Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Music Tagging – A Social Game

Bachelor's Thesis

Dominic Plangger

dominicp@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Samuel Welten, Tobias Langner
Prof. Dr. Roger Wattenhofer

June 16, 2012

Acknowledgements

My special thanks go to my supervisors Samuel Welten and Tobias Langner who guided, supported and helped me in every phase of this thesis.
Furthermore I would like to thank Professor Dr. Roger Wattenhofer for giving me the opportunity to realize this interesting thesis in the Distributed Computing Group.
Last but not least thanks to my friends who helped me to test and design the application and provided me with great ideas and problem solutions.

Abstract

During this thesis, we developed an application on the Android platform called SOUNDMATE, which generates song–tag assignments while users are playing it. It works as follows: two players get assigned to each other and they have to answer certain questions about a currently played song (about genre, instruments, mood, etc.). The goal is to give the same answers to the questions as your partner. While playing, SOUNDMATE sends all given answers to a central server which collects all answers, processes them and makes them publicly available on the SOUNDMATE website[1].
After developing, we released SOUNDMATE and the tag collecting started. We could observe how well the application spread and whether the users liked it or not. In the end, we could evaluate the quality of the collected tags with a customized quality measurement.

[1] SOUNDMATE website: www.soundmate.ch

Contents

Chapter 1

Introduction

1.1 Motivation

Discovering and downloading new songs is as easy as never before. Nowadays, you can purchase and access music just with a few clicks (for example using Amazon or iTunes). This leads to a drastically increasing size of the average personal music collections. Having more than 5'000 tracks in the own music collection is very common these days.
However, along with this trend arises an inconvenient problem; how do we maintain such big music collections? Often you want to sort the songs by music characteristics like genre, instruments, or mood. Doing that by hand is tedious and practically not feasible any more.
A solution would be to assign each song a finite set of categorized tags which describe the song. For example you could have genre tags like Rock, Punk or Jazz which identify the genre of the song. You could also have instrument tags like Guitar, Trumpet or Violin, which describe what music instruments occur in the song. Having such a song–tag assignment for each song would obviously solve the problem of ordering and sorting the own music collection (more information about music tags at [1]). The only question is where to get these tag assignments from? Assigning them by hand is not a practicable solution. Our idea is to create a social game that automatically generates such song–tag assignments while people are playing it.

1.2 Goal of this Thesis

The overall goal of this thesis is to collect categorized high–quality tags for existing songs. Tag categories are for example genre, instruments, tempo, etc. The collected tags should be freely available on the internet and usable without any license restrictions.

This goal should be achieved in the following way:

- Developing, releasing and promoting a social game on the android platform[1] that automatically generates song tags while people are playing it. The game should send all generated tags to a central tag collecting facility.
- Developing a server which acts as the central tag collecting facility. It should collect and process all tags received by the game.
- Developing a web–server which provides a public interface (i.e. a website) through which one can access all collected tags. The interface should provide a possibility to download all collected tags but also a search function to search for tags of specific songs or artists.

The process of developing a program is composed of different phases. It starts with the requirement engineering where the feature specifications and functionalities are elaborated. Afterwards follows the design and implement phase where the actual program code is written. In the end follows the testing phase where all program functions are tested with respect to their intended behaviour.

1.3 Outline

This thesis is organized in the following way: Chapter two presents some related projects and shows the improvements in this project. Chapter three is about the gameplay of Soundmate; in other words, how to play it and how it looks like. The core of this thesis is chapter four. It explains thoroughly the architecture of the system and motivates possible design choices. Afterwards, chapter five describes the highlights of this thesis, namely the release of the application and the results and insights gained by this project. Finally, chapter six provides a conclusion and an outlook on what can be done in the future on this topic.

[1] Android: http://en.wikipedia.org/wiki/Android_(operating_system)

Chapter 2

Related Works

The idea of wrapping tasks that cannot be efficiently done yet by computers into games for humans is not new [8, 6]. The first game implementing this idea is the ESP game. It follows a specific mechanism: always two players are given a certain object they have to describe (i.e. to tag). The goal is to tag the object using the same words as the other player [2]. In the case of the ESP game the objects are images. Having always two players tagging an object enhances the accuracy of the tags. Since the two players have nothing in common except the object, using words that corresponds to the object is the only reasonable behaviour for the players. As the ESP game is about image tagging and not song tagging, we will not discuss it any further but you can find more information at [7].

Now back to Soundmate. A question that may arise is: does there not already exist a public song tag collection? The answer is yes and no. There exist some public tag collections that contain a pretty large amount of song tags (e.g. the Last.fm database). But the problem is that their tags have low quality and they are not categorized. A low tag quality means that some tags may occur several times with similar names (e.g. Rock and Roll and Rock'n'Roll) or that the tags are often very general and abstract (e.g. Techno instead of Progressive Techno).
Another question that may arise is: does there not already exist a project implementing the idea of collecting song tags with a social game? The answer is again yes and no. There are some similar projects but they differ in some aspects to this one. In this chapter we will introduce two of them, namely MajorMiner and Herd–it, and then show what is new in Soundmate, our project[1].

[1] Two similar projects that we do not cover here are TagATune and Listen Game [2, 5]

2.1 MajorMiner

MajorMiner is a game you can play online on their website[2]. The gameplay is pretty simple. You listen to a song for 10 seconds and you can type words into a textbox which should describe the song. Depending on whether somebody else already entered the same word for the song you get more or less points. You may also receive points in the future if somebody enters a word to a song that you already entered.
The important characteristic of MajorMiner is that you play alone, there is no partner. Furthermore you cannot see the title of the song during playback.

2.2 Herd–it

Herd–it is an application for Facebook [4]. Currently it is unavailable due to unknown reasons. However, Herd–it works as follows: at the beginning you join automatically a group of about ten people (your 'herd'). Then all of the group hear a song and a question about the song appears with a set of possible answers. After you have chosen your answer, you get points depending on the number of people in your group that have chosen the same answer.
The important characteristic of Herd–it is that you play with a group of partners. Furthermore like in MajorMiner, you cannot see the title of the song during playback.

2.3 Soundmate

Before we discuss the improvements we give a brief introduction to Soundmate. It mainly follows the game mechanism of the ESP game. You have one partner and you can listen to a certain song. Then some questions about the current song appear (about genre, instrumentation, etc.) with a set of possible answers. Your goal is to give the same answer to the questions as your partner. After about four questions, the song changes.

2.4 Improvements in Soundmate

As all other projects that we investigated did not spread well, we had to analyse the weaknesses of them. In the following we discuss the main improvements in Soundmate through which we tried to address the weaknesses of the other projects.

[2]MajorMiner: http://majorminer.org/info/intro

The most important improvement is what we call the focus on the soulmate principle. It is essential that a player has exactly one partner to play with. Having one partner makes the game personal. This binds the user much more to the game than having no partner (MajorMiner) or a group of partners (Herd–it). We tried to improve the personal relationship between a player and his partner with different features. For example you can see the name, the avatar and the current location (city) of your partner. In addition there exists a next button that you can use for direct interaction with your partner (the next button is used to ask your partner to skip the current song).
Another improvement in Soundmate concerns the fact that simply tagging songs is not very entertaining. The idea is to enhance the game experience with focus on the aspect of discovering new music. To this end we provided for example a song history, where you can look up all occurred songs. Furthermore we also display the cover art, the title and artist name of the current song. This should ease the process of discovering and remembering new songs. To further enhance the game experience, we also created special effects and introduced a time limitation for answering the questions (see more in section 3.2)
The last bigger improvement in Soundmate is that you can only choose tags from a predefined set of tags. You cannot create your own tags. This big difference to most of the other projects increases the quality of the collected tags drastically. It is not possible any more that you have several names for one tag (e.g. Rock and Roll and Rock'n'Roll). Furthermore, in this predefined set of tags we can assign each tag a category (e.g. genre, instrument, etc.) and even establish a hierarchy between the tags (for example Metal is parent of Speed Metal). This is the reason why we can collect tags with higher quality that are categorized.

Chapter 3

Soundmate – The Game

Soundmate is a collaborative game developed on the Android platform. The concept of Soundmate is simple. Start it and you get assigned to another player who is currently playing Soundmate. Both of you will hear the same song. Then some questions about the song will appear (about gerne, instruments, mood, etc.). To each question there is a set of possible answers (tags). If you and your partner choose the same answer, you score points, otherwise you lose some. After four questions, the song will change and you will hear a new one. Your goal is to collect as many points as possible.

