



Computer Science

Adrián Coca Lorente  
Pablo Martínez López

# musIQal

A music sharing web-site

Computer Science  
C-level thesis

Date/Term: 12-06-05  
Supervisor: Katarina Asplund  
Examiner: Donald F. Ross  
Serial Number: C2012:03



# musIQal – A music sharing web-site

Adrian Coca Lorente  
Pablo Martinez Lopez



This report is submitted in partial fulfillment of the requirements for the Bachelor's degree in Computer Science. All material in this report which is not our own work has been identified and no material is included for which a degree has previously been conferred.

---

Adrian Coca Lorente

---

Pablo Martinez Lopez

Approved, June 05 2012

---

Advisor: Katarina Asplund

---

Examiner: Donald F. Ross



## **Abstract**

This dissertation describes the development of a web-site which serves a system for developing music collaboratively between different people no matter where they actually are.

The users cannot only create their own music together with other people but can also have an economical profit by selling music tracks to other users and make their music available to the public.

The web-site uses modern programming technologies and contemporary computer capabilities, such as HTML5, JQuery and ASP.NET MVC3.

The dissertation covers the technologies behind this project and also explain how the project has been design and implemented.





# Contents

<b>Contents</b>	<b>vii</b>
<b>List of Figures</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Project Description . . . . .	1
1.2 Project Features . . . . .	1
1.3 Chapter Overview . . . . .	2
<b>2 Background</b>	<b>3</b>
2.1 Existing Systems . . . . .	3
2.1.1 Soundcloud . . . . .	3
2.1.2 Ubetoo . . . . .	5
2.1.3 Comparison . . . . .	7
2.2 Technology Review . . . . .	7
2.2.1 HTML . . . . .	7
2.2.2 HTML5 . . . . .	10
2.2.3 Dynamic Web Pages . . . . .	12
2.2.4 JavaScript . . . . .	15
2.2.5 AJAX . . . . .	16
2.2.6 JQuery . . . . .	17
2.2.7 CSS . . . . .	17
2.2.8 C# . . . . .	19
2.2.9 SQL . . . . .	19
2.3 Tools . . . . .	22
2.3.1 Visual Studio . . . . .	22

2.3.2	Web browser . . . . .	24
2.4	Chapter summary . . . . .	25
<b>3</b>	<b>Design of the system</b>	<b>26</b>
3.1	Requirements . . . . .	26
3.2	Model-View-Controller . . . . .	27
3.3	Database Design . . . . .	28
3.3.1	Table Users . . . . .	28
3.3.2	Table Songs . . . . .	29
3.3.3	Table Suggestions . . . . .	32
3.3.4	Table Negotiations . . . . .	33
3.3.5	Table Contests . . . . .	35
3.4	Program Design . . . . .	36
3.4.1	Code - Database Interaction . . . . .	37
3.4.2	Model - UserManagement . . . . .	38
3.4.3	Model - SongManagement . . . . .	39
3.4.4	Controller . . . . .	43
3.4.5	View . . . . .	43
3.5	Chapter summary . . . . .	44
<b>4</b>	<b>Implementation</b>	<b>45</b>
4.1	User Identity Security . . . . .	45
4.2	Audio Conversion and Mixing . . . . .	47
4.3	JQuery Asynchronous Requests . . . . .	47
4.4	Chapter Overview . . . . .	50
<b>5</b>	<b>User Interface</b>	<b>51</b>
<b>6</b>	<b>Evaluation</b>	<b>57</b>

6.1	Time Planning . . . . .	57
6.2	Acquired Knowledge . . . . .	57
6.3	What could have been done differently . . . . .	58
<b>7</b>	<b>Conclusions</b>	<b>59</b>
7.1	Future work . . . . .	59
	<b>Appendix A: Database Diagram</b>	<b>60</b>
	<b>Appendix B: Model Class Diagram</b>	<b>61</b>
	<b>Appendix C: Whole System Diagram</b>	<b>63</b>
	<b>Bibliography</b>	<b>64</b>



## List of Figures

2.1	Time comments in a song from Soundcloud . . . . .	4
2.2	Soundcloud’s applications for mobile plataforms . . . . .	5
2.3	List of most played songs on Ubetoo . . . . .	6
2.4	Video player on Ubetoo . . . . .	6
2.5	A HTML div element written in short and long forms . . . . .	9
2.6	A HTML img element with the src attribute set to somePicture.jpg . . . . .	9
2.7	A HTML b element with the inner text: Hello world . . . . .	9
2.8	A HTML div element with a img element as a child . . . . .	9
2.9	A HTML audio element with a song in two different codecs . . . . .	10
2.10	HTML5 example showing a document with 4 lines of text. . . . .	11
2.11	ASP.NET file example in C# writing a HTML showing 4 lines . . . . .	13
2.12	PHP example writing a HTML showing 4 lines . . . . .	14
2.13	JavaScript function that checks and sets which audio format should be played when it is called. . . . .	16
2.14	JavaScript function using JQuery for lookup of the div element ”ajaxSongDescription” and loading the HTML elements of the file description1.html, as its children, asynchronously using AJAX. . . . .	17
2.15	Applying a red background colour to the HTML with id ”ajaxSongDescription” . . . . .	18
2.16	”Hello World” in C#. . . . .	20
2.17	SQL code that retrieves the name and path of all the songs contained in the table SONGS. . . . .	20
2.18	LINQ C# code that retrieves the name and path of all the songs contained in the table SONGS. . . . .	21
2.19	Visual Studio 2010 debugging C# code. . . . .	23
2.20	Visual Studio 2010 SQL query tool. . . . .	23

