooses to record Sorcha's playing to capture the subtle variations, ornamentation and dynamics. Gráinne returns home with a long list of audio recordings on her phone which were not transcribed and are without titles, both from classes and from sessions. In some cases, teacher Sorcha may not have remembered the names of tunes anyway or may have more than one title for a tune. She feels she has learned so many tunes that she starts to mix them up. Over the coming months, Gráinne learns

some of the tunes from her recordings, but does not know the titles. She uses Tunepal to quickly ID the tune after she learns it, which provides her the titles and the musical score to go with it. This helps her learn the tune later, ensuring she doesn't mix up or lose the tune she has recorded. Gráinne has always loved the simple user interface and the immediately accessible "record" button. However, Gráinne lately has noticed that her Tunepal app for Android doesn't function well on the latest version of the Android operating system which she uses on her phone. She has saved the web version to her phone's home screen.

One feature Gráinne finds useful in Tunepal is the display of archive recordings on each tune from Europeana sounds, which retrieves recordings of tunes that have been identified through an API call to the Europeana API. Europeana Sounds is a project curated by the British Library and aggregates collections of music related content such as audio, manuscripts etc (Europeana Sounds, 2017). This API call adds the ability for a user to immediately hear the tune as it was played by musicians of previous generations, capturing the nuances of the music that would be missed through learning from a score.

A user, such as Gráinne, may have encountered new sources of tunes in collections (often referred to as "tunebooks"). Examples of well-known tunebooks are *O'Neill's Music of Ireland* or *Martin Mulvihill's First Collection of Irish Music*. Musicians or enthusiasts often compile collections or compose new tunes. Therefore, having a fixed database of tunes is not desirable for many users. Tunebooks containing specific versions of well-known session tunes or newer compositions may not be present on the current database. For such cases, many users wish to have the ability to expand the scope of tunes that Tunepal can identify. The solution to this is a feature that allows someone to digital music collections onto the database.

4. Modelling

This section outlines the conceptual design of Tunepal and of the pre-existing codebase: identifying the existing components and to map out the architecture to understand how and where any new feature development or fixes would be integrated into the existing project.

4.1. Conceptual Design

The initial aim of "Tunepal Nua" will firstly be to achieve the same basic functionality as the original version of Tunepal, and the UI of this first release is aimed to mirror the original – except built in Godot.

Tunepal's UI has a simple layout and intuitive design. When a user opens the app, the first page they see is the "Record" page (Figure 1 (a)). A large "Record" button allows the user to start audio recording which will capture audio for 12 seconds with a short countdown before recording begins (Figure 1 (b)). After recording, a list of tune matches is returned,

and the closest match is shown at the top of the page with the percentage confidence (Figure 1 (c)). One left hand side is a simple sidebar menu with the list of available pages.

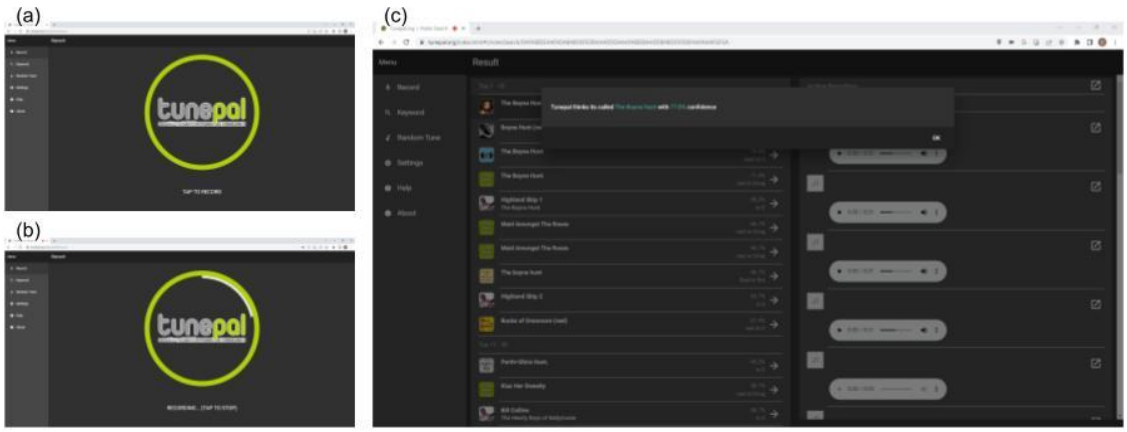

Figure 1 (a) Record page (landing page) (b) during recording (c) showing search results

The second menu item in the menu sidebar is the keyword search. When selected, this presents the user with a simple search bar where the search by keyword is carried out (Figure 2 (a)). The keyword search returns all the matches as well as a side panel of audio recordings for this tune retrieved from the Europeana Sounds API (Figure 2 (b)). The Europeana sounds integration makes a simple API call to Europeana sounds using the tune's title.

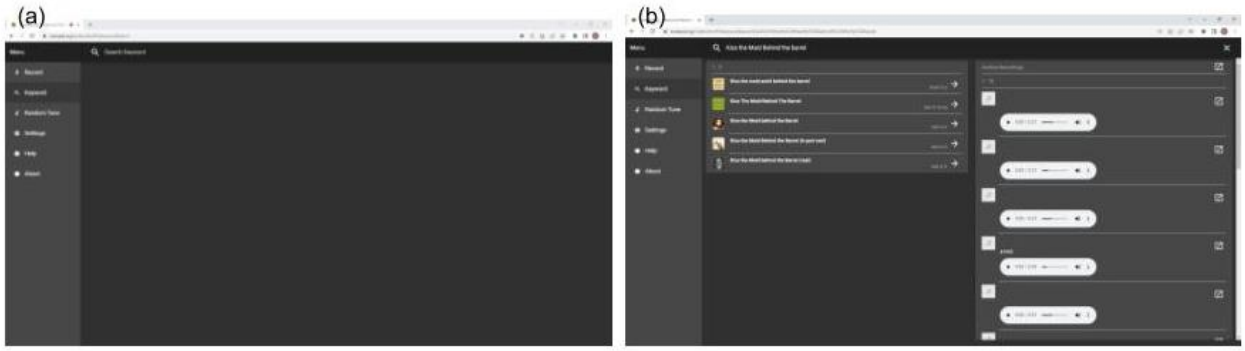

Figure 2 (a) keyword page (b) with returned search result

If a tune is selected, the user is taken to a page which displays a score (generated from ABC by abcm2ps) and is shown a list of audio recordings retrieved from the Europeana Sounds API. This is shown in Figure 3. The exact same view is rendered when a user selects "Random Tune" from menu sidebar, however with a tune picked randomly from the database.

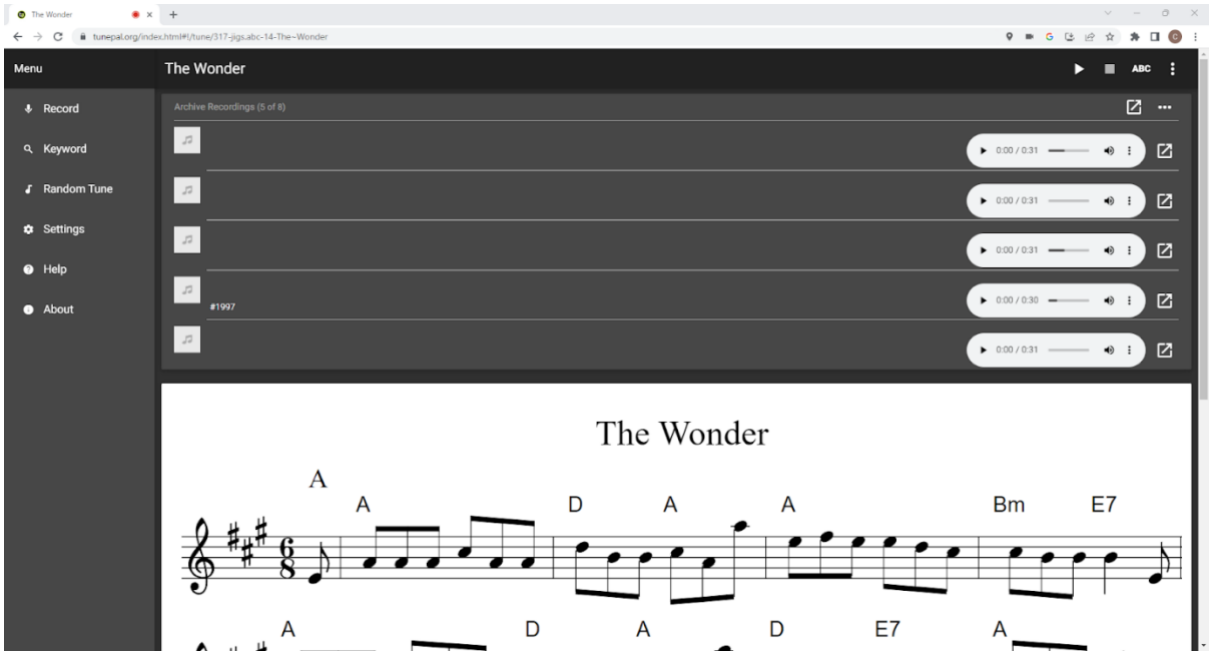

Figure 3 Random tune page

In Figure 4 is shown the "Settings" page. This allows user to change the countdown timer, control sample rate, transcriber frame size, fundamental note. Fundamental note is used for reed or woodwind instruments with different fundamental key, for instance a B-flat set of Uilleann Pipes). Users can also filter searches by source (tunebooks) or by time signature by configuring the settings here.