In this chapter, the first section gives an introduction to Soundmate explaining how Soundmate wants to achieve its goal, the tag collecting. The second section is about the core of the application, the gaming screen. The third section focuses on the different features like the profile or highscore that Soundmate offers.

3.1 Introduction

Before we dig into the details of Soundmate, we want to answer the following question: how does Soundmate achieve its goal, namely the generation of categorized high–quality song tags while people are playing it? The answer is simple. The words that you chose as answers to the questions in the game are directly used as tags for the corresponding song. For example, if you choose the word 'Jazz' as an answer to the question 'Genre?', Soundmate adds 'Jazz' as a tag under the category genre to the song. Then this tag is sent to the central server that collects it as a potential tag for the song. It is only a potential tag because a single occurrence of a tag is not enough to be sure that the tag really corresponds to the song. We require a high confidence of a tag to finally assign it to a song. The exact quality measurement of a tag is described in section 5.2.

3.2 Gaming Screen

The gaming takes place on one main screen which is divided into three parts (see figure 3.1). The top most third is about the current partner. It contains the partners name, his avatar and his current location. The location includes the country and the next bigger city close to him. The location is based on the IP address he uses. The central third of the screen is about the current song. It is composed of the song title, artist name, cover art, current points and a next button. The next button can be used to ask the partner to skip the current song. The bottom most third is about the current question. It displays the question phrase and shows below the set possible answers.

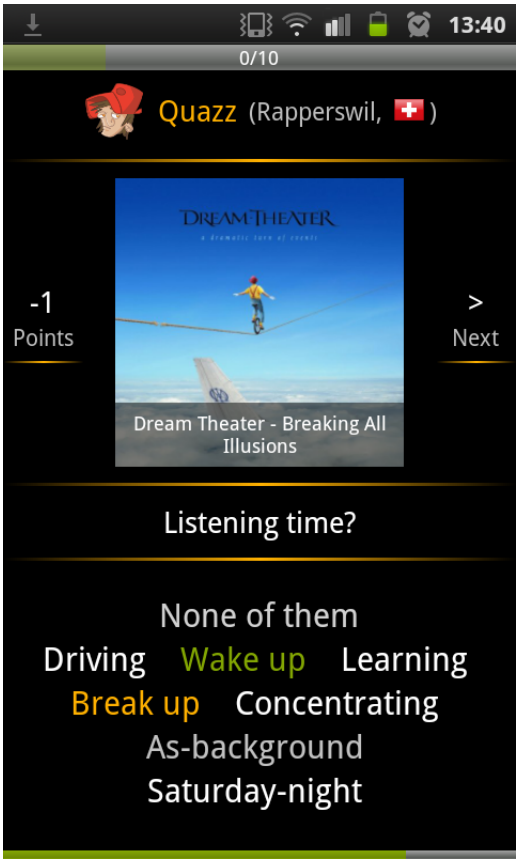


Figure 3.1: The gaming screen of Soundmate. The top most third is about the partner, the central third shows the current song and the bottom most third contains the current questions with all possible answers. The figure shows a mismatch (the player chose 'Wake up' and his partner 'Break up').

Transitions

There are always four consecutive questions for one song. To replace one question with a new one we introduced a visible, intuitive transition which should make clear to the user what happens. The transition is simple: First the old question and answers fade out to the left side of the screen. Then the new question and answers fade in from the right side of the screen. This transition is also applied to the cover art, song title and artist name when the current song should be replaced with a new one.

Bot

If you start Soundmate and there is no other player available, you get a bot as a partner. A bot is a 'computer–player' with a predefined behaviour. Mostly you do not recognize a bot. He possesses also a name, an avatar and a location. The only way to reveal your partner as a bot is on the basis of the answers that your partner gives. As the bot is only a computer, he may give some strange answers sometimes. The behaviour of a bot is as follows: either he chooses the same answer as his partner (with a probability of 30–40%) or he chooses a random answer[1].
Now if another player starts playing Soundmate while you are playing with a bot, then your bot partner is replaced with the real player. The replacement is signalized with a pop up containing the phrase 'You get a new partner'. To effectively replace your partner, the above mentioned transition is applied to the name, location and avatar of your partner. This should further clarify the partner replacement.

Questions

One might ask what exactly the questions are. The following list contains all questions we use including some example answers.

- *Genre?* Answers: *Pop,Rock,Jazz,Hip Hop,etc.*
- *Feels like...?* Answers: *Motivating, Sad, Strange, Sleepy, etc.*
- *Tempo?* Answers: *Fast, Normal, Slow, etc.*
- *Vocal style? What is it like* Answers: *Chant, Hum, Hiss, No vocals, Whistle, Rap, etc.*

[1] If a song was never tagged before, it is hard to do better than choosing a random answer. A little improvement would be to assign each tag a certain overall occurrence probability that the bot can take into account or use external tags from the Last.fm database or similar tag collections

- *Any instruments...?* Answers: *Guitar, Cello, Drum, Violin, Flute, etc.*
- *Listening time?* Answers: *Driving, Saturday–night, Sleeping, Break up, Being–creative,etc.*

There were two more questions that we took out because they were rather irritating than entertaining for the user.

- *Do you see a color?.* Answers: *red, blue, green, gray, etc.*
- *Dynamics of the song?.* Answers: *soft, normal, loud, etc.*

Answer Sets

The answer set of a question is the set containing all possible tags, that might be displayed as answers to the question. The point is that we often cannot display the whole answer set because that would overcrowd the smartphone display. In average we can display about seven tags. Most of the answer sets contain lots of more tags.
The answer sets to the questions about vocal style and listening time are unstructured. To decide which tags should be displayed as answers, we simply choose a random subset of the whole answer set. This can be done because the answer sets of these two questions are very small (about 10 entries).
In contrast the questions about genre, instruments, mood and tempo have a hierarchical answer set. That means the tags can have parents and children. A parent tag is more general (e.g. House) and a children tag is more concrete (e.g. Acid House). To further illustrate this structure we provided an excerpt of the genre answer set in figure 3.2.
A hierarchical structure is needed when an answer set contains too many tags to display all of them. For example let us consider the genre answer set. It contains about 400–500 different genre tags. With the hierarchical structure we are able to display only the most import genres at the beginning (i.e. display only the root nodes). This would be Pop, Rock, Electronic, Jazz, Folk, etc. Now if both players choose the same tag, for example electronic, then the next question is only about the sub–genre of electronic (Dance, House, Techno, Minimal, etc.). This goes on until either the players do not choose the same tag or the chosen tag does not have any children.

As it can be seen in figure 3.1, there is always an answer tag called None–of–them. It should be used when none of the given answer tags are applicable. This is possible because we cannot always display the whole answer set.

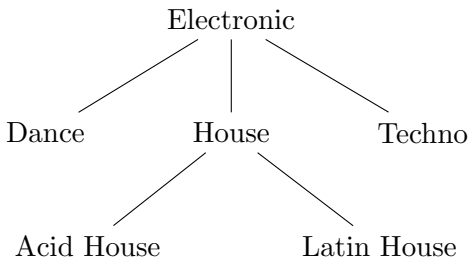


Figure 3.2: An excerpt of the genre answer set. Every child defines a sub–genre of its parent. For example Acid House and Latin House are sub–genres of House.

Time, Matches and Special Effects

To enforce a basic game flow, we introduced a maximum time limit to answer. The time limit is based on how many answers are displayed. It ranges between 7–20 seconds. When the time runs out, a random tag is chosen. The time left is indicated through a progress bar at the bottom of the screen (see figure 3.2). The color of the progress bar goes from green to red depending on how much time already passed.
To further motivate the player to choose an answer rapidly, the points you score respectively lose depend on the elapsed time. Answering quickly results in scoring more respectively losing less points.

There is also a progress bar at the top of the screen indicating how many consecutive matches the player had with his partner. In other words, it tells how often they have chosen successively the same answer. As soon as the they choose different answers, the bar goes back to zero. However a transparent second color still indicates the maximum of consecutive matches they had overall.