2.21	Google Chrome developer tools debugging JavaScript Code. . . . .	24
3.1	MVC pattern diagram. . . . .	27
3.2	Users database table. . . . .	28
3.3	Songs database table. . . . .	29
3.4	Streams database table. . . . .	30
3.5	SongFolders database table. . . . .	30
3.6	SongStreams database table. . . . .	31
3.7	SongUsers database table. . . . .	32
3.8	StreamUsers database table. . . . .	32
3.9	Suggestions database table. . . . .	33
3.10	Negotiations database table. . . . .	34
3.11	Comments database table. . . . .	35
3.12	Contests database table. . . . .	35
3.13	StreamVotes database table. . . . .	36
3.14	User class. . . . .	38
3.15	Comment class. . . . .	39
3.16	Song class. . . . .	40
3.17	Stream class. . . . .	41
3.18	Folder class. . . . .	42
4.1	JQuery asynchronous POST request. . . . .	48
4.2	JQuery asynchronous POST request. . . . .	49
5.1	The main web-page of the web-site. This page is still to be filled by the customer. . . . .	51
5.2	User's Login web-page. . . . .	52
5.3	List of published songs in the system. . . . .	52
5.4	List of ongoing contests in the service. . . . .	53
5.5	List of the songs created by the user. . . . .	53

5.6	Page used to upload tracks and keep track of the song status. . . . .	54
5.7	Page to make suggestions to other people's songs. . . . .	54
5.8	Users permissions of a song. . . . .	55
5.9	Negotiation proposal. . . . .	55
5.10	Negotiation response. . . . .	56
5.11	Error page. . . . .	56





# 1 Introduction

## 1.1 Project Description

This dissertation, in the field of software engineering, was suggested by professor Martin Blom at Karlstad University, who has also been our customer for the project. Our assignment was to develop a web-site for creating music in a collaborative way on the Internet. Currently, there are not so many web-sites offering an easy way to create music together with other people. Therefore, this project presented here will try to fill the existing gap in this field.

The web-site will allow its users to upload music tracks, either to their own songs or to other users' songs. When a user uploads a track to other user's song it is called a suggestion. Once a suggestion is made there must be a negotiation between the owner of the song and the owner of the suggested track where they both agree on the price of the track. To encourage users to make suggestions to a specific song, the owner of the song can create a contest where he or she can choose the instrument of the suggestions.

## 1.2 Project Features

The web-site provides the user with a system capable of:

- Storing user's music tracks.
- Receiving or sending suggestions for other people's songs.
- Automatic mixing and encoding of the suggestions and the original tracks.
- Negotiating between the song's administrator and the person who makes the suggestion.

- Contesting between suggestions to allow the song's administrators to find the best suggestion for their music.
- The web-site was developed using HTML5 and offer interactivity with the users.

### **1.3 Chapter Overview**

The dissertation consists of the following chapters:

Chapter two contains a description of the project's background. This chapter present a comparison between our service and the existing services and also describes all the technologies used for developing this project.

Chapter three describes most of the design decisions made for the project, both for the database and for the application.

Chapter four discusses the most relevant implementation details for the project, such as the audio processing, the security aspect and the programming on the client side.

Chapter five offers a pre-view of the final user interface.

Chapter six gives a view of the project from retrospect, describing how the developing has been and what could have been better.

Chapter seven presents the conclusion and what needs to be done in the future.

## 2 Background

This chapter describes the project's background. It presents the existing systems and compare them to our system. It also describes the technologies used for developing the system and some of the tools used during the development.

### 2.1 Existing Systems

Nowadays, the world of music is changing and new ways to make and share music are developed; these ways allow people to share their creations more easily across the Internet. When we look on the Internet we find a large number of systems which allow us to upload and share our music with other people and compare it. Many of these systems work online (e.g. soundcloud [11], ubetoo [1], MixMatchMusic [12], jukeboxalive [10], blip [2]), while others use P2P technology (e.g. blubster [3] or bearshare [13]), but not all the systems we can find are exclusive for music. Usually, they are systems to upload, share and store any type of data.