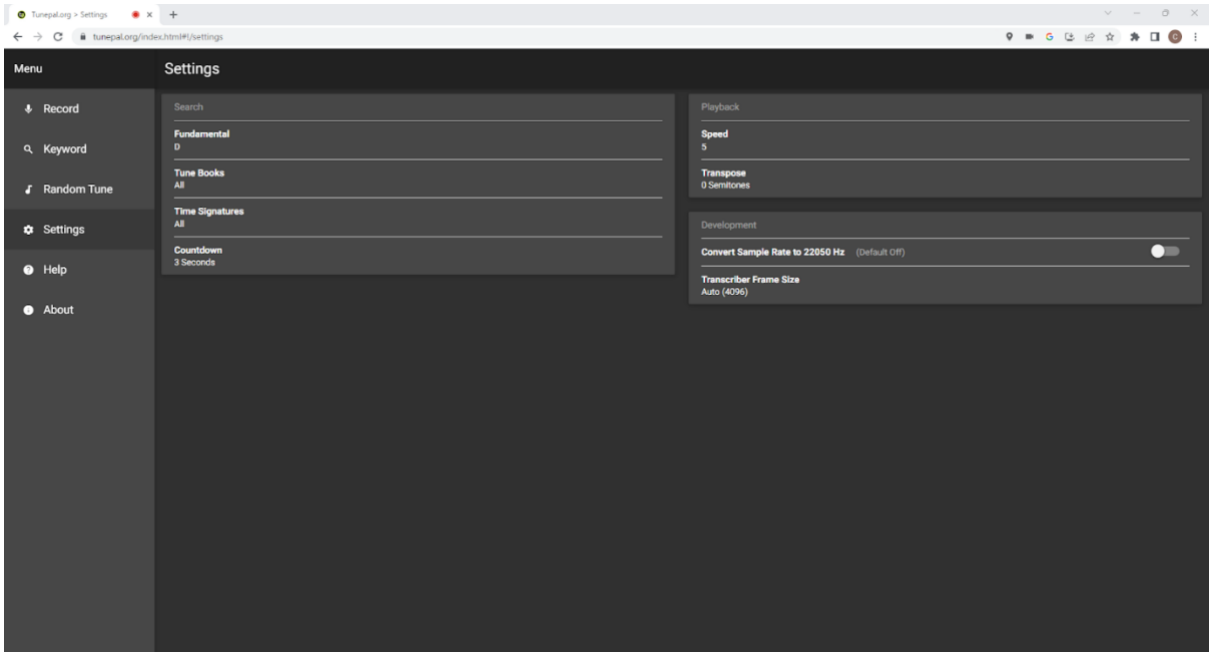

Figure 4 Settings page

Figure 5 shows a "Help" page which displays help information: with simple usage instructions.

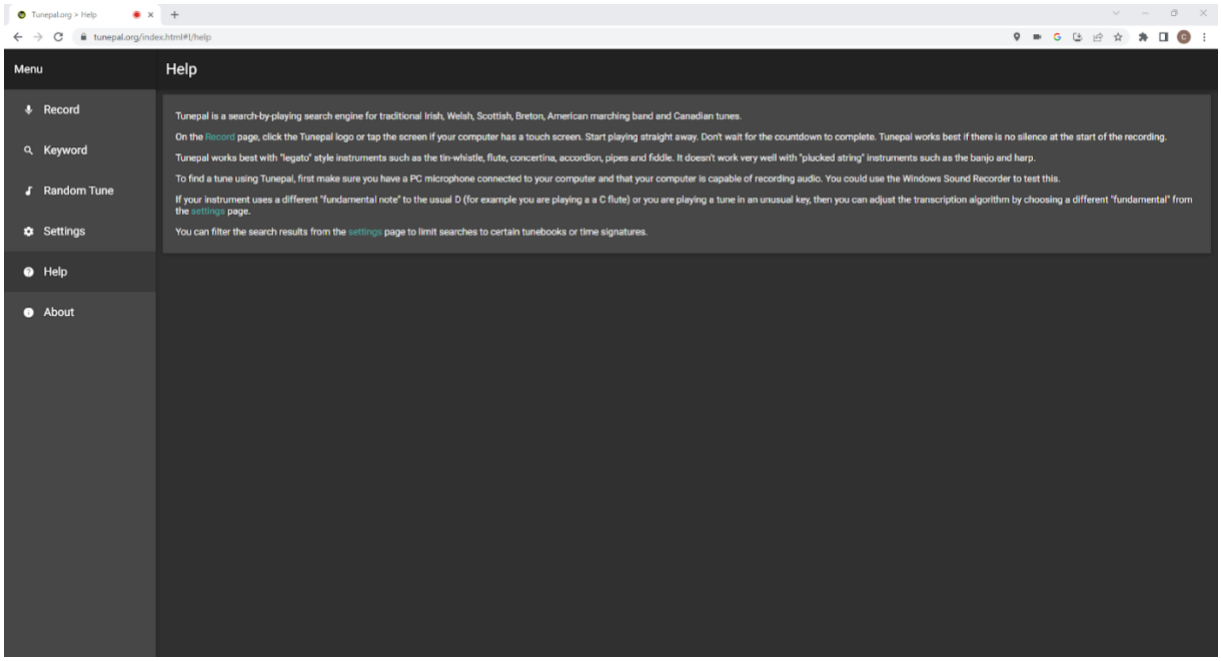

Figure 5 Help page

The about page in *Figure 6* shows background information, information on contributors, and the list of sources.

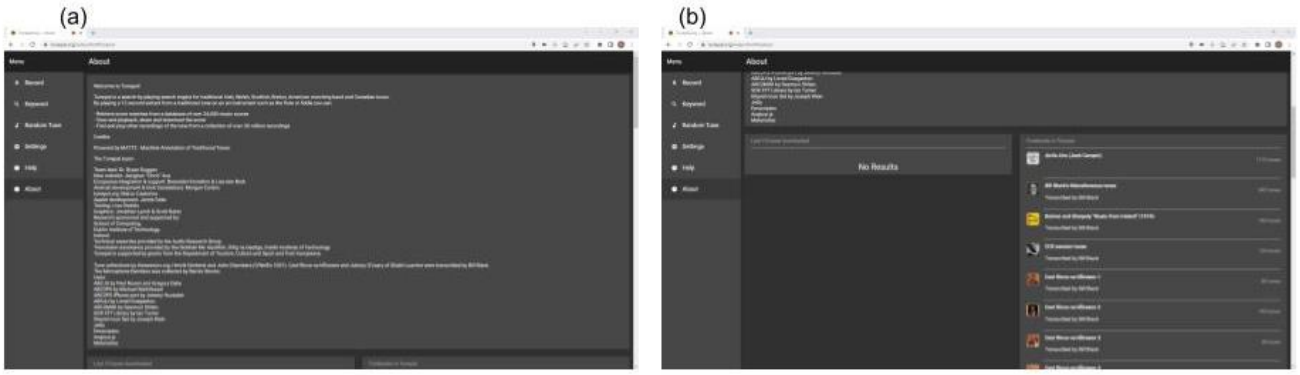

Figure 6 About page (a) top of page (b) bottom of page showing tunebook (sources)

4.2. ABC Music Notation

"ABC Notation" is a plaintext-based form of music notation which is widely used, particularly in folk and traditional music. It was introduced by musician and computer scientist Chris Walshaw in 1993 (Chambers, 2002). It enables music notation to be easily read and processed by computer programs: such as abc2ps (which renders ABC notation as a postscript version of standard music notation) or abc2midi (which converts ABC notation to a midi file file). ABC notation has seen a growing, vibrant community supporting this format. For instance, thesession.org hosts all its tunes for free in ABC format. It is the *de facto* standard for most other contemporary traditional and folk online collections. For example, notable collectors such as Henrik Norbeck and Bill Black use ABC notation for storing their vast digital collections of music (Norbeck, 2021; Black,

2024). The principal online source of tunes for traditional musicians “thesession.org” hosts all of its tunes in ABC. ABC notation simplifies music notation compared with conventional sheet music notation, by using only upper- and lowercase letters (a-g, A-G). Other key information such as time signatures, key signatures, tempo, are stored as metadata in headers. An example of traditional tune “The Star of Munster” in ABC format is shown in Figure 7:

```
X: 1
T: Star Of Munster, The
Z: Jeremy
S: https://thesession.org/tunes/197#setting197
R: reel
M: 4/4
L: 1/8
K: Ador
|:ed|c2Ac B2GB|AGEF GEDG|EAAB cBcd|eaaf gfed|
cBAc BAGB|AGEF GEDG|EAAB cded|cABG A2:|
|:de|eaab ageg|agbg agef|gfga gfef|gfaf gfdf|
eaab ageg|agbg agef|g2ge a2ga|bgaf ge:|
```

Figure 7 example of a tune transcribed in ABC Notation

Most important header fields are X (index number), T (title), S (source), R (rhythm), M (meter / time signature), L (length of base note), and K (key signature). The index number (X) always comes first, and the key signature (K) always comes last. There are a range of other headers which convey other useful metadata: such as Composer (C), Transcriber (Z), discography (D) amongst many others though these are optional as they do not convey musical information or direction (Chambers, 2002). The notation itself comes after the headers and primarily features characters (A-G, a-g). Other characters which denote bar divisions (|) repeats (|: , :|) or ornamentation such as triplets, trills, rolls etc.