To make the game more appealing, we added some special effects. When the player and his partner choose the same answer, an achievement award pops up with a short background music. The award and the background music depend on how many consecutive matches the players had. Let us make some examples. One match results in ‘Not bad’. Five consecutive matches results in ‘Awesome’. Nine consecutive matches get you a ‘Godlike’. If you manage ten consecutive matches, you get ‘Soundmate’ with some fanfares as background music. Achieving ten consecutive matches is very hard and can be seen as another goal of the game besides collecting points. At this point we would like to point out that the received points per match do not depend on the current amount of consecutive matches.

3.3 Features

In the last section we presented the main screen where the gaming takes place. In this section we look at the other screens like the profile, the highscore or the song history that Soundmates offers. All this other screens are accessible through the corresponding button on the start screen.

Profile

The one thing you should do before playing is to adjust your profile (see figure 3.3b). The profile is deliberately kept simple. It consists of a text–box where you can enter your name and a list of possible avatars to chose. It is important to adjust your profile, because your partner can see it.[2] What you cannot determine is your location. Your location (the next bigger city close to you) is automatically determined based on your current IP address.
One might ask what the purpose of avatars is. Obviously they are not needed for the gameplay. They exist to enhance the personal relationship between the player and his partner. It is a further possibility to customize ones profile. The avatars are intendedly chosen to be as different as possible. One is male and sends a cool and strong signal, the other is female and sends a funny and likeable signal and so on. Last but not least we should not forget the simple fact that in general people like choosing avatars.

Highscore

The highscore is divided into two parts (see figure 3.3c). The upper part is the global highscore. It contains the top five players including their best score, their country and the date when they achieved their best score.
The lower part is about the personal highscore. On the left side is a listing containing the numbers how often each achievement award has been received. The right side contains:

- Your best score (the maximum points you have ever reached)
- Your longest consecutive match
- Your current rank based on your best score

Song History

Like the name already suggests, the song history contains entries for all songs that you encountered while playing Soundmate (see figure 3.3d). An entry

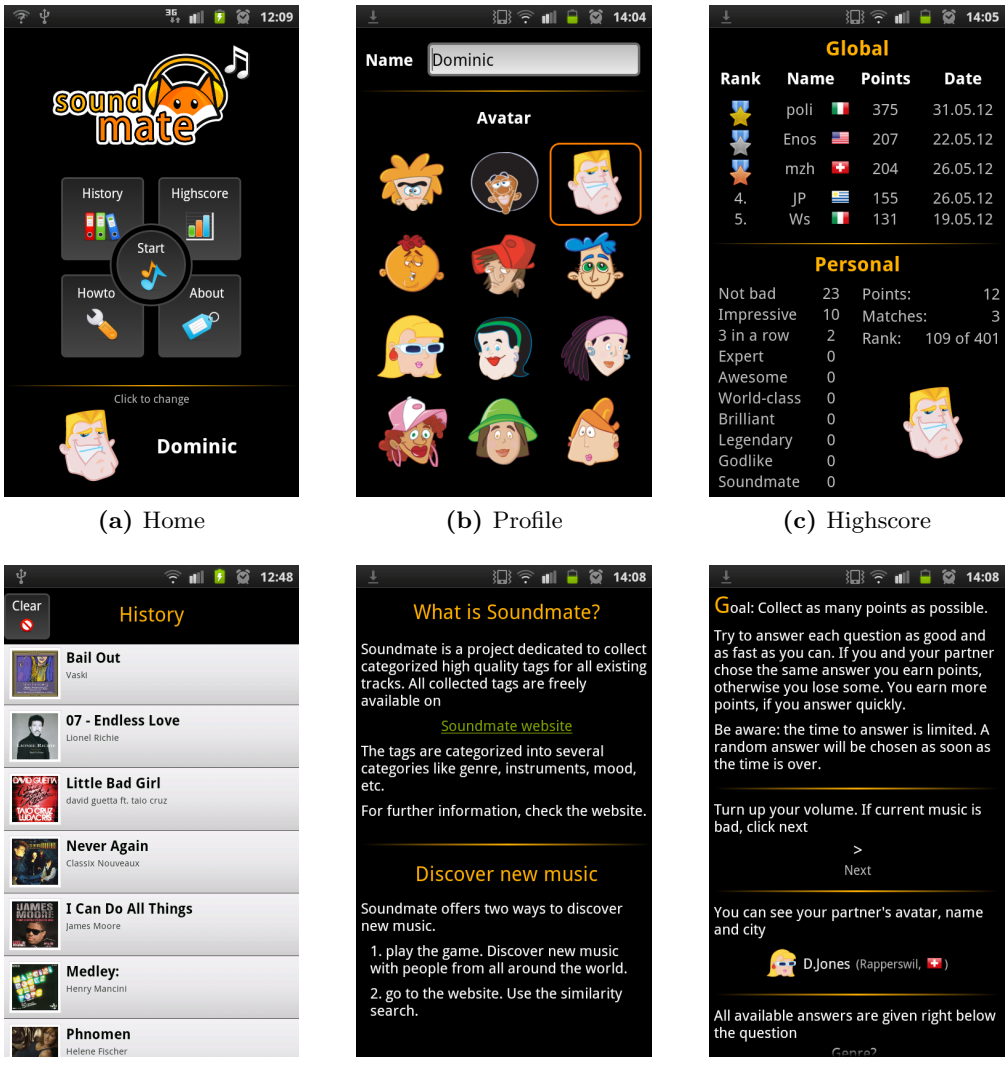
[2] Looking at figure 3.1, you can see in the top most third of the screen the name and avatar of your partner.

consists of a song name, an artist name and a cover art of the corresponding album.
The purpose of the song history is to enhance the music discovering aspect of Soundmate. If you come across a song you like while playing, you mostly do not have the time to write down the name of the song. Using the song history, you can conveniently go through all heard songs after you finished playing.

About and Howto

The about screen explains the greater good behind Soundmate (see figure 3.3e). Namely the collecting of categorized high quality tags. It may further motivate people to play this game if they know, that they contribute to a good cause. To this end we also reference the Soundmate website [3] which contains a counter for the currently collected tags.
The howto screen introduces the player into the gameplay of Soundmate (how it works and what the goal is. See figure 3.3f).

[3] Soundmate website: www.soundmate.ch



(a) Home **(b)** Profile **(c)** Highscore

(d) Song History **(e)** About **(f)** Howto

Figure 3.3: The different screens of Soundmate (except the gaming screen, which can be found in figure 3.1). The screens c–f are accessible through the corresponding buttons on the home screen. The profile can be accessed by clicking on the avatar or the name.

Chapter 4

System Design

In this chapter we describe the system architecture and its design. The system has three different components. There is the client (the Android application Soundmate), the server that organizes the tag collecting and the web server that provides a public interface (i.e. a website) for accessing the collected tags. First we focus on the overall structure of the system. The overall structure defines the relationship between the different components, how they communicate with each other and which roles (i.e. tasks) each component has to fulfill. Afterwards follows a detailed description of each component. All three components are developed in Java.

4.1 Overall Structure

The overall structure of the system is best described by a diagram (see figure 4.1). It is basically a simple client–server model extended with a standalone web server that accesses the tag database. As the web server provides an interface to the tags, access to the tag database is required.
The central server communicates with all clients, whereas the clients do not communicate directly with each other. All communication between the clients comes and goes through the server. The server is the central organization unit that coordinates the gameplay between the players.

4.1.1 Component Roles

To make clear which component does what, we would like to give an overview over the distribution of the most important tasks between the components. We tried to follow the thin–client paradigm as much as possible. In other words, we tried to do most work on server side. This has some advantages: after releasing, updating the server is much easier than updating every client. Furthermore, a

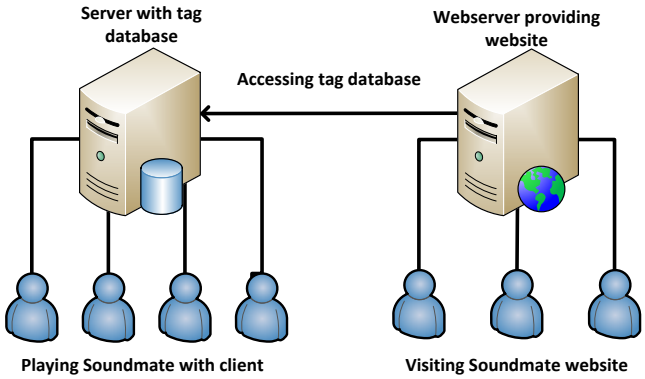


Figure 4.1: The overall structure of the system. The Android clients send all generated tags to the central server. The latter collects all tags and maintains them in a local database. There is no direct communication between two clients, every message comes and goes through the central server. The web server provides a public website through which the tag database can be accessed.

central architecture is less complex than a distributed one. And last but not least, as most smartphones have little computing power, they reach their limitations very fast.