In this section, we focus on two web-sites designed for music that have an objective and functionality similar to ours, Soundcloud and Ubetoo.

#### 2.1.1 Soundcloud

Soundcloud [11] is a web-site dedicated to creation and sharing of originally-created music. The users can easily upload or record sounds. The web-site also allow users to choose their privacy settings for each song, with many options between *private* (only the creator can see it) and *public* (everyone can see it).

The main goal of this web-site is to allow people to easily share music with other people and get feedback from their listeners. The web-site has designed a set of tools to facilitate the user to reach this objective. The *Time comments* system (see Figure 2.1) allow listeners to evaluate specific moments of each track. The web-site also give you statistics about your

tracks. The *groups* system allows the user to fastly share a song with a specific group of users. This web-site also has several applications for Iphone and Android (see Figure 2.2). Another advantage of this web-site is the possibility to create a *remix contest* for any of your songs. However, this feature requires that every component of the song is uploaded in a separate track.

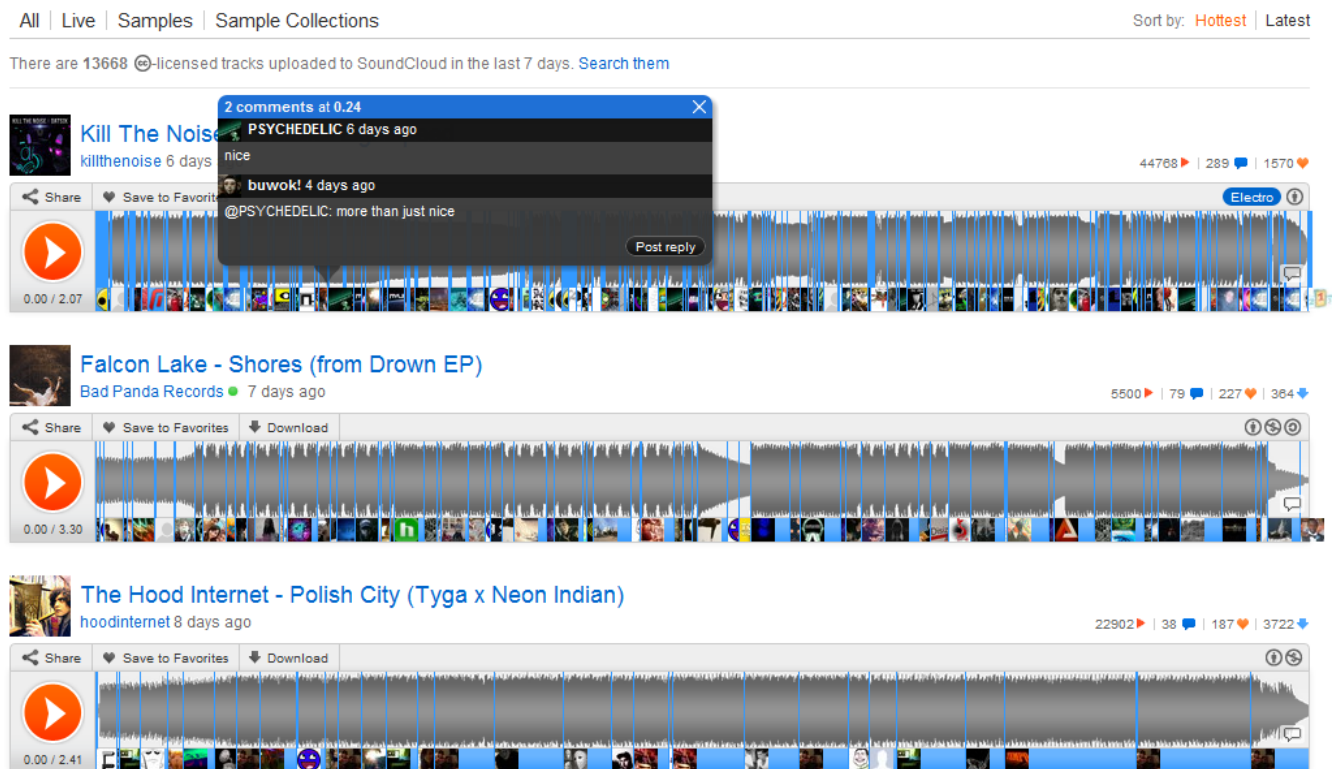


Figure 2.1: Time comments in a song from Soundcloud

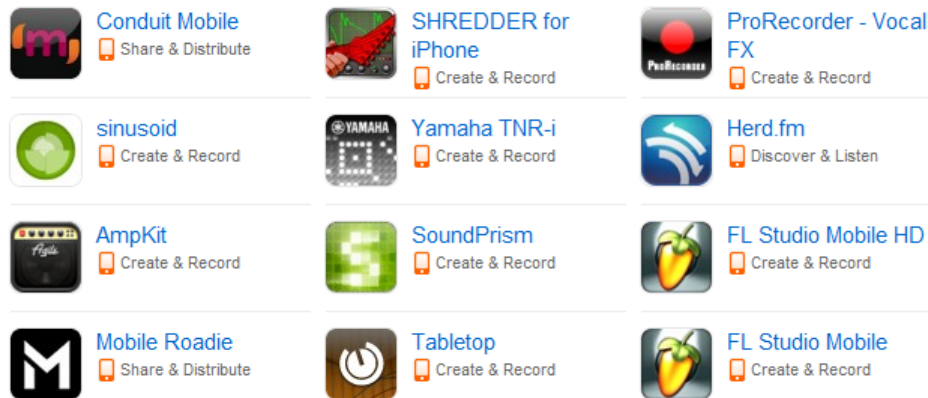


Figure 2.2: Soundcloud’s applications for mobile plataforms

One of the drawbacks of the web-site is that to get the full operation of the web-site the user has to buy the *pro account*. With the free account there is a limit in upload minutes, download per tracks, contacts list, statistics and some others tools of the web-site. When the user selects one of his or her tracks as not downloadable for the public, it is also forbidden for the user to download his or her own track. It is also not possible to create collaborative music.

### 2.1.2 Ubetoo

Ubetoo [1] is a web-site aimed for people who wants to make money from their music. For the users, it is free to share their creations and they earn money according to the number of views their tracks get thanks to advertisements. However, if users want to distribute their tracks to music stores, such as iTunes, Spotify, and Amazon MP3, they must pay. The web-site allow users to upload tracks and create albums and singles that are automatically accessible to everyone that is visiting the web-site (see Figure 2.3). One advantage of this web-site is that you get money every time that any of your tracks is played. Furthermore, you can get some feedback from the comments that other users have on your creations. The web-site is not exclusively for audio tracks but also for video tracks. These

are managed in the same way as the audio tracks (see Figure 2.4).