4.3. MATT2 Transcription Algorithm

The “core” engine of the original Tunepal, like the original version first used in MATT2 (Duggan, 2009), takes digitised audio input and matches this with tunes in a corpus of Irish traditional music made up of various tune collections. It differs from acoustic “fingerprinting” used in Shazam as it does not directly compare digitised audio signal spectrograms, but rather compares a signal transformed to a simplified version of a tune’s notation made up from a string of characters representing the notes (Wang, 2003). This involves two core subsystems (Figure 8)**: (1)** audio transcription and **(2)** tune pattern matching. These subsytems were converted from their original Java implementation to

C++ by Tunepal's creator in the beginning stages of *Tunepal Nua*'s development, to allow the core MATT2 transcription algorithm to be compiled and utilised with the GDExtension binding such that it can be called in Godot scripts.

4.3.1. Transcription

Transcription is further subdivided into subsystems for onset detection, pitch detection and pitch spelling. Note onset detection is enabled through using Finite Impulse Response (FIR) comb filters. Pitch detection is achieved using Fast Fourier Transform (FFT) of the audio signal. Ornamentation filtering is achieved through creation of histograms of note durations and subsequent removal of note ornamentation from longer notes. Pitch spelling converts an audio frequency to an alphabetic character representing the note played on a musical scale. There is also breath detection: which is required for woodwind instruments such as concert flute, in which gaps (due to player taking a breath) occur during the tune.

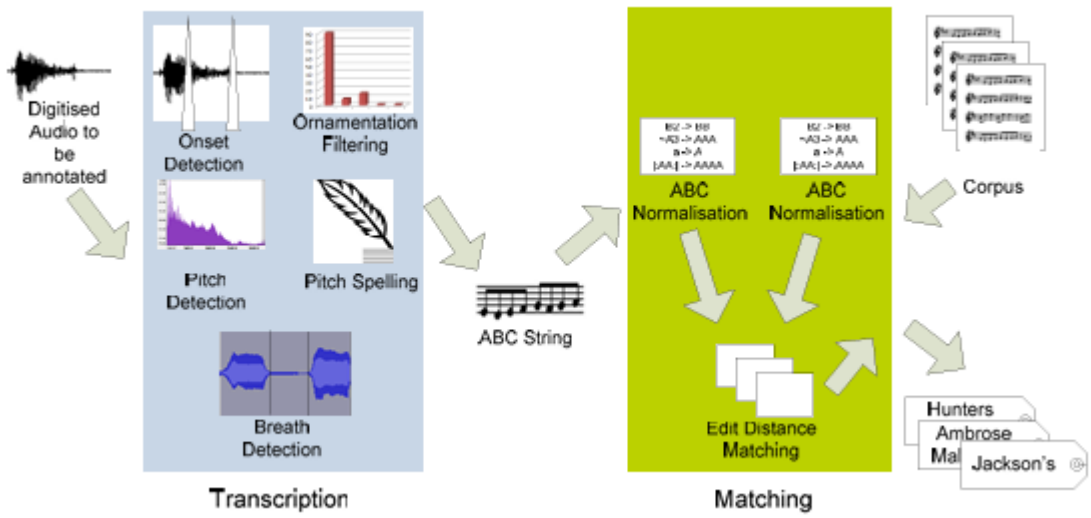

Figure 8 High level diagram of original MATT2 system developed by creator Bryan Duggan (Duggan, 2009).

The output of the transcription algorithm is a string (herein referred to as an "ABC string") representing the musical notation present in the tune's recording.

4.3.2. Pattern Matching

The second major component of Tunepal's transcription engine is the pattern matching algorithm (as shown in Figure 8). This is carried out through "Edit distance" matching against the corpus of traditional tunes. "ABC Normalisation" is required to be carried out on the tune corpus and on the transcribed musical notation. For the corpus of traditional tunes, this involves stripping out characters which represent ornamentation, bars, and long notes, and semitones: returning a string that solely comprised of uppercase characters A to G. An example of this string normalisation for the "A" part of the traditional tune "Come West Along the Road" is shown in Figure 9.

Original:
d2BG dGBG|~G2Bd efge|d2BG dGBG|1 ABcd edBc:|2 ABcd edBd||

After ornamentation filtering:
d2BGdGBG|G2Bdefge|d2BGdGBG|1ABcdedBc:|2ABcdedBd||

After note expansion:
ddBGdGBG|GGBdefge|ddBGdGBG|1ABcdedBc:|2ABcdedBd||

After section expansion:
ddBGdGBGGGBdefgeddBGdGBGABcdedBc
ddBGdGBGGGBdefgeddBGdGBGABcdedBd

After register normalisation:
DDBGDGBGGGBDEFGEDDBGDGBGABCDEDBC
DDBGDGBGGGBDEFGEDDBGDGBGABCDEDBD

Figure 9 Normalisation stages for the A part of the tune "Come West Along the Road" (Duggan, 2009).

A subsequent transformation can be carried out by the matching algorithm if corpus strings are shorter than the transcribed string: they are duplicated until their length exceeds the transcribed string – but in a manner which reflects how a traditional player would naturally duplicate the length of a tune.

The edit distance matching can be simply described as an algorithm which counts the number of required substitutions, deletions, insertions *etc*., to create a perfect match between the transcribed ABC string and the corpus ABC string. The algorithm returns the top ten matches of ascending distance from the transcribed ABC string (Duggan, 2009).

Re-creating the string normalisation algorithm was one key focus of the work carried out in this report.

One of the greatest challenges in creating a database builder for Tunepal is in creating a search key suitable for use by Tunepal's query-by-playing search engine. As described in section 4.2 and illustrated in Figure 9, the ABC notation must undergo "Normalisation", in order for the edit-distance matching algorithm to function. *i.e.* the database indexer must carry out (1) ornamentation filtering (removing extraneous characters such as "~"), (2) note expansion (replacing long notes with repeated notes based on standard note length and time signature). (3) Section expansion: where "parts" of a tune with repeat markers are duplicated and (4) register normalisation: all notes changed to uppercase.

4.4. Modelling System Design and Architecture.

Given that work had been previously carried out on this project in a private GitHub repository, contributions for this project were added whilst this project was already at a relatively advanced stage. It was necessary to document and model the existing structure of the project codebase. At the point of commencement of contributing to the project,

the codebase was already large and complex. The diagram in Figure 10 shows a simplified model which highlights the key aspects of Tunepal, built using the Godot engine.

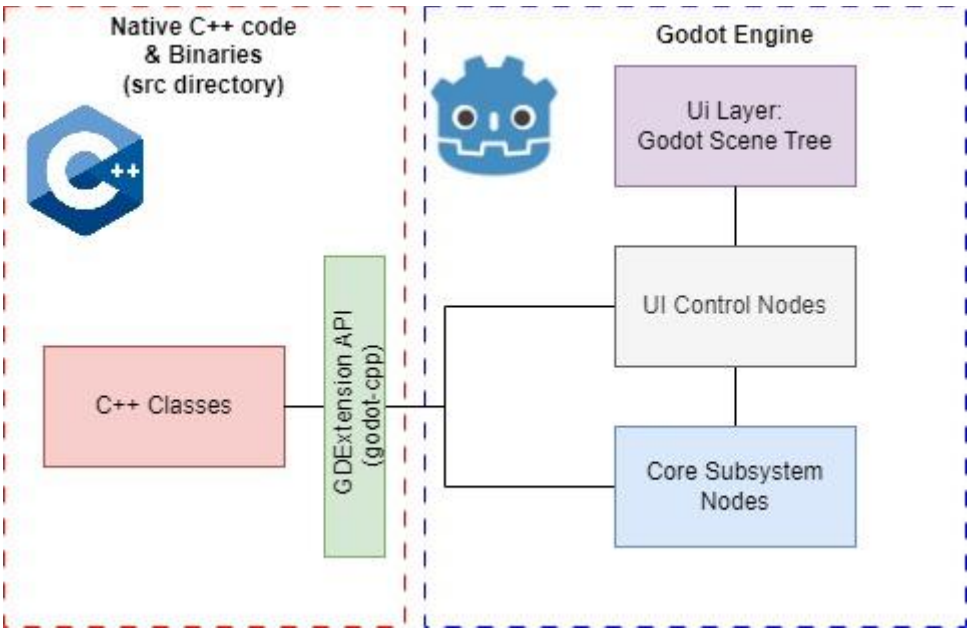

Figure 10 Simplified Model of Tunepal Nua Built using Godot Engine

On the right-hand-side of Figure 10 are the systems which comprise the Godot engine. These can be classified as the "Scene tree", UI Control Nodes and Core Susbsystem Nodes. The scenes dictate the UI layout and function and instantiate various nodes – which are organised in a tree-like structure. "Scenes" and Scripts can be used together or interchangeably (Godot Documentation, 2025). The scenes used in Tunepal include navigation, menus, buttons, scrollbars and containers. Closely related to these are "UI Control Nodes". These are the nodes (or classes) that act as "controllers" and are executed when a user interacts with the scene tree. These are generally user-defined scripts (written in GDScript). Others are pre-built nodes in Godot that can be linked to a scene. Finally, the core-subsystem nodes can be compared to the application's backend. Examples of these with regards to this project are the ABC importer and ABC parser scripts. The core subsystem nodes also consist of the utility nodes which are generally Godot-native classes that can interact with device hardware, such as AudioStreamPlayer and FIleAccess.