Client

As already mentioned, most work is done on the server side. Therefore the clients tasks are more or less just displaying the contents received by the server and send the user actions to the server. The following list contains a few more details on the tasks of the client.

- Sending a start request to the server when the user wants to start playing.
- Informing the server about all user actions; actions like giving an answer to the questions, making a next request or quitting the game.
- Displaying the contents received by the server. This includes the information about the partner (his name, location and avatar), the current song title and artist name, but also the cover art of the corresponding song album. The server sends only the link to the cover art, thus the client first has to download the image from the target location. Furthermore the client has to display the question phrase and the corresponding answer set, which he receives also from the server.
- Managing the playback of the current song. We use internet radio stations as a source of music (this decision is motivated in section 4.2). It works

as follows: the server sends a link of a chosen internet radio station to the client. Then the client starts streaming from the radio station. As soon as enough music is downloaded, it starts the playback of the song.

Server

The following list contains all tasks for which the server is responsible.

- Accepting client start requests, creating pairs between new clients and starting a new game for each new pair.
- Managing and synchronizing the game of each pair. That means telling each client which radio station he should listen to, who his partner is (i.e. his name, location and avatar), what the current question including the answer set is, where to find the cover art of the album and of course what the current song title and artist name are. Furthermore, the server has to exchange the answers between the two players (synchronizing the two players).
- Maintaining a list of possible internet radio stations. The server has to know which song is currently played on which radio station, so that it can direct the clients to the desired radio stations.
- Maintaining a database that contains for example all collected tags, all users, the user highscores, a log, etc.

Web Server

The web server is not part of the game. Its main task is to manage the public access to the collected tags. To this end, the web server provides a website that allows downloading a dump of all collected tags. The website also contains a search function through which one can search for tags of specific songs or artists. Last but not least, the website references the Soundmate application on Google Play and explains the main purposes of Soundmate.

4.1.2 Communication Protocol

A central aspect in the system architecture is the communication between the components. As the web server does not directly communicate with the other components, we only have to discuss the communication between the client and the server.
The communication is based on string messages sent and received using sockets. The messages have a header and a body. The header is a unique string that identifies the message type. The message body is a JSON string containing the

name–value pairs that we want to transmit. JSON is a simple human readable format for data exchange. To create the JSON string, we used the GSON library from Google[1]. It allows to convert (serialize) objects into strings with JSON format and vice versa (deserialize). This may sound complex, but looking at listing 4.1 should make the message structure clear.

```
1 StartRequestMessage
2 {
3         "name": "Dominic"
4         "userID": "550e8400-e29b-11d4-a716-446655440000"
5 }
```

Listing 4.1: An example of a start request message. We want to transmit a player name and his unique ID. The first line is the header (a unique string) identifying the message type. The other lines are the body: an object serialized to a JSON string using the GSON library.

Now that we defined the message structure, we can talk about the message protocol. There are three main message types. We already saw the first type, namely the start request message. As already mentioned, a start request message is used once at the beginning to inform the server that the user wants to play. The next message type is the new round message. As the name already suggests, it is used by the server to start a new round on the client side. A round corresponds to a song and four questions. As soon as all four questions are answered, a new round begins. Therefore a new round message contains the link to a new radio station and the first question to the new song. The last message type is the answer message. It is used by the client to send the answer tag chosen by the player to the server. Then the server forwards the answer message to the other player. However, before forwarding the message the server automatically appends the next question to the answer message. This simplifies the communication protocol and saves one more message exchange.
An example of an average communication between a client and the server is shown in figure 4.2.

For the sake of completeness we want to mention that there exist five other message types. Two of them are used for the highscore (one to request the highscore and the other to send the highscore). The other three are used to implement the next request (one to do a next request, one to deny and one to grant a next request). A next request can be used by a player to ask his partner to skip the current song. The partner can either accept or deny the next request.

[1] GSON: http://en.wikipedia.org/wiki/GSON

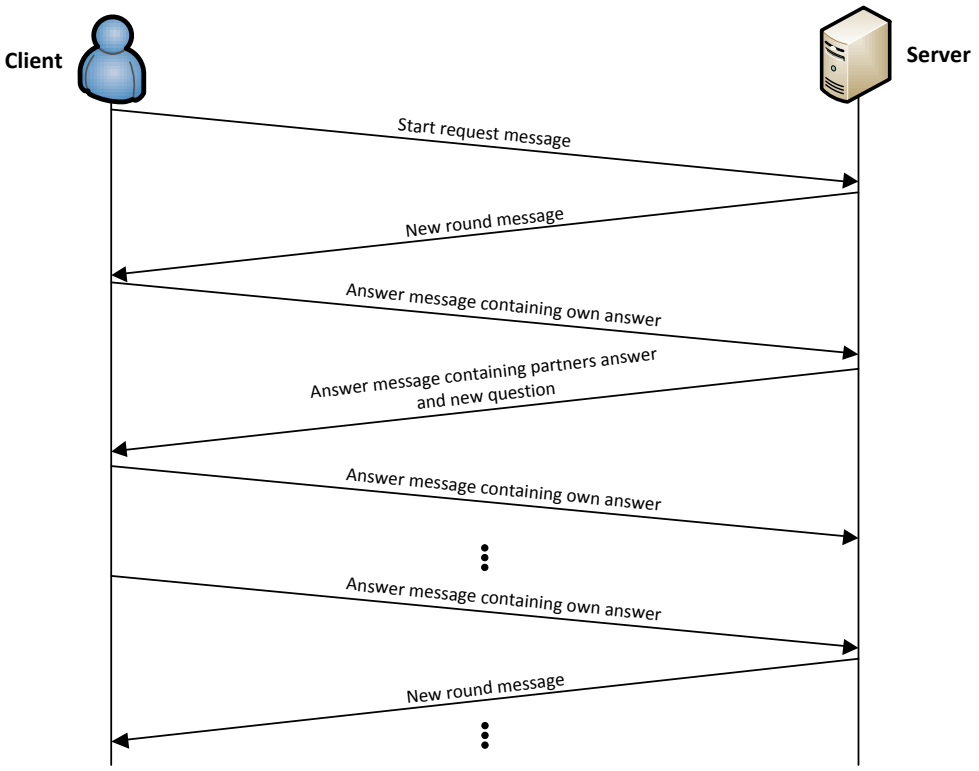


Figure 4.2: An example of the communication between a client and the server. To start, the client sends a start request message. The server answers with a new round message containing the new song, the new partner, the first question, etc. Then the client sends its own answer and receives afterwards another answer message containing the answer of his partner.

4.2 Client

4.2.1 Internal Design

The client is designed to be as stateless as possible. The client mainly sends the input from the user to the server and displays whatever the server sends back. If the server sends a new round message with a new song, then the client simply replaces the old song with the new one. If the server sends an answer message containing the answer from the partner, then the client simply displays the answer from the partner.
Designing the client stateless has some advantages. First the client is way more robust and flexible. The server can change the gameplay (e.g. how many questions per round) without changing the client. A stateful architecture is also often more error–prone. Secondly a stateless architecture is less complex to implement because there are no states to take care of.

The overall structure of the client is simple. Each screen that we presented in chapter three corresponds to an Android activity[2]. In the following we want to discuss some important aspects of the client design like the network IO handling or the music streaming without going into too much implementation details.

Client ID

The central server needs a way to distinguish and identify all clients. After a Soundmate restart, the server still must be able to identify the client. Without that, the server could not count the number of total Soundmate players or save personal highscore data like the current rank or so. Therefore, every client owns a unique ID. The ID is generated locally the first time the client is started. We use UUID's[3] as ID's. Therefore the probability that two clients generate the same ID is so small and we shall assume a total uniqueness of the ID's.
The client ID is specified in every start request message so that the server can identify the client. Furthermore, the client ID is also used when requesting the personal highscore (the rank of the player). An example of a UUID can be found in listing 4.1.

Network handling

The only two activities that need network handling are the activities for the gaming screen and the highscore screen. In both cases, the network handling is done by a background thread. The background thread waits until a message from the server arrives. Then it tries to identify the message type and forwards the message content to the corresponding activity.
The background thread for the gaming activity is also responsible for setting up the connection with the server and sending the start request message. When trying to connect to the server, the thread always tries to connect to an alternative port if the main port failed. This is not essential but it leaves more freedom in cases of server crashes or server updates.