The screenshot shows the Ubetoo website interface. At the top, there is a search bar and navigation links for Notifications, Mailbox, and Stalkers. Below the navigation bar, there is a menu with links for Songs, Videos, Artists, Playlists, Store, and Forums. The main content area is titled "Most played songs on Ubetoo last 24 hours" and lists five songs with their respective album covers, titles, artists, categories, view counts, and publication dates. To the right of the list, there are sections for "All charts" (New uploads, Most played, Most played (last 24 hours), Top rated, Last pushed) and "Categories" (All, Acoustic, African, Afrobeat, Alternative, Ambient, Bachata, Balearic, Trance, Beats, Bluegrass, Blues, Dubstep, Easy listening, Electronica/Syr, Emo, Experimental, Folk, Folk Rock, Funk, Game, Go-Go, Gospel, Gothic Rock, Pop, Post Hardcore, Powerpop, Progressive, Psy Trance, Punk, Rap, Reggae, Reggaeton, Religious, Rnb).

Rank	Song Title	Artist	Category	Views	Published
1	Friday Night	Mirror of Myself	Experimental	15,819	12:35 AM Mar 20th 2010
	Formlessness	Mirror of Myself	Acoustic	2,654	12:50 AM May 3rd 2011
	Gaab : You're not Johnny Depp	lalouline	Rock	6,940	11:50 AM Dec 8th 2011
	Michel Clement : Suzie Bang	lalouline	Reggae	5,555	6:18 PM Dec 1st 2011
5	Lion's : Il fallait vivre sa vie	lalouline	Pop	6,132	11:56 AM Dec 1st 2011

Figure 2.3: List of most played songs on Ubetoo

The screenshot shows a video player on the Ubetoo website. The video is titled "Crazy Town (Venice Video) by Mirror of Myself on Ubetoo" and is by Mirror of Myself. The video player has a progress bar at the bottom showing 00:09. To the right of the video, there is a WIMP logo and a button that says "Get started right away". Below the video player, there is a green bar with various icons for Copy Link, Share, Embed, Add to Playlist, Add to Favorites, Push, Report, Tweet, Me gusta, and Share.

Figure 2.4: Video player on Ubetoo

One of the drawbacks of this web-site is that you cannot decide the privacy policy of your tracks. Also, if the users want to use the distribution function, their songs cannot be distributed by any other distributor at the same time. It is also not possible to create collaborative music, the tracks that the user upload must be finished creations to be played.

### **2.1.3 Comparison**

As we saw in the previous sections, there are some differences between the web-sites described above and our web-site.

The main function that makes our web-site different is the possibility of create collaborative music. That is, many users can participate in the creation of a song by adding a new track containing the new part of the song (a new instrument, a second voice, etc).The user also has the possibility to create a contest to find a specific component of his song, e.g. the voice or any instrument. In soundcloud, the user can create a remix contest, but the song must be finished to create the contest; Ubetoo do not allow users to create collaborative music. Ubetoo is only for sharing finished songs.