On the left-hand side of the diagram in Figure 10 are custom nodes, written in C++ which can be written for- and utilised by the Godot engine through the GDExtension API. These are all contained in the *src* directory of the codebase. These include the transcription engine. The C++ GDExtension (called godot-cpp) is a separate repository (Snopek, 2025), included as a submodule to the main repository. When compiled it allows the Godot engine to interact with other native C++ libraries at runtime (Godot Engine Documentation, 2025), allowing integration of third-party components.

Figure 11 shows a more detailed diagram illustrating the fundamental components of the application and their relationship to one another. The UI layer is governed within Godot engine and can be directly edited within the Godot editor, or a separate IDE (preferably The scene configurations are defined in .tscn files. The nodes are component-based *i.e.* they can be added or removed from other nodes as child nodes to add functionality, unlike in OOP which solely relies on object inheritance (which nodes are also capable of) (Godot Documentation, 2024). In Figure 11, the UI layer is at the top half of the diagram, and native C++ code (in the src directory) is on the lower half of the diagram.

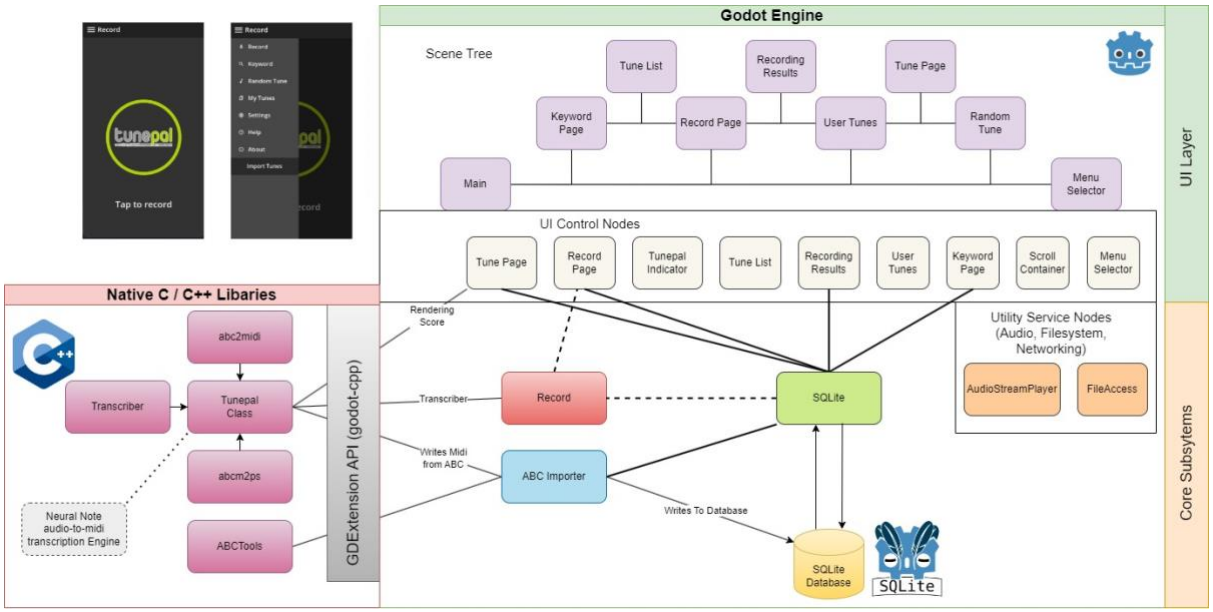

Figure 11 High level architecture diagram of "Tunepal Nua".

All the subsystems involved in transcription and matching (as described in section 4.2) are found in the *src* directory and are present as C++ code converted over from the original Java implementation. There are also various C and C++ libraries in the src directory which add functionality to the Tunepal node: such as **(1)** score rendering and Midi conversion: abcm2ps (Moine, 2016) and **(2)** abc2midi (lewdlime, 2016) respectively. These are compiled and integrated into the project coupled with the godot-cpp GDExtension API mentioned above.

The "Sconstruct" file in the root of the project is a build-configuration file for used by the SCons software construction tool (equivalent to other tools such as CMake). It a python script, in which the paths and flags passed to the compiler are outlined. SConstruct, used by "SCons" program is equivalent to a Makefile for CMake.

In addition, the core "utility nodes" shown in Figure 11 include for example the FileAccess and Clientside Nodes (for filesystem operations). Godot has a range of Nodes out-of-the-box which make interaction with device hardware extremely easy: for Tunepal this is extremely useful for audio recording and playing: through using instances of the AudioStreamPlayer and AudioServer Nodes. Another key system node used by Tunepal

is the SQLite node: provided by the Godot-SQLite plugin, which is a “GDNative” wrapper for sqlite binary, and handles database operations in the Godot engine to a database file. The primary focus of this work was on developing the ABC Importer subsystem: which involved utiilising the Godot-SQLite wrapper, and creating a range of new nodes and scripts in the Godot engine.

Further to the architectural model presented in Figures Figure 10Figure 11, other noteworthy components of this codebase are listed as follows:

Scenes

The “scenes” directory contains all of the “.tscn” files. These are the building blocks of Godot games. They contain all the data for a “scene” of a game, or in this case, typically a UI screens or layouts, and the hierarchy of nodes in each scene, in a text-based format.

Scripts

This directory contains all of the GDScript files used as core or UI control nodes.

Addons

The “Addons” directory contains the .gdextension files and binaries. This include pre-built plugins that are easy to install and integrate into the Godot project such as Godot SQLite and midi plugins. Custom GDExtension binaries and .gdextension files such as the “Tunepal” addon. The “.gdextension” files are essentially configuration files which define which binaries the Godot engine to look for: *i.e.* builds for different operating systems. In addition, some of the scripts initialising plugins that utilise device hardware such as midi playback and audio recording (AudioStreamPlayer and AudioStreamRecord respectively) are located in this directory and can be deployed as nodes in Godot engine.

Android

The “android” directory contains all the necessary files for android builds (such as gradle files and associated libraries) to compile .apk android build files.

Assets

The “assets” directory contains files which are utilised by the running application in the Godot engine. For instance: images, icons the local database file “tunepal.db”, other persistent data (“user-tunes.json”). The local device database is an important new feature added to the new version Tunepal, in which a database located on the device is used rather than making API calls to an external database server. Whereas in previous version, the app could only be used with an internet connection. The local database is present on the device, enabling offline use. This is a significant enhancement to Tunepal, as users frequently wish to use the app in areas with poor internet access.

4.4.1. Tunepal Database Structure

The SQLite database used for "Tunepal Nua" features a relatively simple schema with just three tables: *tuneindex*, *tunekeys* and *source*. The database schema is shown in Figure 12. The *tuneindex* table contains most of the tune information as represented in the original ABC source file: including essential header fields and notation (before normalisation). *Tunekeys* includes the "normalised" abc notation, (search key used in "query-by-playing"), midi sequence (numerical representation of notes) and "Parsons" code (text representation of the melodic contour of the tune using characters "S", "U" and "D").

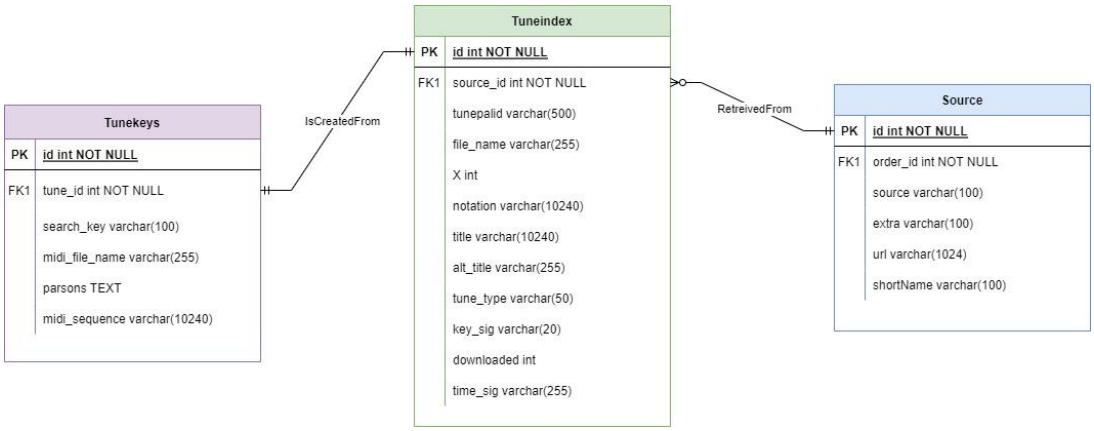

Figure 12 Tunepal Database Schema

The *source* table contains information on the sources of the tune – extracted from the ABC source file. This includes a source name, id, URL and abbreviated name. Each tune shares a 1:* relationship to their sources and a 1:1 relationship with its individual tune key. This could be expanded in future to a 1:* relationship between tunes and tune search keys to account for tune settings and avoid repetition however this implementation is still under active development.