Music Streaming

The music streaming from radio stations works as follows: first the client starts streaming from the radio station until a certain amount is buffered. Then it starts playback and continues with streaming. The goal is to stream enough data at the beginning so that the buffer is never empty and the playback never has to stop until the round finishes (no buffer interruptions). Unfortunately, with

[2] If you do not know Android activities, just think of it as a programming module that is responsible for a certain screen of the game. More about Android activities on: http://developer.android.com/guide/topics/fundamentals/activities.html

[3] UUID: http://en.wikipedia.org/wiki/Universally_unique_identifier

an average internet access it takes up to five seconds for this initial streaming. Obviously it would be unacceptable to let the players wait about five seconds for playback after every song change.
The dangerous aspect of this problem is that not solving it good enough would completely destroy any game flow that might come up while playing Soundmate. Therefore we spent much time finding and implementing an acceptable solution. The main idea is the following: we always start buffering the next *two* songs (using two background threads). Ideally, the buffering of the second song finished completely when the new round begins. As an average round takes about 25 seconds, this is almost always the case. Nevertheless there is one problem: When the players successively use the next button to skip songs, the buffer process may eventually not catch up any more. That means, skipping immediately at least two songs may lead to buffer interruptions in the playback. However practical experience has shown, that on average the buffer system can tolerate up to five or six successive skips before buffer interruptions are needed[4].

Another important aspect of the music streaming is the bit–rate of the radio streams. The client downloads the different songs from lots of different internet radio stations. Every radio station can configure its on bit–rate of the stream. We registered bit–rates from eight bits per seconds up to 917 bits per seconds. Now, as already mentioned the client initially downloads a certain amount before playback starts. We found out that downloading about five seconds of the song is a good threshold to avoid almost any future buffer interruptions. However, to know how many bytes corresponds to five seconds of the song, we need to know the bit–rate of the stream. To this end the server automatically specifies the bit–rate of every new internet radio station in the new round messages (section 4.3 explains how the server obtains the bit–rates). This allows the client to compute the number of bytes to download for the initial streaming.

The download of the cover arts works the same way. The two background threads that are responsible for downloading the songs download also the cover arts. They first download the cover art, which often takes much less than a second. Then they start streaming from the radio stations.

Highscore

The highscore is composed of a global and a personal highscore. The global highscore contains the top five players with their best result. The client always queries the global highscore from the server when the user goes to the highscore

[4]The practical reason for this is that a next request contains some delays. It usually takes at least two to three seconds until the partner answered the request and the message is propagated back to the requester.

screen (no caching). This is for the simple reason that the entries of the global highscore might change at any time. In contrast, most of the personal highscore is saved locally[5]. The only thing the client queries from the server concerning the personal highscore is the current rank of the player (including the total number of players). The current numbers of achievement awards, the player's best score and his maximum matches are saved locally.
To query the server for highscore values, the client uses a background thread that sets up a connection to the server and sends a highscore request message containing the clients unique ID. In return the server sends back the highscore values. We would like to point out that the highscore contains only one entry per client. That means even if one player manages to achieve the best two scores, he would only be displayed at first position. This is for the reason that the highscore is a player highscore and not a point highscore.

Song History

The central question about the song history is: how to remember (save) the details of the occurred songs? For the song title and artist name, the client just writes them as a list into a local file (internal storage). More important is how to save the cover arts. There are two possibilities. Either one writes all images directly into a local file or one just saves the link to the images. The client implements the second solution because it is simpler to save links than images and it does not need as much memory[6]. Every time the user opens the song history, the client downloads all cover arts again. As the images are not very big, the total download finishes after a reasonable amount of time. It is implemented with a background thread that successively downloads the images and informs the song history activity after every image download. In this way we do not have to wait until all images are downloaded.
The last thing to say about the song history is that it has a limit of 100 entries. If the limit is reached, for every new entry the oldest one gets deleted.

Error Handling

Finally we want to cover some aspects of the error handling. The general behaviour in case of an error is to close the current screen (activity), return to the home screen and display the general error message (see figure 4.3). Most errors are due to a bad internet connection. That means that the server closes

[5]For the ones who are familiar with Android: the client uses Shared Preferences to save the best score and the maximum matches. For the achievement awards, it uses internal storage. More on http://developer.android.com/guide/topics/data/data-storage.html

[6]To write the list of links into a file, we use Java object streams. More about that on: http://docs.oracle.com/javase/tutorial/essential/io/objectstreams.html

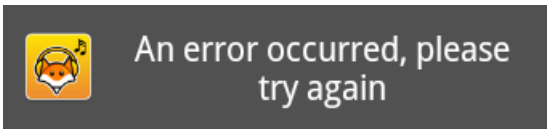


Figure 4.3: The general error message when something went wrong while playing. Most errors are due to insufficient network connectivity (timeouts). In case of an error, the current game is closed and Soundmate returns to the home screen.

the connection to the client because of a timeout. It is also possible that an unexpected internal error happened or that the server closes the current game because of some errors on the server side. However, as it would be very expensive to implement an error handling that distinguishes between all different error causes, we just use a simple general error message.
The general behaviour of returning to the home screen in case of an error might be improvable. There are errors that are recoverable without closing the current screen. For example, in case of a timeout the client may just try to reconnect instead of closing the game. Unfortunately we had to drop any error handling improvement due to time reasons.

4.2.2 GUI

Now that we discussed the internal design of the client, we can move on to the graphical interface of the client. The goal of this section is not to present the GUI (that happened in chapter 3), but rather to motivate the graphical design choices and to give an insight on how the different GUI elements have been built.

Style

Our goal was to create an intuitive, clear and modern design. We came up with a simple color style that is present throughout the whole application. The background is black, text is white and special symbols or titles are orange. We have deliberately chosen a black background which is neither fancy nor creative. But it is neutral, it provides good contrast to most other colors and it does not distract the user from the text on top of it. Using a white text on a black background ensures a good readability of the text. The orange title and symbol color is mainly chosen because the logo and the icon are orange. We could take them from the Jukefox project (figure 4.3 contains the icon and figure 3.3a the logo of Soundmate) .

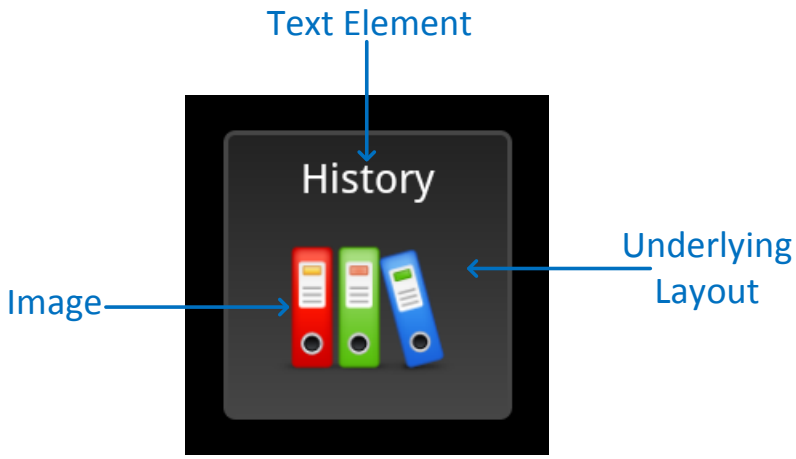


Figure 4.4: The structure of a button on the home screen. The underlying layout defines the shape of the button. The color of the layout is a gradient going from dark gray to light gray. The layout contains also a thin border of an even lighter gray to delimit the clickable area. On top of the layout is a text element and an image describing the action of the button.

Button Structure

Without going into implementation details, we want to illustrate how the GUI elements in Soundmate are built. To this end we discuss the structure of the buttons on the home screen. This type of button did not exist as template, we had to build it using different components. Looking at figure 4.4 wee see that such a button is composed of three different elements that are organized on two levels. On the bottom level is a rectangular layout that defines the shape of the button (the clickable area). The main color of the layout is a gradient going from dark gray to light gray[7]. The layout contains also a thin border consisting of an even lighter gray to delimit the button area. On the second level on top of the rectangular layout is a text element that describes the action of the button. Below the text element is an image further clarifying the action of the button. The goal of the image is that even if the user is not sure what the text element means, he still can imagine what the button does.