Another important point where our web-site differs from the others is the possibility of negotiating the price of a song between all participants.

## **2.2 Technology Review**

This subsection makes a brief overview of the technologies used in this project to provide the reader with enough information to understand further chapters.

### **2.2.1 HTML**

HTML (HyperText Markup Language) is a markup language designed for representing web-pages. A markup Language is a language that defines how a text shall be treated, processed or shown.

HTML is the main starting point when you learn about web development - all the web-pages that we can see on the Internet use some version of this language to encode their web-pages. A web browser (Client side) has to acquire the HTML code of the web-page and then interpret the code and most likely download some more content indicated in the acquired HTML file and execute their JavaScript scripts if any. We will take a look at JavaScript in section 2.2.4.

In order to retrieve a HTML file, the web browser communicates with the web server (Server side) by using the HTTP protocol.

HTTP (HyperText Transfer Protocol) defines how the communication between the web client and the web server has to be accomplished. Among a few other aspects, this protocol basically establishes the request methods available to the web client and also the return codes that the web server may use as response.

The most important request methods are "Get" and "Post". The "Get" method allows the web client to ask the web server for a resource - this resource is most likely HTML code of a web-page, but it can actually be data of any type. The address of the resource desired is specified by a URI (Uniform Resource Identifier).

The "Post" method is used to make a resource request but also to send data to the web server. It is usually used for submitting a web form and for uploading files to the web server.

Once a HTML file has been received by using the "Get" or "Post" HTTP method, it should be processed. In order to find out how this is done, let us take a look at the contents of a regular HTML file.

A HTML file is composed of a set of elements called "Tags". There are two ways to represent tags; the shorter way is where the tags have their name enclosed by the symbols < and \ >, the longer way is where the tags follow the format <TagName> <TagName\ >.



see Figure 2.5.

```
<div \>  
<div> <\div>
```

Figure 2.5: A HTML div element written in short and long forms

Each HTML element represents an object – such as a text box, a label, a menu, a list, etc. – which the web browser has to understand and show if necessary. The complete list of HTML elements version 5 of the current draft can be consulted on the w3c web-page [15].

Depending on the HTML element, it can have different attributes or inner text or inner html code, see Figure 2.6, 2.7 and 2.8 respectively.

```

```

Figure 2.6: A HTML img element with the src attribute set to somePicture.jpg

```
<b> Hello world <\b>
```

Figure 2.7: A HTML b element with the inner text: Hello world

```
<div>  
    
<\div>
```

Figure 2.8: A HTML div element with a img element as a child

Element attributes set the properties of the object – such as width, height, destination address of a link, which script should be called when a certain event occurs, style preferences (See section 2.2.7), etc.

Inner HTML code allows for establishing a hierarchy between the HTML objects. Thus, in

each HTML document, there will be parents, children and siblings. Inner HTML code is useful, for instance, for making tables –where you have a table object as parent and rows as children–, and for making lists and group boxes.

### 2.2.2 HTML5

The fifth version of HTML is currently a draft, thus its specification may change, and probably will, when the final version get released. According to that, some parts of this section may not be valid in the future.

This new version of the HTML standard aims to make the use of plugins unnecessary – external software installed in the web browser – for multimedia purposes. This is accomplished by a set of new HTML elements which handles audio and video playback, see Figure 2.9 for an example.

```
<audio id="audio1" autobuffer controls>
  <source src="song.mp3">
  <source src="song.ogg">
</audio>
```

Figure 2.9: A HTML audio element with a song in two different codecs

As can be seen in Figure 2.9, several sources of the same song in different codecs have to be provided. The reason for this is that the standard does not declare a unique format and codec for audio and video. This means that each web browser is only compatible with the formats and codecs that they had decided to implement. This is not a problem by itself, the problem is that there are several popular codecs and formats that are proprietary and these cannot be implemented on open source web browsers like Mozilla Firefox.

Currently there is an ongoing battle between web browsers that implement proprietary

codecs, web browsers that implements free codecs and web browsers that implements both. At the end of the day, this battle is translated into a problem for everyone: It is a problem for web developers, who once again have to check for compatibility with all web browsers. It is also a problem for the server side that has to store several copies of the same resource in different codecs. It also may be a problem for the final user who may be forced to choose the web browser that is compatible with the web-sites that he or she uses, like in the old days before HTML 5.

At first, HTML 5 also offered several other features such as local storage, web sockets, WebGL, geolocation, etc. but these features now have their own specification outside of the HTML 5 specification.