4.4.2. ABC Importer system design

The ABC Importer is a core new subsystem of the Tunepal application. This new version affords the user control to index their own tunes and tunebooks, as well as providing a way for maintainers to update the database with a way of building a fully up-to-date database.

The local database will give a significant advantage over previous versions of Tunepal in that the database will now be available offline as well as online.

Figure 13 provides a high-level interpretation of the system design and function of the ABC importer. A user may download a collection of digitised tunebooks in ABC format, or transcribe tunes themselves. For example, these could be new compositions, or transcribed tune collections from archives. The ABC files, saved in any directory can then be opened in the Tunepal app. The Importer will read the contents of the directory. If it

recognises an ABC file, it will find which parts of the file are the headers, the notation and any other text or comments and processes them accordingly. All relevant information is added to the *tuneindex*, *tunekeys* and *source* tables in the database and inserted.

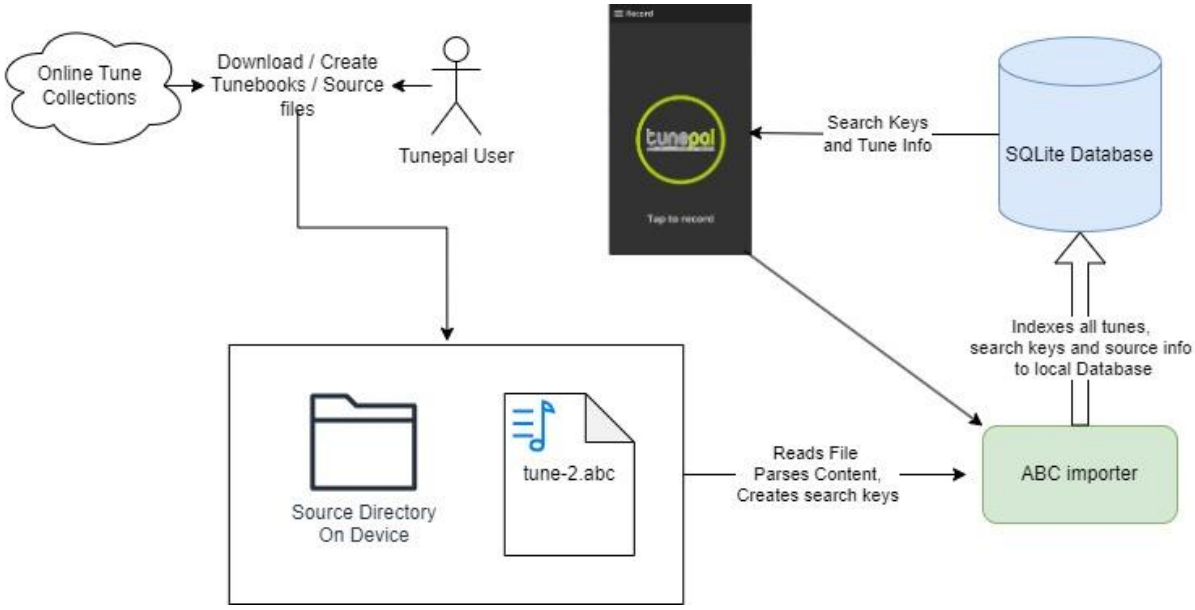

Figure 13 High level overview of ABC Importer System

A more detailed view of the ABC Importer System is given in Figure 14 highlighting some of the subcomponents involved. Central to the functioning of the application is the Godot-SQLite plugin, which is a Godot-native wrapper for SQLite and enables database transactions or queries to be run on Nodes in the Godot engine. The SQLite.gd script is the principal controller for SQLite in this application. It not only initialises the database but also contains the functions which are triggered by the UI button in the menu to run the app. Alongside this are the nodes ABCToolsClass, ABCParser, and ABCImporter. The ABCImporter node is central to the functioning of this system. It contains the functions which read directories and add the tune data to the database.

The ABCToolsClass contains helper functions used in the ABC importer and calls upon the abc2midi library via the Tunepal Class to retrieve midi sequences from ABC, and melodic contour (parsons code).

The ABCTools is a class written in C++ which performs string normalisation. However a further class was required for pre-processing of full ABC files: this is the ABCparser and is a node created in GDScript.

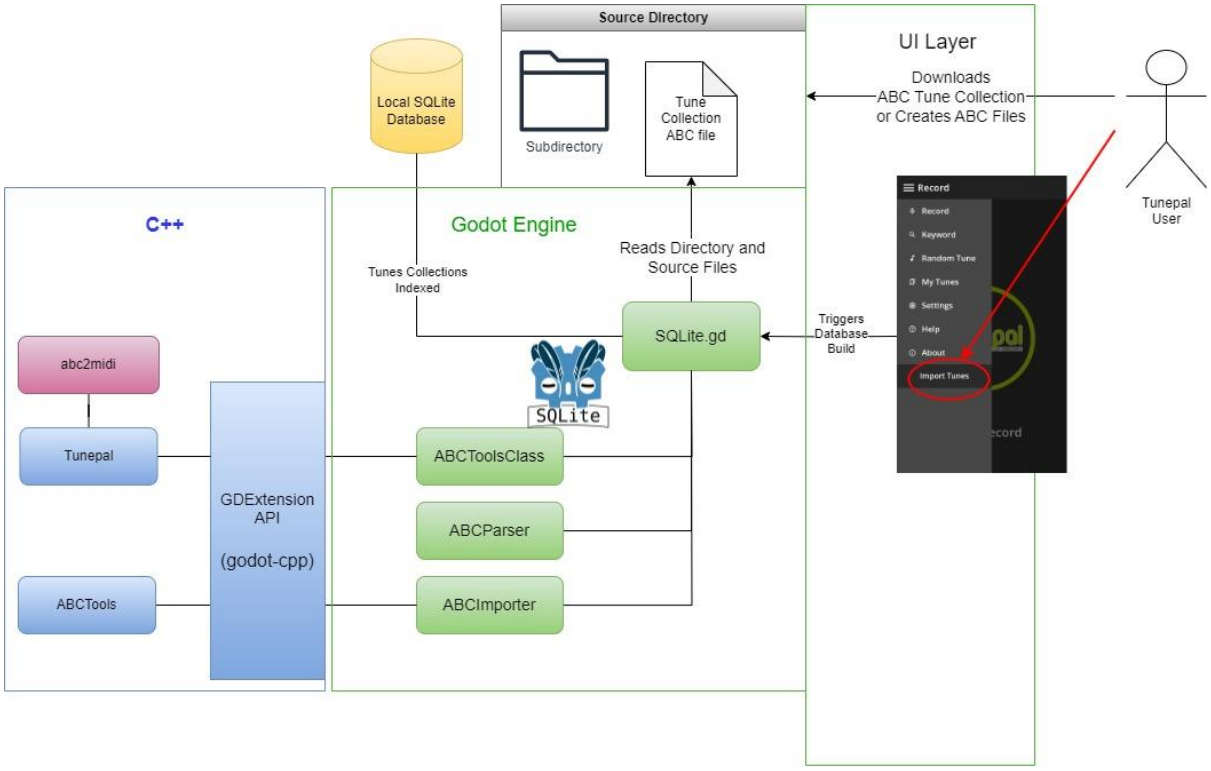

Figure 14 System Design for ABC Importer

The Tunepal and ABCTools classes, written in C++, are made available to the Godot engine through the GDExtension API. The C++ GDExtension is compiled from godot-cpp submodule. The functions in this class can then be called in the scripts used in Godot.

Breaking the system architecture down further for the ABC importer in Figure 15 one can take a deeper dive and examine the principal functions that are utilised for processing raw ABC, and inserting them to the tunepal database. We can see the entry point at the UI layer in "menu-selector" which contains the "import tunes" button on the UI. This calls a FIleDialog node in the *show_directory_select_dialog()* which allows us to open the device filesystem choose a directory on the device. These functions are contained within the sqlite.gd script file.

This passes our directory path to the ABCImporter node, specifically the *import_files_from_directory()* function. Here, two processes occur: Firstly, the directory is opened and a list of subdirectories is created, a source ID (integer) is assigned and then *import_source_directory()* is called to further carry out the indexing of all the tunes in the directory. Secondly, all files in the base directory (not subdirectories) are passed to the *import_source_directory()* along with the base directory path.