After the discussion of the button structure we can now move on to the next component of the system, the server.

[7] Precisely: from #464646 to #222222

4.3 Server

The overall structure of the server is best described by figure 4.5. The server is composed of four different programming modules and the database. The client acceptor module accepts new client start requests and tries to make pairs between the clients. For each new pair it creates a new game instance. A game instance is a thread that manages and synchronizes the gameplay between the two assigned clients. It uses the global radio station handler which provides a list of radio stations and their currently played song. The database utility is a module that abstracts the access to the SQL database.

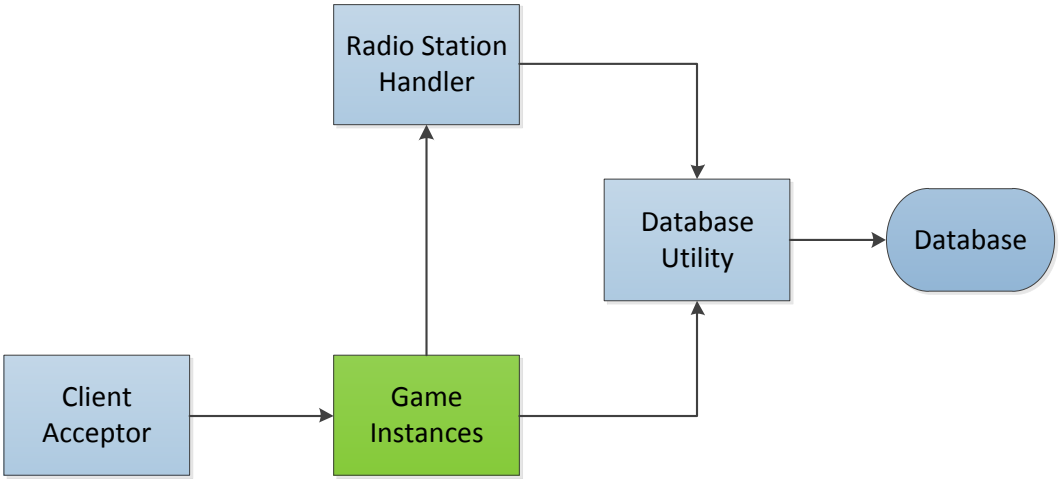


Figure 4.5: The structure of the server. The client acceptor accepts start requests and creates a new game instance for every two players. A game instance manages and synchronizes the gameplay between the two assigned players. The radio station handler provides a list of radio stations with their currently played song. The database utility is used to abstract the access to the database.

The main priority for the server design was robustness. We do not want a server crash after release. We designed the server modules in a way that they are able to recover from all expected errors or failures without affecting any other modules. To this end, we tried to reduce the communication and dependencies between them to a minimum.

In the following we discuss the four different programming modules; i.e. how they work and how they communicate with each other.

4.3.1 Game Instances

The game instances are the core of the server. For every active game between two players exists a corresponding game instance on the server side. A game instance is a single, isolated thread that manages a particular game. Having individual threads for each game that do not communicate or interfere with each other enhances the robustness of the server. In this way, a crash of a single game instance cannot affect any other game instances.
The behaviour of a game instance is basically depicted by figure 4.2. Essentially, it just repeats the following two steps: first it sends a new round message to the two players. Then it exchanges four times the given answers between them. In case of a next request from one of the players, the game instance forwards the next request to the partner. If he grants the request, the game instance cancels the current round and starts the next round. If she denies the request, the game instance informs the requester and continues normally.

A central part of the game instances concerns the new round messages. We know that they contain a radio station and a cover art (links), a song title, an artist name, a question and a set of answers. The question we want to answer is: how do the game instances choose these values? The radio stations with the song titles and artist names are retrieved from the global radio station handler, but more on that later. The questions for each round are chosen randomly, except that the first question is always about the genre. The answer sets of the questions are saved in the database. The game instance uses the database utility module to retrieve the answer set for a certain question. Which answers of the current answer set are finally used depends on the question and is described in section 3.2. What remains to explain is where the game instances get the links to the cover arts from.

Cover Arts

Here it gets a little bit trickier. The following list contains all sources where the game instances try to find a cover art for the current song.

- Amazon Product Advertising API
- Last.fm database
- CDDB and FreeDB databases

The Amazon Product Advertising API allows access to all cover arts from Amazon's music products. We retrieve most cover arts from this source. The API is accessible with either a SOAP or a REST protocol. We use the REST protocol as it is more lightweight. Listing 4.2 shows an example REST query. The query

specifies among others a set of search keywords, the type of the result items, an access key and the associate tag. The last two are mandatory[8]. The result of such a query is an XML file containing a link to the cover art (if available). If there is no result for a certain song, we try the query again but just using the artist name as a keyword. This may lead to a slightly wrong cover art, but that is still better than no cover art.
The Last.fm database also contains cover arts for a certain amount of albums. Unfortunately, it does not provide information on which song corresponds to which album. And as we only know the song that is currently played on a radio station, we need this information. Therefore we use the Last.fm database combined with the CDDB and FreeDB database which provide exactly such a album–song relationship. But as the average image quality of the Last.fm cover arts is rather bad, Last.fm is the last resort only in case the Amazon Product Advertising API did not give any results.

```
1  Http://ecs.amazonaws.de/onca/xml
2          ?AWSAccessKeyId=*********
3          &AssociateTag=*********
4          &Keywords=One%20Republic%20All%20the%20Right%20Moves
5          &Operation=ItemSearch
6          &ResponseGroup=Images%2CItemAttributes
7          &SearchIndex=All
8          &Service=AWSECommerceService
9          &Timestamp=2012-06-07T17%3A28%3A45Z
10         &Version=2012-04-12
11         &Signature=ONkMbCX9XiNHBdAh94mqtllRkGfwINrw0f378G0xers%3D
```

Listing 4.2: An example REST query used to retrieve the cover art of the song 'One Republic - All the Right Moves' from the Amazon Product Advertising API. The Keywords parameter specify the set of words to search for in the Amazon product database. The ResponseGroup defines the type of the result items. The signature is computed out of the remaining query.

4.3.2 Client Acceptor

The client acceptor is responsible for managing all new clients that want to start playing. For every two clients, it creates and starts a new game instance. If there is no partner for a client, it creates a new bot that acts as partner for the client. The client acceptor listens on a specified port if any new start request message from a client arrives. As soon as one arrives, it puts a player structure representing the player that wants to play into a global queue and continues listening on

[8] For more information about the query structure, go to http://docs.amazonwebservices.com/AWSECommerceService/latest/DG/Welcome.html?r=2392

the port. Parallel to that exists a scheduled task that periodically goes through the mentioned player list and tries to create pairs between the contained players. For every pair, the task creates and starts a new game instance. From then on, the game instance is responsible to start and handle the game between the two players.

4.3.3 Radio Station Handler

The radio station handler provides a public list containing links to valid internet radio stations including the song titles and artist names of the currently played songs. The list is accessed by the different game instances (for every new round) to get a new internet radio station.

Motivation

Before we dig into how the radio station handler works, we want to say some words about why we chose radio stations as our source of music. We have a database table containing about 12'000'000 different songs, which we got from Last.fm. Our goal is to finally assign tags to all of these songs. Therefore we need a music source which can (at least eventually) provide all of these songs. Some researches yielded at least three possible music sources that we can use: Jamendo, Youtube and internet radio stations. Jamendo is a free public song collection where artists can add their songs under public domain. The problem with Jamendo is that it only contains about 350'000 songs. Using Youtube as a music source might be a good choice at first sight. However it proved to be hard because of possible license restrictions and the fact that most of the songs exists several times with similar names. Therefore a better solution is to use internet radio stations because just directing clients to radio stations surely does not violate any license restrictions. Furthermore, one can assume that using lots of different radio stations make sure, that a reasonable fraction of the 12'000'000 songs will eventually occur.

Structure

Now after we motivated the choice of radio stations, we can talk about how the radio station handler works. In our database is a table containing about 50'000 links to different internet radio stations. We got these links from different websites providing such lists. The central questions is: how does the radio station handler know, which song is currently played on which radio station? There are at least two ways to find that out. One could periodically start streaming a short time from the radio stations just to take the song information out of the included metadata of the stream. The problem with this approach is that

lots of radio stations do not like it. It distorts their user statistics because we would periodically start and stop streaming from their radio station. The second approach – the one we implemented – is to use their status website. As all of our radio stations use either SHOUTCAST or ICECAST as a stream server, their status websites have all the same structure (at least if they *have* a status website). And this structure includes also the currently played song of the stream. That means the second approach is to read the currently played songs out of the radio station websites. Figure 4.6 shows an excerpt of an example SHOUTCAST status website.