For our project, we have only used the new audio element and some other minor new elements and attributes such as sections, placeholders, header, nav and progress. A basic HTML5 page is shown in Figure 2.10.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Webpage 1</title>
  </head>
  <body>
    line #1
    line #2
    line #3
    line #4
  </body>
</html>
```

Figure 2.10: HTML5 example showing a document with 4 lines of text.

### 2.2.3 Dynamic Web Pages

We have now discussed development of static web pages using HTML. The next question is: How can you create pages that show dynamic content? In the most typical cases you would like to link some content from a server database - to upload user's information or data, for example.

For this purpose, at the server side, we will need to generate the HTML code on the fly when the client requests the web page.

There are several platforms and programming languages designed to make this job. The process that all of them follow is to execute a piece of code at the server which outputs HTML code that will be sent back to the client.[9]

The two most common programming languages for this purpose are ASP.NET and PHP, which are described below.

#### **ASP.NET**

ASP.NET is a platform built on top of the Microsoft .NET Framework.

The .NET Framework is a large set of libraries and high-level programming languages.

The compiled code for the framework is known as Common Intermediate Language (CIL) - also called managed code because it is executed within the framework's environment. Since the code is not compiled to machine code, it needs a virtual machine which reads the CIL code and transforms it into the assembler language of the processor where the application is executed. The framework also gives the possibility of making calls to unmanaged libraries - code that runs outside the framework's environment in assembler code, which allows low-level programming as well.

Since the CIL is not related to any platform, processor architecture or operating system, a .NET assembly can be executed on any processor or operating system if there is a virtual

machine for it.

In ASP.NET you define what is to be executed when a HTTP request method ("get" and "post" usually) is received. By using a programming language supported by the .NET Framework the code will create the HTML page that should be sent to the client.

The ASP.NET files contain the regular HTML code that is given back to the client, but they also contain code written in the chosen programming language. In order to state what HTML is and what belongs to the programming language, special ASP.NET directives are used. The most common directives are `<% %>` and `<% : %>`. The former executes the inner code and the latter prints out to HTML the value of the inner variable.

Figure 2.11 shows how ASP.NET can be applied to generate a document like the one shown in Figure 2.10.

```
<%@ Page Language="C#" %>
<!DOCTYPE html>
<html>
  <head runat="server">
    <title>Webpage 1</title>
  </head>
  <body>
    <% for (int i = 1; i <= 4; i++) { %>
      <p>
        <%: "line #" + i %>
      </p>
    <% } %>
  </body>
</html>
```

Figure 2.11: ASP.NET file example in C# writing a HTML showing 4 lines

## PHP

PHP is a programming language which has quite a large number of libraries the programmer can use.

Like the .NET Framework, PHP was not meant to be compiled to machine code although there are some compilers available which usually can speed up the execution of PHP.

The execution of native code is a little more complicated than in the .Net framework. To do this execution, a new PHP extension in C has to be written, then compiled and inserted in the PHP server, adding more complexity to the process of developing a wrapper.

As can be seen in Figure 2.12, the idea behind PHP is really the same as behind ASP.NET - a HTML document with embedded code of a programming language.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Webpage 1</title>
  </head>
  <body>
    <? for ($i = 1; $i <= 4; $i++) { ?>
      <p>
        <?= 'line #' + $i ?>
      </p>
    <? } ?>
  </body>
</html>
```

Figure 2.12: PHP example writing a HTML showing 4 lines

## 2.2.4 JavaScript

Now we understand the technology behind a static web-page and how to generate dynamic web-pages. However, there is still one last important issue to be addressed and it is the interactivity with the user.

In some scenarios it is desirable to execute some actions at certain events, such as a click on some button, a mouse pointer over some panel, or a periodic task. There are only two choices, either executing a script at the server side or executing it at the client side. If it is executed at the server side, as discussed in section 2.2.3, the script will encounter high delays for interactive use since the server is most likely to be on a remote machine.

To solve this problem, those actions that are meant for an interactive use have to be executed on the client side.

The programming language designed for this purpose, without the need of installing a plugin in the web browser, is JavaScript.

The computer resources that can be used by JavaScript is strongly limited, since security issues may arise otherwise. Therefore, the code that is given to the web browser by the server is not compiled. Instead, it has to be interpreted in the web browser – for security and compatibility reasons. Thus, one of the disadvantages of this limitation is also that JavaScript code executed at the client side is exposed to the client and can be copied and tampered with. We need to be aware of not compromising the server's security since that code will be public.

JavaScript is based on C, but it does not have pointers and it is weakly typed. The language is designed for managing the elements contained in the HTML code and its values and attributes. This is accomplished by the Document Object Model (DOM). Each time a web browser loads a HTML document it generates a tree of objects, where each HTML element is contained along its attributes, values and element hierarchy, which is

accessible through the Document Object Model API. Any change made to those objects will be reflected on the actual web-page.

By adding, deleting or editing the objects along some others built-in functions, we can provide the client with a real time interactive web-page.