The *import_source_directory()* function looks for ABC files in the directory, extracts source info the files contents, calls the abc parser to process the raw files into uniform blocks.

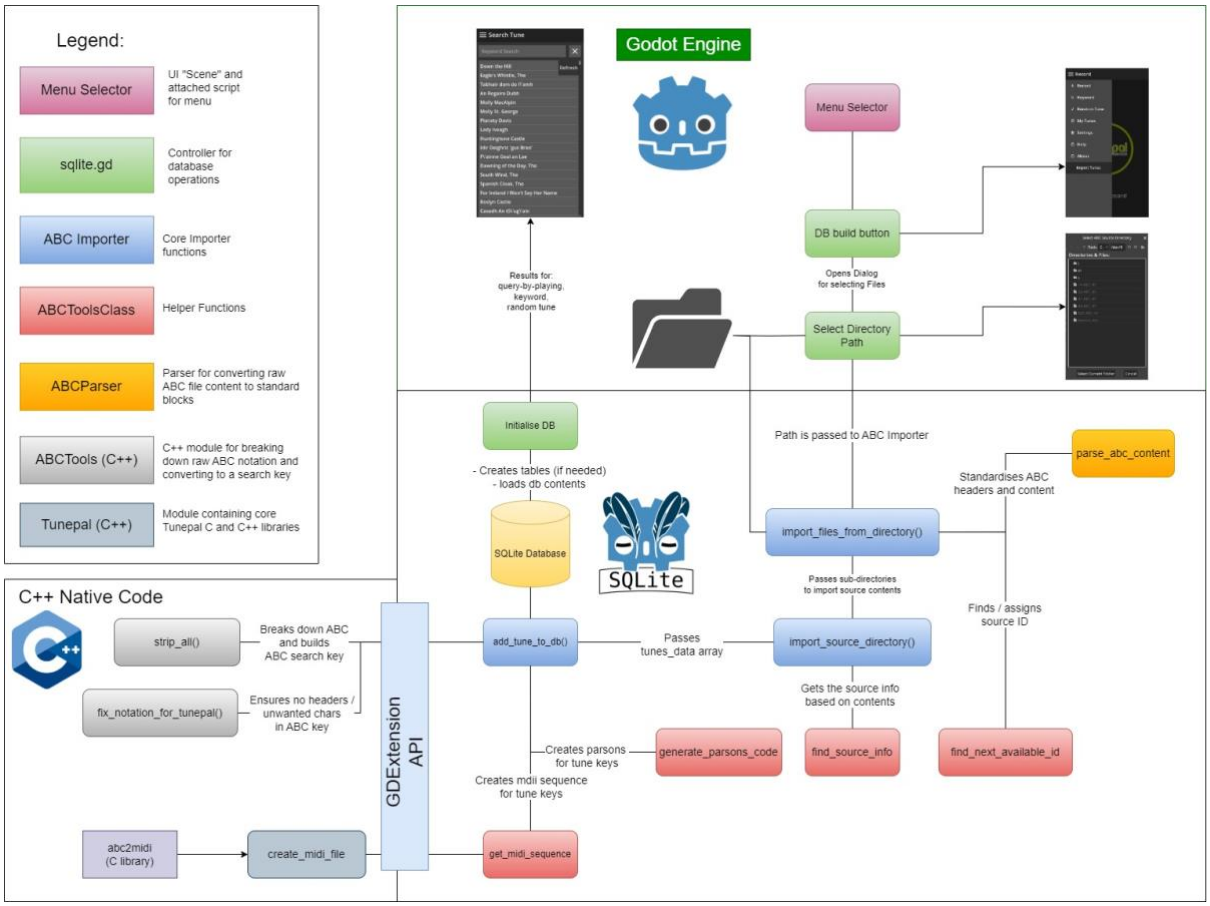

Figure 15 Detailed system design for ABC Importer

The ABCParser's *parse_abc_content()* is responsible for converting the raw ABC file, which may contain extraneous characters, text blocks (e.g. comments) or whitespace and returning an array of uniform "tunes" that are ready to be indexed to the database's *tuneindex* table. Each "tune" in this array is itself a *dictionary* object comprising key-value pairs denoting the essential metadata contained in ABC headers as well as the abc notation. ABCImporter's *add_tune_to_db()* function adds each "tune" in the array to the database. It contains three SQL queries: one which inserts the data in each "tune" object to the tuneindex table, one which inserts the tune key to the *tunekeys* table and a query that inserts the source information to the *source* table.

The tune key is also a *dictionary* object containing search key, numeric midi sequence and parson's code. The search key is obtained using ABCTools *via* GDExtension which performs "normalisation" on the abc notation. The *create_midi_sequence()* function takes in the search key as an argument and calls the "Tunepal" class (*via* GDExtension) to create a temporary midi file using abc2midi and extracts the midi notes from this file. A simple algorithm is then used to extract the parsons code melodic contour from the midi sequence.

One other operation worth noting here is how the "source" object is constructed. Similar to "tune" and "tunekey", "source" is a *dictionary* object, containing key-value pairs which are inserted to the *source* table on the sqlite database. Each source ID is assigned from either the folder name (if it is an integer) or through an algorithm which finds the next available source ID, using the *find_next_available_id()* function. The *find_source_info()* function assigns a source name based on the folder's name. It also extracts a URL for the source from the file's contents and assigns this value to the "*url*" key. All these functions are present in the ABCToolsClass class. The resulting dictionary can then be inserted into the database by the ABCImporter class' *add_tune_to_db()* function.

5. Planning

The code repository for Tunepal had initially been developed privately by a small team before being made public, with the intention of making this project open-source at a later stage. The project was lacking a structure for planning and team collaboration, and had no open issues in GitHub against which pull requests could be made. This would prove to be a crucial component towards making this project an open-source community.

All initial plans and feature requests from Tunepal's creator were on team discussions which took place on a Discord channel in an unstructured manner. It became evident from the beginning that an agreed development workflow would be required: measures would need to be taken for effective project management and to clarify and outline the requirements for fixes and features. On the team Discord channel it was agreed that a team meeting would be held in which the GitHub issues and other planning for this project could be underpinned. Agile, Scrum, Sprint, Kanban, Waterfall, Prototyping and Iterative were the development methodologies considered for the execution and development workflow of this project.

The agile development cycle offered some of the best tools for continuous collaboration and improvement (Atlassian, 2019), specifically: Sprint and Kanban were selected as the most suitable planning methodologies.

Kanban offers a visual aid in a shared workspace in which all collaborators could edit and monitor the progress of the project. This offers a highly structured, transparent real-time tool for visualising completion of project deliverables (Radigan, 2024). It was ideally suited to this collaboration as all the requests for features, fixes and bugs – previously only found on Discord and email threads could be migrated onto the Kanban board. It also allowed for assignment of these tasks to the different collaborators, and visualisation of progress.

The Sprint development cycle was chosen for execution of the Kanban items, as it is easy to home in on a single task (or more than one small tasks) and use short time-boxed periods as windows for completion of these tasks (Rehkopf, 2019). Two-week sprint block was identified as a suitable sprint window.

6. Implementation

6.1. Early Work and Onboarding

Early onboarding work involved learning how to compile C++, and choosing the best compiler to do so. MSYS2 was used for Windows (as most development was carried out on a Windows machine). However, Linux builds were also carried out on a Windows Subsystem for Linux (WSL) terminal using the gcc compiler. It was necessary to begin learning fundamentals of C++ and its use as C++ bindings for Godot projects, using the godot-cpp GDExtension binding.

This research phase also involved becoming familiar with SCons and the SConstruct build configuration files for C++ builds, as that is what was currently in use. Android Studio was another technology that was needed for emulating Android devices and testing android builds: this involved exporting .apk files in Godot editor for Android and running emulator on Android Studio. It was necessary to become comfortable with using GDScript (the scripting language used in Godot) as a part of this project. However, testing was carried out in the Godot engine's debugger as it allowed for immediate proof-of-concept results from running the application.

6.1.1. Kanban and Sprint Development Workflow Introduction

A professionalised approach to the development workflow was required for visibility on progress, assisting collaboration on features and to pave the way for open-sourcing of the codebase.

All communication occurred in a Discord server, and Creator Bryan Duggan had outlined a list of features during a team meeting to be re-created in Godot and problems to be addressed. A list of challenges (labelled P1 – P7) is given in Table 1.

Challenge	Description	GitHub Issue Number(s)
P1	More modern transcription engine	#15, #6, #7 #3
P2	ABC Importer: Indexing tunes, and import ABC from user	#12, #23, #13, #21, #28, #37, #29, #32, #30, #35, #37, #20, #34, #22
P3	Embedded Browser	# 9
P3	Score rendering	#14
P4	Integrate API Call to Europeana Sounds	#4
P5	Quest Build	#8
P6	Reduce Startup Time	#5
P7	Tune Display: link to thesession.org and YouTube	#10
P8	Reduce High CPU Memory usage of ABC Importer	#37

Table 1 outstanding features and issues to be addressed in re-work of Tunepal

As a part of the early work on the project, a Kanban board was created using Github projects, and issues were placed into "To-Do", "In progress" and "Done" categories. Tasks were assigned to different collaborators. The feature requests and fixes mentioned above were added to the Project's Kanban board and were opened as issues on the GitHub Repository

A table of pull requests as of writing this report is given in Table 2.