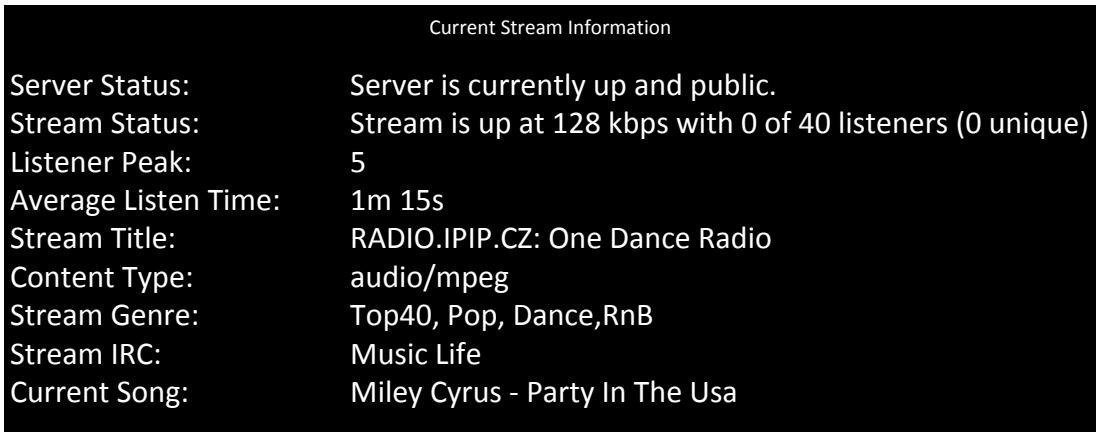


Figure 4.6: An excerpt of a SHOUTCAST radio station website. The website provides information about the currently played song. The radio station handler uses this fact to periodically check the websites of all radio stations to know which radio station plays which song.

To realize this approach, we have a set of so–called stream crawlers and one global list which contains the information which radio station plays which song at the moment. A stream crawler is a thread to which about 20 radio station are assigned. It goes periodically (about every ten seconds) through the websites of all assigned radio stations, and extracts the current song out of the website HTML. If any radio station changed its current song, the stream crawler replaces the old song of the radio station in the global list with the new song. We have about 20 stream crawlers that are working in parallel. In this way we have one global list that provides the current song information for about 400 radio stations.
One problem with this approach is that we do not know, when a song will change. It would be irritating if we direct a client to a radio station and the song changes during the round. To solve this problem, we use just radio stations that have made a song change in the last 90 seconds. As most songs have a length of at least 3 minutes and an average round finishes after 25 seconds, this solution reduces the possibility of a song change during a round to a minimum.
Another problem is that there are some radio stations that are broken and do

not send any music or that often stream conversations instead of music. To eventually remove these bad radio stations from the global list, we periodically replace the assigned radio stations of every stream crawler with better ones. To this end we assume that the players make next requests if the current radio station is bad. To remember how often a next request has happened to each radio station, we maintain a 'next counter'. Using only the radio stations with the lowest next counter for replacement, we eventually use better and better radio stations. However, one might say that some players make next request just because they do not like the current song and this would increase perhaps the next counter of a good stream. That is true, but we want to allude to the fact that the next counter of a stream is just increased if the other player accepts the next request. Therefore, the next counter of a good stream is only increased if both players do not like the current song.
Finally we want to point out that this approach is only on best effort basis. The format of the current song on the SHOUTCAST or ICECAST websites differs from website to website. And we can only use radio stations that play songs that are contained in the list of the 12'000'000 songs that we know. That means there are lots of radio stations we cannot use[9]

4.3.4 Database Utility

All database queries that the server needs are written in the database utility module. Our database is an SQL database, thus the database utility contains just a set of SQL queries. Every database access is done over the database utility. As all stream crawlers and all game instances use this module, the database accesses are highly parallelized. Creating a new database connection for every query could be a bottleneck. Therefore we make connection pooling using the library c3p0[10]. Basically connection pooling is the process of caching database connections for future use. In this way, one does not have to create a new connection for every query.

4.4 Web Server

The web server is the third component of the system. Its purpose is to make all collected tags freely available to the public without any license restrictions. As the web server is a rather small component, we can keep the discussion short.

The web server provides a website for Soundmate using the HttpServer library from sun. The two most important contents of the website are a link

[9] From the initial 50'000 radio stations that we know, there are finally about 15'000 which provide a status website and stream in a usable format.
[10] c3p0 manual: http://www.mchange.com/projects/c3p0/

to the current tag dump and the search function. The tag dump is an SQL file containing a dump of the database table with the collected tags. The table has the following columns: song name, artist name, tag name, tag category, tag quality, quantity and song ID. The tag quality is the quantity divided by the number of other tags collected for the same tag category. The search function can be used in three ways. If you specify just the artist, then the website lists all songs of that artist[11]. If you specify just the song name, you get a list of all artists which produced an equally named song. If you specifying the artist and song name, then the website lists all available tags for this song and suggests a list of similar songs (according to the tags).

[11] Unfortunately, querying all tags for a certain artist would take too long because some artists have up to 7'000 songs in our database. Therefore we cannot provide an artist to tag search function.

Chapter 5

Promotion and Results

In this chapter we first talk about the promoting that we have done for Soundmate. Then we show some interpretations and illustrations of the data gathered during the first several weeks after release.

5.1 Promotion

Since the game experience of Soundmate is much better when playing it with a real partner instead of a bot, it would be desirable that there are always some people playing it. Therefore we used several places to promote and spread Soundmate.

The first possibility to promote an Android application is to upload it to multiple 'App–stores'. We uploaded Soundmate to Google Play, Getjar and the App Center of Androidpit. The second possibility is to directly contact IT–blogs asking them to spread the word about the application. We contacted lots of different IT–blogs including Gizmodo, Androidpit, Engadget, etc. Unfortunately, we did not get a big reaction from them. Another possibility is to present the application in forums or similar platforms. We assume that we reached most of our players through the forums from Androidpit, Androidcentral and Reddit.

5.2 Results

5.2.1 Quality measurement

By now, we have collected about 15'000 tags. That is enough to make a first interpretation on the quality of the collected tags. As our goal was to collect high–quality tags, we would like to have a quality measurement for each collected tag describing our confidence that it accurately describes the respective song. A first approach was to take the relative occurrence rate of the tag as measurement.

For example if we have ten times the tag Guitar and five times the tag Drum for a certain song in the category instruments, the tag Guitar has $\frac{10}{10+5} = \frac{2}{3}$ and the tag Drum $\frac{5}{10+5} = \frac{1}{3}$ confidence. However this example also shows a severe flaw of this strategy. It is not possible to have more than one confident tag in a category. If there are two tags that occurred exactly equally often, they both just have a confidence of 0.5. Therefore we extended this measurement in the following way: we consider all tags that occurred at least about 80% of the highest occurrence frequency as *maximum tags*. When computing the confidence for maximum tags, we do not include any other maximum tags in the computation. For not maximum tags, the computation remains the same. For example lets say we have the following three instrument tags for a certain song: 20 times Guitar, 18 times Violin and six times Piano. As 20 is the highest occurrence number, Guitar and Violin are maximum tags. Therefore Guitar has $\frac{20}{20+5} = \frac{4}{5}$ and Violin $\frac{18}{18+6} = \frac{3}{4}$ confidence. The computation of the confidence for Piano remains the same as it is not a maximum tag: $\frac{6}{6+18+20} = \frac{3}{22}$. In this way we have also a more explicit distinction between good and bad tags. The latter have often very low quality numbers whereas the former have high quality numbers. Normally, there is not much between.
To further illustrate the concept of the quality measurement, figure 5.1 shows the collected maximum tags for the song 'NOFX – Bob' with their confidence. After listening to the song, it gets clear that most of the tags are justified. NOFX is a punk–rock band and the song Bob contains a guitar, heavy screaming and is fast.