We can see how to search for an element with the attribute id set to "audioPlayer" by using the Document Object Model in JavaScript in Figure 2.13.

```
<script type="text/javascript">
  function changeSong(songPath) {
    audioElement = document.getElementById('audioPlayer');

    if (audioElement.canPlayType('audio/mpeg')) {
      audioElement.src = songPath + '.mp3';
    } else if (audioElement.canPlayType('audio/ogg; codecs="vorbis"'))
      audioElement.src = songPath + '.ogg';
    } else {
      //FALLBACK
    }

    return false;
  }
</script>
```

Figure 2.13: JavaScript function that checks and sets which audio format should be played when it is called.

### 2.2.5 AJAX

AJAX (Asynchronous JavaScript And XML) is a technique that allows a web browser to retrieve data from the web server to process through JavaScript. The communication is achieved by a new JavaScript object named XMLHttpRequest. This object controls the ingoing and outgoing traffic through http or https – others protocols may be implemented but the W3C's specification leaves this uncovered. The object provides us with methods and attributes to make requests and to listen to responses. The type of data that can be



sent and received can be of any data type, not only XML, as may be inferred by its name. However, the most usual use is to send and retrieve HTML text code or XML and for that reason the object is prepared with special methods for those formats.

### 2.2.6 JQuery

JQuery is a JavaScript library which makes an abstraction to several new technologies and techniques, such as AJAX, event handling and animation. This library removes the underlying details, such as the communication protocols and file formats used. Therefore, in this project wherever the AJAX technique is used it will be through JQuery, since the methods provided by the XMLHttpRequest object are low-level and would slow down the development of the web-site.

Figure 2.14 shows an example of AJAX using JQuery.

```
<script src="jquery - 1.5.1.js" type="text/javascript"></script>
<div id="ajaxSongDescription" \>
<script type="text/javascript">
    function loadDescription() {
        $ $("#ajaxSongDescription").load("description1.html");
    }
</script>
```

Figure 2.14: JavaScript function using JQuery for lookup of the div element "ajaxSongDescription" and loading the HTML elements of the file description1.html, as its children, asynchronously using AJAX.

### 2.2.7 CSS

Whilst HTML was growing up in its early days, developers realised that if the HTML files contains the attributes for visual presentation and style per each HTML element, the

HTML files become really complex and code is repeated in most of the elements. In order to address this situation, the Cascading Style Sheets (CSS) language was created. Therefore, the presentation and the content of a web-page was split into a CSS and a HTML file respectively. Nevertheless, the presentation information can still be written in the HTML file – the use of CSS is not mandatory.

A CSS file holds only presentation and style information. It contains a set of declarations with the name or type name and class of the elements to apply the style (Selectors) and its style, see Figure 2.15.

The language is called *Cascading* Style Sheets because several declarations can match a single element from multiples sources. The algorithm that solves this problem chooses the style with the most specific selector. For instance, the selector "#ajaxSongDescription" is more specific than "div", since "div" is an element type and "#ajaxSongDescription" is a name id. Thus, the style declared by the selector "#ajaxSongDescription" will be applied.

————HTML CODE————

```
<div id=" ajaxSongDescription" \>
```

————CSS CODE————

```
#ajaxSongDescription  
{  
    background-color : red ;  
}
```

Figure 2.15: Applying a red background colour to the HTML with id "ajaxSongDescription".

### 2.2.8 C#

C# is a general purpose, object oriented programming language developed by Microsoft and has become an European Computer Manufacturers Association (ECMA) and International Organization for Standardisation (ISO) standard. C# is used in this project on the server side along with ASP.NET.

C# is one of the programming languages supported by the .NET Framework described in section 2.2.3.

There is an ongoing free and open source project named Mono, which aims to create a platform compatible with the .NET Framework platform. Currently, Mono supports almost the entire .NET Framework version 4.0 (released on mid 2010) but it lacks the support of the Entity Framework, which is used in our project and will be described in section 2.2.9. When the Entity Framework gets implemented in the Mono project, it will be an option to be taken into account by the customer, since deploying this web-site on a web-server by using the Mono platform should be licence-free.

C# has a similar syntax as C, C++ and Java. It also has support for pointers, but they should not be used whenever it can be skipped, since the use may destroy the compatibility with different system architectures, because of different pointer lengths, word endianness, etc.

Figure 2.16 shows the "hello world" program written in C#.

### 2.2.9 SQL

Structured Query Language (SQL) is a language used to make queries to database systems. SQL allows to retrieve, insert, modify and delete data from a certain database. SQL is powerful enough to retrieve exactly the data that you want to, avoiding the need of post-processing the data with any other programming language to filter the results.

```

class Program
{
    public static void Main()
    {
        System.Console.WriteLine("Hello _world" );
    }
}

```

Figure 2.16: "Hello World" in C#.

The information in the database is stored in tables, where each table has its own fields. There can exist relations between fields from one table to another. That is the basic concept behind a SQL-database. A basic SQL-query is shown in Figure 2.17.

```

SELECT name, path
FROM SONGS;

```

Figure 2.17: SQL code that retrieves the name and path of all the songs contained in the table SONGS.