Pull Request	Description	Status
#16	Transcriber: Compiled third party libraries, integration to existing transcription	Closed
#18	DB Builder: fetch and index tunes	Closed
#31	Reading from ABC files and populating tables (index and search keys)	Closed
#33	ABC Importer: source directories	Closed
#36	ABC Importer – Improved Performance and indexing sources	Open
#38	Merge ABC Importer	Open

Table 2 Pull Requests

Though this agile development workflow was not fully embraced by all the team members and primarily used by the author, it was a useful tool and lays the groundwork for future collaboration as the team of contributors grows.

6.1.2. Modern Transcription Engine

One of the first objectives of the work carried out for this project, as assigned by creator Bryan Duggan, was to investigate how to compile and integrate an AI-based transcription engine, as a more modern alternative to MATT2. For this, a library had been selected by Duggan: NeuralNote (Ronssin, 2025) which transcribes audio to midi with the aid of machine learning inferencing. This would potentially allow key invariant transcription which cannot be achieved with MATT2. MATT2 is also not effective for certain instruments, *i.e.* can only transcribe *legato* instruments such as flute and fiddle and not staccato style instruments such as banjo or guitar, which is a great problem for many users. NeuralNote offers potential for these instruments to now be included as it claims to be capable of transcribing any tonal instrument.

The existing prototype had included most of the Model and ModelData directories from the NeuralNote AI-based transcription library. However, this was not compiled or integrated into the existing project, is was labelled on GitHub as issue #3: https://github.com/TechnoLukas/TunepalGodot2/issues/3.

Research was carried out into how to integrate this library as a third party submodule, with some basic instructions available on the NeuralNote repository (Ronssin, 2025). The machine learning model data was already supplied as a part of NeuralNote. It was discovered that further dependencies would be required to enable the audio to midi

transcription: RTNeural, and ONNXRuntime. ONNXRuntime is an inference and training accelerator used for running the machine learning model (ONNX Runtime Documentation, 2025). RTNeural is a C++ based neural network inferencing engine (Chowdhury, 2021). ONNX Runtime was installed on the machine and its path was passed to the SConstruct script as an environment variable. RTNeural repository was included as a submodule within the project. Once these were all configured, the project could compile without build errors.

However work remained incomplete on integrating this new transcription engine with the existing prototype. The changes resulting in successful compilation of the AI-based transcriber are found at pull request #16: https://github.com/TechnoLukas/TunepalGodot2/pull/16. Progress on this system was shelved in favour of the ABC importer which was considered a more urgent obstacle to tackle.

6.2. Implementation of the ABC Importer

The ABC Importer, as described in section 4.4.1 was requested Tunepal's creator Bryan Duggan, as a core subsystem to the project's progression. Without the ABC importer, Tunepal would not have a modern up-to-date corpus for offline use. It was also requested that the importer could be triggered by a user from the UI and provide the option of indexing one's own tunebooks. Therefore, the initial work on the ABC Importer all happened within the Godot Engine, in GDScript.

The code implementation was based on code originally developed as a part of Bryan Duggan's PhD thesis, a program called corpusIndex.java (https://github.com/skooter500/matt2/blob/master/PhD/src/matt/CorpusIndex.java).

The code is relatively old, and was subsequently refined by his student, Pierre Beauguitte and in a private repository named "DBBuilder". The first step in tacking this was to study the codebase for the original "DBBuilder" and attempt to replicate this for Godot. At this point, the database had simply one "Tunes" table with all the tune data. The Java version of ABCTools was ported to GDScript, this was a significant milestone in getting the early version of the ABC Importer up and running, through a process of trial and error it managed to perform much of the stripping and normalisation correctly. The full JSON dump was downloaded using Godot's built-in HTTPRequest Node. This was written to a file, the file was read, and contents were all inserted directly into the Tunes table. This however did not retrieve a midi sequence or obtain parson's code. However the migration of the normalisation algorithm would form the bedrock for subsequent work on this system and this was merged in pull request PR #18 (Table 2). The "dbbuilder" branch on the repository fork still contains the code for this JSON index-from-url feature to be resumed at a later date: "build from URL" feature is still under development.

However this also led to a simple menu button added to the menu to allow the build to be triggered from the debugger.

Subsequent meetings with the development team highlighted that a different angle was needed for indexing the corpus of tunes: This was, rather than retrieving from a JSON dump, to import from ABC files directly, as this was the preferred format for majority of online traditional and folk music collections. It was decided then to move away from indexing the session.org data from GitHub, although this is subject to current and future work is intended to be incorporated later.

The next significant development on the ABC importer was the capability of adding "tunekeys". In PR #31, much work was carried out in understanding how abc2midi could be used to generate a midi sequence, in a midi file. At this point, abc2midi would create the midi sequence but would also output a midi file and duplicate abc files for every abc file that was processed, however the get_midi_sequence() file would successfully extract the midi notes. An algorithm was developed to then use this sequence to create a melodic contour using Parson's code, similar to that created previously by Bryan Duggan.

This pull request successfully closed three issues: #28, #29 and #30. Issue #28 was concerned with the initial ABC parsing step which was refined prior so that tune dictionaries could be inserted to the tuneindex table: and the correct desired schema for tuneindex was first implemented. Issues #29 and #30 were concerned with creating the midi sequence and Parson's code respectively.

At this point, the importer was capable of parsing, normalising, creating midi, creating parsons and mostly inserting the data correctly to the *tuneindex* and *tunekeys* tables. It was at this point that the ABC importer would need to be expanded to read not just one directory, but through subdirectories as well as the base directory for ABC files. This introduced significant challenges and roadblocks, in which extensive testing trial and error needed to be conducted to overcome various bugs and errors that would cause crashes when for instance, large numbers of files, very large files or files containing comments or different structures were encountered. Much refinement and fixes were carried out as a part of pull request PR #33. One significant achievement which was a part of this pull request was porting ABCTools to C++ which greatly helped improve speed, performance and prevent crashes when normalising full directories of source files.

Following this, one major issue that needed to be addressed was correctly populating a "source" table. The source information such as URLs, source names and a source ID associated with each tune was required to be gathered and indexed to the database. As a part of the work carried out for PR #36, algorithms were developed to search the files for URLs to be associated with a source. In addition, source IDs, names and short names could now be assigned as the primary key for this table. The capability of the importer

was then extended to other formats other than files with the .abc extension; some popular digital tune collections such as Bill Black's Cape Irish (Black, 2024) existed as .rtf files. This file format required addition of further parsing logic to the ABCParser to remove .rtf formatting sufficiently. This includes removing numerous "%" and "\" delimiters throughout the files. This significant achievement greatly increases the scope of the ABC Importer's capabilities.

The ABC importer was then subjected to extensive testing. The ABCImporter was tested on the entire Henrik Norbeck collection (Norbeck, 2021) as well as large parts of Bill Black's collection. With 80 files and over 4100 tunes indexed at once. The entire thesession.org tune collection was downloaded as a JSON dump and converted to ABC using a command line tool. The file containing over 51,000 tunes was run on the ABC importer.

It managed to index 15000 tunes before it crashed. Though an impressive quantity of tunes, it highlights shortcomings in the application and areas for which future improvements will be made on completely revisiting the performance and using low level language such as Go or Rust.

When processing all thesession.org tunes as individual ABC files the ABC importer performed better, successfully indexing over 44100 tunes to the database. Current solutions being implemented include adding performance logs in different scripts to pinpoint performance bottlenecks. One useful feature of Godot 4.4 is the monitoring panel in the debugger where real-time monitoring of static memory can be analysed in real time, or performance monitoring logs can be generated within scripts. These monitoring logs indicate a steady increase in static memory usage while importing large amounts of ABC files. This may indicate a memory leak. Following some small changes, the number of indexed tunes in one run increased to 43,600 such as adding database transactions and clearing up arrays. However, the steadily increasing static memory indicates a possible memory leak. Which is currently the subject of troubleshooting. Issue #37 has been opened to reflect this. Solutions are currently being investigated such as porting the GDScript over to a low-level language to increase processing speed, improve memory safety, and multithreading, however this is still under development. However it was a significant milestone to have built a working ABC importer system, which is capable of processing ABC and indexing these to the local database. The integration of this system to *Tunepal Nua* complete with a UI menu button will allow users to index their own tunebooks and carry out offline searches.

An open pull request (#38) as shown in Table 2 which aims to merge the ABC Importer feature with the main branch in the Tunepal repository. The goal of this will be to enable the ABC Importer to be included in the first release of Tunepal allowing its capabilities to be demonstrated.

7. Conclusion and Future Work

The work carried out for this project culminated in a significant contribution to the project: a system for indexing text-based music collections, based on the ABC-importer feature.