NOFX - Bob

Tag name	Category	Quality	Quantity
Positive	Mood	0.33	8 of 24
String Instr.	Instruments	0.5	8 of 14
Punk	Genre	0.36	24 of 65
Fast	Tempo	0.55	11 of 16
Scream	Vocal style	0.4	6 of 13
Break up	Listening time	0.45	5 of 14
Fighting	Listening time	0.45	5 of 14

Figure 5.1: The collected tags with the highest occurrence rate for the song 'NOFX – Bob'. The tags describe pretty good the real characteristics of the song. The quality measurement is a function of the relative occurrence of the tag.

We want to point out that there are several other possibilities to compute

the confidence of a tag. Chapter 6.1.1 presents a further quality measurement that is probably a lot more accurate. However, due to time reasons we could not implement it.

5.2.2 Tag Popularity

One general problem with music tagging games is, that the players will always try to *game* the system [3]. For example, they often choose generic tags like Pop over less common tags like Melodic Death Metal. To prevent such a behaviour, one can forbid popular tags or – like we did – create a hierarchical tag structure and force the player to choose less common tags (child tags) after they have chosen generic tags (top–level tags).
However, we could observe the behaviour of choosing the most generic tag also between different top–level tags. For example, the top-level tag Pop was chosen much more frequently than all other top-level tags in the category genre. Figure 5.2 shows the distribution between all genre top-level tags. Just to remind you, top–level are the root tags of a hierarchical answer set.
We point out that figure 5.2 does not necessarily prove the mentioned behaviour of the players. It is possible (and reasonable) that the genre of the played songs by the radio stations are not uniformly distributed.

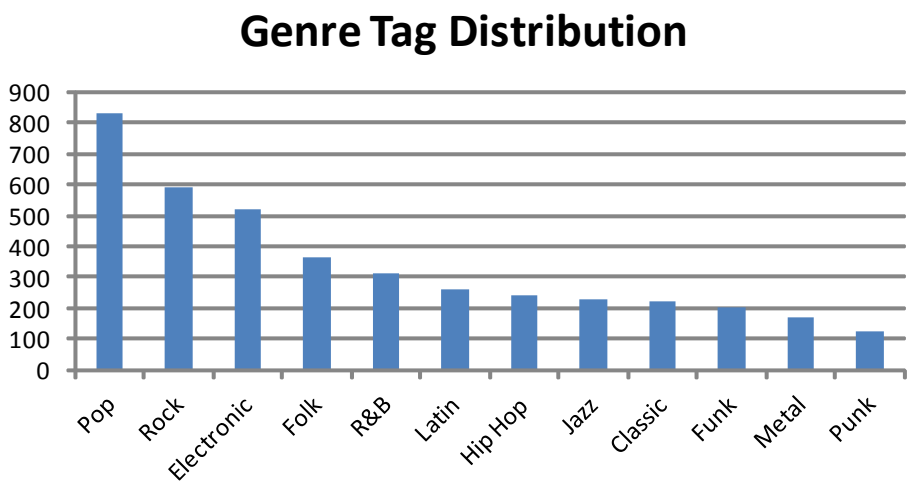


Figure 5.2: The absolute frequency of all genre root tags. Obviously the three most general tags Pop, Rock and Electronic have been chosen much more frequently than the others. This is for the reason that choosing more general tags is a better win strategy than choosing less common tags. The fact that the genres of the played songs may not be distributed uniformly further amplifies the phenomenon.

Chapter 6

Future Work and Conclusion

6.1 Future Work

Thinking in long terms, creating a Soundmate version for iPhone and a version for computers would be surely a good possibility to get more players. As the game suffers from a lack of publicity, a reasonable amount of work should be invested into the promotion and the spreading of Soundmate.

Thinking in short terms, there are some features that could be added to Soundmate. The most important one would be a possibility of doing a self–assessment after every given answer. That could be a simple bar from one to ten where the player can easily indicate after every question how sure he is about the just given answer. The bar should also display the self–assessment value of the other player. These self–assessments can be used for quality measurements (see chapter 6.1.1) but are maybe also another fun factor for the players[1].
Another feature would be the possibility to favour certain songs and (if possible) to save them locally on the phone for future use. This feature would increase the desired aspect of music discovering while playing Soundmate.
Furthermore, a possible improvement concerns the interaction between the two players. We realized that the game makes much more fun, if there are possibilities to interact with your partner. Therefore one could add for example a chat facility or a rating function to rate the answers of your partner.
The last extension would be to assign titles and levels to the players. If they reach a certain amount of points, they can move on to the next level. For each level, they get a certain title.

[1] For example, we could inform a player that he underestimates himself in case when she often indicates a low confidence of the answer but regardless chooses tags with a good quality measurement

6.1.1 Quality Measurement

In the standard quality measurement (we use this term to reference the approach presented in chapter 5.2.2) we considered the answers from different players as equally good. All tag occurrences had the same weight. Unfortunately, in reality, this is not true. There are big differences between the tagging skills of the players. Therefore, the idea is to first make a quality measurement on the players themselves before measuring the quality of the tags. With such a quality measurements of the players, we could assign a weight to each tag occurrence. A tag occurrence from a player with a high quality is weighted higher than an occurrence from a player with a lower quality. These weights could be used to further improve the accuracy of the standard quality measurement.
The quality of a player consists of a set of quality numbers. Each quality number corresponds to one tag category. The quality numbers indicate how often a player has chosen a tag that can be assumed as correct from the corresponding tag category. To find out which tags can be assumed as correct, we simply use the standard quality measurement. To make the quality numbers of the players even more accurate, one could further consider the self–assessment values described above in the computation.

The big benefit of this approach is the following: first we compute the quality of the players using songs that have been tagged enough times so that the standard quality measurement is accurate. Then we can use the quality of the players to state the confidence of tags for songs that have not been tagged enough times yet so that the standard quality measurement is accurate. For example, if we know that five players with very high quality have chosen the same tag for a certain song, we can sate the quality of this tag much higher than the standard quality measurement can.

6.2 Conclusion

Collecting tags with a social, collaborative game is not easy. The critical part is to get enough people playing it. We assume that the developers of the other mentioned projects got the same insight. At the time of writing we have not collected enough tags yet so that they can really be used (due to too little players). However, during this thesis we gained much understandings concerning music tags and the collecting of them. Especially we want to emphasise the benefit of using a predefined, hierarchical set of categorized tags. It is maybe more time–consuming to build one, but it significantly simplifies any further usage of the collected tags.
Another part of the thesis we would like to highlight is the use of internet radio stations as a music source. It was one of the central problems we had to solve in

this thesis, namely where to get the music from. Using internet radio stations, one can listen to high quality music without worrying about license restrictions. With the developed program module, we can retrieve live information about the currently played songs from thousands of internet radio stations. This program module can be easily used in other projects, as it has no external dependencies.

All in all, we think that with the development of Soundmate we made another step towards a revolutionary new way of handling big music collections. By now, we are not there yet, but this state might not last long. One famous blogger and one big IT website promoting Soundmate and the situation maybe looks completely different. Anyway, we are looking forward to an exciting future where creating customized playlists and sorting songs is not in the slightest annoying any more.

Bibliography

[1] P. Lamere, *Social tagging and music information retrieval*, Journal of New Music Research, Routledge, 2008, pp. 101–114.

[2] E. Law and L. von Ahn, *Input-agreement: A new mechanism for collecting data using human computation games*, CHI '09 Proceedings of the 27th international conference on Human factors in computing systems, 2009, pp. 1197–1206.

[3] D. Turnbull, L. Barrington, and G. Lanckriet, *Five approaches to collecting tags for music*, ISMIR 2008: Proceedings of the 9th International Conference of Music Information Retrieval, 2008, pp. 225–230.

[4] D. Turnbull, L. Barrington, G. Lanckriet, and D. O'Malley, *User-centered design of a social game to tag music*, HCOMP '09 Proceedings of the ACM SIGKDD Workshop on Human Computation, 2009, pp. 7–9.

[5] D. Turnbull, R. Liu, L. Barrington, and G. Lanckriet, *A game-based approach for collecting semantic annotations of music*, Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007), 2007.

[6] L. von Ahn, *Games with a purpose*, IEEE Computer Magazine (2006), 96–98.

[7] L. von Ahn and L. Dabbish, *Labeling images with a computer game*, CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems, 2004, pp. 319–326.

[8] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum, *recaptcha: Human-based character recognition via web security measures*, Science **321** (2008), 1465–1468.