Stellenbosch University

Honours project

Music identification tool: Stelhound

Matthew Jenje

supervised by
Prof. Brink van de Merwe

March 22, 2017

Contents

1 Introduction

1.1 Purpose

This document specifies the requirements for the music recognition project, which will forthwith be known as Stelhound. The purpose of Stelhound is to become a music identification tool which, using various techniques, will be able to record a part of a piece of music and report back to the user specific details about the music, such as its name, recording artist and release year. The tool will present similar matches in a list below, showing fewer details but allowing the user to select and pull up more information from a thumbnail. The tool shall also store a history of previous matches which the user may access and view at any time.

1.2 Motivation

The music industry is vast and ever expanding and often times you may hear a song and would like to identify it. The most direct way to do this is to record a part of the song and let a program determine what that song is. There are some applications which already do this such as Shazam and Sound Hound, however these applications use proprietary techniques in their software. This project will attempt to increase the accuracy and speed of a song lookup, as well as creating an open source solution by using Parsons code and concurrent searching instead of Shazams custom fingerprinting or Soundhounds Sound2Sound system.

1.3 Overview

The tool will be designed to run on a computer which either has a built in microphone or has one connected as a peripheral. If the implementation on a computer is effective, the tool may also be implemented for Android.

1.4 Scope

The result of this project will be a fully operational music identification tool. The software is expected to be able to record music, determine what the audio track is that it has recorded and report back to the user the result. If the result is that no match is found, the software will ask the user to try again. The software will also fetch and display partial matches and other suggested tracks with the aim of improving the user experience.

An attempt will be made to have all database queries and string matches done in parallel. This will hopefully speed up the data fetching from the database considerably.

2 Constraints

The program will require a constant connection to the internet in order to access the webservice which contains all of the music Parsons codes. It will also require a microphone interface in order to record the music which the user wishes to search for.

The tracks available on the webservice are limited to what the service has indexed and this may mean that some music will not be searchable. This will depend on how the service is maintained and how regularly tracks are added to the database.

3 Requirements

3.1 External interface requirements

3.1.1 User interface

1. The user shall be connected to the program through a computer running Java with an internet connection.
2. The software may be run on an Android device with an active internet connection.
3. The software shall interact with a user through a GUI.

3.1.2 Hardware interface

1. The computer shall require a connected microphone.
2. The computer shall require networking capability to connect to the internet.

3.2 Functional requirements

3.2.1 Recording

R1. The software shall record music through a microphone attached as a periphery or built in to the device running the software.

- R1.1. Searching shall begin immediately and recordings shall grow longer over time until a match is found.
  Reason: The software's goal is to find a music match in as little time as possible. To achieve this multiple searches shall be conducted on growing recording sizes until a unique result is found.
- R1.2. Recordings shall be checked by the program for matches and returned if true or false.

R1.3. Recordings shall be stored on the device temporarily in order to convert them into Parsons code.
Verification: access and listen back to recordings taken before conversion.

R1.4. All recordings shall be deleted once a match has been found.

R1.5. All recordings shall be done with a bit depth of 16-bit and a sample rate of 44.1KHz.

3.2.2 Database

D1. The Musipedia database shall connect to the software, enabling searching for audio tracks.

D1.1. The database shall be accessed through the Musipedia webservice which uses the Melodyhound search engine.

D1.2. The database shall only be used by the software for fetch requests.

3.2.3 Recording data

RD1. Recordings shall be processed to extract data for searching.

RD1.1 The software shall take an audio file and remove as much unrelated audio data, or noise, as possible from the desired audio track.

RD1.1.1 The software shall attempt to remove 80% of all unrelated audio data.
Verification Noise shall be digitally inserted into a noise-free recording and then removed with the software which will provide controlled test cases for the software.

RD1.2 The software shall take the cleaned audio file and convert it into Parsons Code which will then be used to search through the database.
Verification Correct Parson codes will produce correct matches when searching and this shall be used as a verification technique.

3.2.4 Searching

S1. The software shall search through the database to try and find a match with the generated Parsons Code.

S1.2. Searches shall also return partial string matches as part of requirement I2.4.

S2. Research shall be done into whether concurrent string matching is viable and affective or not.

S2.1. If concurrent searching is viable, it may be implemented into the programming solution.

S2.2. If concurrent searching is not viable, a single threading solution shall be used.

3.2.5 Interface

I1. The interface shall allow users to search for audio tracks.

- I1.1. The interface shall contain a simple search button which will automatically start the record and search functionality of the program.
- I1.2. The interface shall contain a button to stop recording and searching for an audio track.
- I1.3. The software shall terminate the search once a match has been found or the search runs for 10 seconds without finding a match.

I2. The interface shall display search results.

- I2.1. If a match is found, search results shall be displayed as a list of similar matches, the exact match, the year and the album that the audio track came from.
- I2.2. If a match is not found, search results shall be displayed as a list of similar matches.
- I2.3. Searches may be stored to be viewed again later.
- I2.4. Partial search matches shall be displayed below exact matches as a list of similar tracks.

I3. The interface will report all errors and warnings.

- I3.1 The interface shall report internet connection errors.
- I3.2 The interface shall report database connection errors.

I4. The interface shall provide visual feedback when actions are performed in the software.

- I4.1. When the search button is clicked, its style shall change to indicate being active.
- I4.2. When a result is found, the search button's style shall revert back to its initial state.

3.3 Non-functional requirements

N1. Searches shall find matches within 10 seconds of pressing the search button.

N2. The user's search data shall not be stored remotely or sold to third-party companies.

N3. The software shall run on a personal computer.

- N3.1. The software will require Java installed on the computer to run.
- N3.2. The software shall require a microphone peripheral to be connected to the computer

N4. The software may be run on certain Android devices.

N5. The software shall not require any specialized training to use.

N6. The software shall use version control during development.

N7. The software shall be tested thoroughly with a JUnit test suite.

N8. The software shall store some information permanently on the computer. **Reason**: Information for previous music matches will need to be stored on the computer to see search history.

N9. The software shall use searching algorithms which are scalable and efficient due to the time constraint as well as potentially having to sort through incredibly large amounts of data.

References

[1] Avery Li-Chun Wang *Digital Audio Essentials* Shazam Entertainment, Ltd, Palo Alto, CA, USA, 200.