Significant learning and onboarding were required to become fully integrated as a contributor to this project. The teams development workflow was professionalised with the introduction of a project Kanban in GitHub projects and utilisation of Sprint development cycles. This helps move towards a better structured development workflow which will aid in onboarding and managing development team as it grows. The use of GitHub issues addresses one of the objectives towards forming an open-source community for *Tunepal Nua*.

Part of the early onboarding was in doing the foundational work on integrating a machine-learning assisted audio-to-midi transcription engine. This involved integration of third-party libraries (Neural Note) as a submodule within the project and compiling this. Work on this feature had been postponed by the development team early in the project timeline, in favour of building towards a simpler early prototype. However, this phase had proved a valuable task for onboarding and gaining a strong understanding of how *Tunepal Nua* functions when built in the Godot engine. The integration of ML-driven transcription will require further work before it can replace MATT2, which for current purposes is sufficient. This groundwork for a next-generation Tunepal that aims to have key-invariant searching and tune matching on any instrument, not just *legato*-style instruments.

The ABC importer was successfully built and can generate search keys, midi, and indexing source information. There are some known issues with the current importer which have been identified and are being worked on currently. One is the large demand on memory and CPU when processing ABC. GDScript has some distinct limitations which make it less suitable for use in low-level string manipulation programs. Notably, these are the lack of error handling (GDScript does not have try/except blocks as does its close relative, python). It is also dynamically typed and slower than C++ or statically typed programming languages, which may make it a performance bottleneck in some cases. Due to the high overhead of a dynamically typed scripting language such as GDScript, some improvement could be achieved by using C++ as was achieved with abctools.cpp or by implementing all of these in a low-level programming language such as Go or Rust for faster processing speed, precise control over memory management and elimination of overhead processes. Rewriting all the file parsing and normalisation algorithms in Go or Rust be later incorporated somehow into the Godot engine may be far more appropriate for this regarding performance. However, the process of building this system in Godot has proven a vital learning process on this project.

A useful future expansion of this capability afforded by the local SQLite music database in *Tunepal Nua* w would be the ability for users to share or publish their own tunebooks and compositions, similar to how tunebooks can be uploaded on thesession.org (thesession.org, 2025). In addition to this, integration with thesession.org using their API, and integration with Europeana Sounds (as had been featured on the original version of Tunepal) are project goals in the pipeline for *tunepal Nua,* and collaboration with the maintainers of Europeana Sounds API has already been initiated.

It must be noted that this work has led to the creation of an important and essential subsystem which affords *Tunepal Nua* much better capability than its predecessor and vital groundwork laid out for the future development of this application.

Bibliography

Alessandrelli, F., 2021. *Godot Web progress report #8: Progressive Web Apps.* [Online]
Available at: https://godotengine.org/article/godot-web-progress-report-8/
[Accessed 01 2024].

Atlassian, 2019. *The Agile Coach: Atlassian's no-nonsense guide to agile development.* [Online]
Available at: https://www.atlassian.com/agile
[Accessed 01 2025].

Black, B., 2024. *Bill Black's Cape Irish.* [Online]
Available at: http://capeirish.com/
[Accessed 2025].

Chambers, J., 2002. *ABC Music Notation.* [Online]
Available at: https://trillian.mit.edu/~jc/music/doc/ABCtut.html
[Accessed 01 04 2025].

Chambers, J., 2002. *ABC Music Notation: Headers.* [Online]
Available at: https://trillian.mit.edu/~jc/music/doc/ABCtut_Headers.html
[Accessed 2025].

Chowdhury, J., 2021. RTNeural: Fast Neural Inferencing for Real-Time Systems. *arXiv preprint arXiv 2106.03037*.

DragonflyDB, 2024. *Question: What is a scene in Godot?.* [Online]
Available at: https://www.dragonflydb.io/faq/godot-what-is-a-scene
[Accessed 11 2024].

Duggan, B., 2009. *Machine Annotation of Traditional Irish Dance Music,* Dublin: Technological University Dublin.

Duggan, B., 2010. *Tunepal - Disseminating a Music Information Retrieval System to the Traditional Irish Music Community.* Dublin, Technological University Dublin.

El-Kouly, A. S., 2022. *Godot doesn't work with screen readers #58074.* [Online]
Available at: https://github.com/godotengine/godot/issues/58074
[Accessed 02 2025].

Europeana Sounds, 2017. *Europeana Sounds: Europe's sound heritage at your fingertips.* [Online]
Available at: https://www.eusounds.eu/
[Accessed 10 04 2025].

GameFromScratch.com, 2024. *Godot Developed Non-Game Applications.* [Online]
Available at: https://gamefromscratch.com/godot-developed-non-game-applications/
[Accessed 02 2025].

Godot Documentation, 2023. *Scripting: Introduction.* [Online]
Available at: https://docs.godotengine.org/en/2.1/learning/step_by_step/scripting.html
[Accessed 09 02 2025].

Godot Documentation, 2024. *Applying object-oriented principles in Godot.* [Online]
Available at:
https://docs.godotengine.org/en/stable/tutorials/best_practices/what_are_godot_classes.html
[Accessed 09 02 2025].

Godot Documentation, 2024. *Exporting for Android.* [Online]
Available at:
https://docs.godotengine.org/en/stable/tutorials/export/exporting_for_android.html
[Accessed 01 2024].

Godot Documentation, 2024. *Nodes and Scenes.* [Online]
Available at:
https://docs.godotengine.org/en/stable/getting_started/step_by_step/nodes_and_scenes.html
[Accessed 01 2025].

Godot Documentation, 2025. *When to use scenes versus scripts.* [Online]
Available at:
https://docs.godotengine.org/en/stable/tutorials/best_practices/scenes_versus_scripts.html
[Accessed 02 2025].

Godot Engine Documentation, 2025. *What is GDExtension?.* [Online]
Available at:

https://docs.godotengine.org/en/stable/tutorials/scripting/gdextension/what_is_gdextension.html
[Accessed 10 04 2025].

lewdlime, 2016. *GitHub*. [Online]
Available at: https://github.com/xlvector/abcmidi
[Accessed 10 04 2025].

Lorraine Morgan, P. F., 2007. *Benefits and Drawbacks of Open Source Software: An Exploratory Study of Secondary Software Firms,* Maynooth: Maynooth University.

Moine, J.-F., 2016. *GitHub*. [Online]
Available at: https://github.com/lewdlime/abcm2ps
[Accessed 10 04 2025].

nextjs.org, 2024. *Progressive Web Applications (PWA)*. [Online]
Available at: https://nextjs.org/docs/app/building-your-application/configuring/progressive-web-apps
[Accessed 09 02 2025].

Norbeck, H., 2021. *Henrik Norbeck's ABC Tunes*. [Online]
Available at: https://www.norbeck.nu/abc/
[Accessed 2025].

Oesterle, B., 2024. *Making (Non-Game) Software With Godot – Benjamin Oesterle – GodotCon 2024,* Boston: GodotCon 2024.

ONNX Runtime Documentation, 2025. *ONNX Runtime Documentation*. [Online]
Available at: https://onnxruntime.ai/docs/
[Accessed 01 2025].

Radigan, D., 2024. *Kanban: How the kanban methodology applies to software development*. [Online]
Available at: https://www.atlassian.com/agile/kanban#:~:text=In%20Japanese%2C%20kanban%20literally%20translates,in%20a%20highly%20visual%20manner
[Accessed 2025].

Rehkopf, M., 2019. *Scrum Sprints: Everything You Need to Know*. [Online]
Available at: https://www.atlassian.com/agile/scrum/sprints
[Accessed 01 2025].

Ronssin, D., 2025. *GitHub - DamRsn/NeuralNote: Audio Plugin for Audio to MIDI transcription using deep learning.*. [Online]
Available at: https://github.com/DamRsn/NeuralNote
[Accessed 10 04 2025].

Sizov, Y., 2024. *Bosca Ceoil: The Blue Album*. [Online]
Available at: https://yurisizov.itch.io/boscaceoil-blue
[Accessed 01 2025].

Snopek, D., 2024. *C++ for Godot with GDExtension – David Snopek – GodotCon 2024,* Boston: GodotCon 2024.

Snopek, D., 2025. *GitHub - godotengine/godot-cpp: C++ bindings for the Godot script API*. [Online]
Available at: https://github.com/godotengine/godot-cpp
[Accessed 10 04 2025].

Suárez, C., 2024. *fullstack.com*. [Online]
Available at: https://www.fullstack.com/labs/resources/blog/progressive-web-applications-pwa-with-flutter
[Accessed 09 02 2025].

thesession.org, 2025. *The Session*. [Online]
Available at: https://thesession.org/tunes/
[Accessed 10 04 2025].

Wang, A. L.-C., 2003. *An Industrial-Strength Audio Search Algorithm ,* Palo Alto: Shazam Entertainment Ltd.,.

Wyllie, T., 2020. *FolkFriend - A *free new app* for recognising tunes by audio.* [Online]
Available at: https://thesession.org/discussions/44025
[Accessed 11 2024].

Wyllie, T., 2025. *FolkFriend*. [Online]
Available at: https://folkfriend.app/help
[Accessed 02 2025].