In this project we are not using the SQL language directly. Instead, we are using the Entity Framework, linked to a SQL Server which provides us with the database system. The Entity Framework maps a database to objects in the .NET Framework. With this approach, we do not longer depend on the database system and language, but on the model provided by the Entity Framework. As long as the database fits the model done by ourself or the model generated automatically by the Entity Framework, and the Entity Framework is compatible with the database system, the database system can be replaced with another system.

The technology used to query the database through the Entity Framework is Language-Integrated Query (LINQ). LINQ extends the capability of C#, bringing us new language syntax to query databases in a very similar syntax as SQL has – this is why it is also very

important to know how SQL works. The Entity Framework will take care of the translation between LINQ and SQL. We can compare how LINQ differs from SQL (see Figure 2.17) by looking at Figure 2.18. Here we can see the two major advantages; the result is in C# objects directly and the database system is abstracted, while the code does not really differ from SQL – LINQ is as powerful as SQL.

```
musicsharedbEntities context = new musicsharedbEntities ();  
  
var queryResult = from song in context.Songs  
                  select new { song.Name, song.Path };
```

Figure 2.18: LINQ C# code that retrieves the name and path of all the songs contained in the table SONGS.

## 2.3 Tools

In this section we show the general features of the main tools used in the project. We do not review all of the tools that have ever been used in this project due to the large amount of technologies and API's which this project relays on, as it has been stated in the Technology Review section. Therefore, we describe only the two most important tools, Visual Studio and the web browser Google Chrome, which cover the development and debugging of the technologies used at the server side and at the client side, respectively.

### 2.3.1 Visual Studio

Visual Studio 2010 is an IDE (Integrated Development Environment) made for working with several general-purpose programming languages for the Windows platform, such as ASP.NET and C#. Visual Studio 2010 also has tools for managing SQL server and therefore for making SQL queries. These features makes Visual Studio 2010 perfectly suitable for developing the server side of our project since it provides us with high quality tools to work with and for debugging the languages that our project requires.

Visual Studio 2010 helps us with code highlighting, code auto-completion and real-time compiling – it compiles our code as we write it, showing up syntax errors earlier – but the greatest and most helpful feature that an IDE can have are the debugging tools; hence, we are going to show them in the upcoming figures.

Figure 2.19 shows the use of Visual Studio 2010 for debugging C# code of the web-site and Figure 2.20 shows the SQL query tool that allows creating and debugging databases easily.

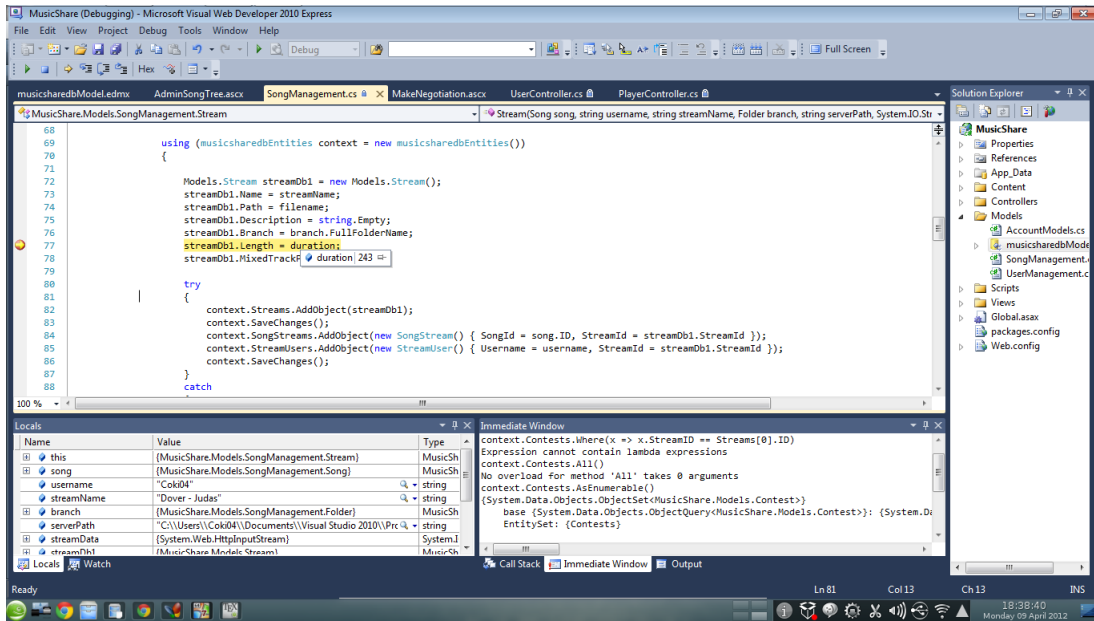


Figure 2.19: Visual Studio 2010 debugging C# code.

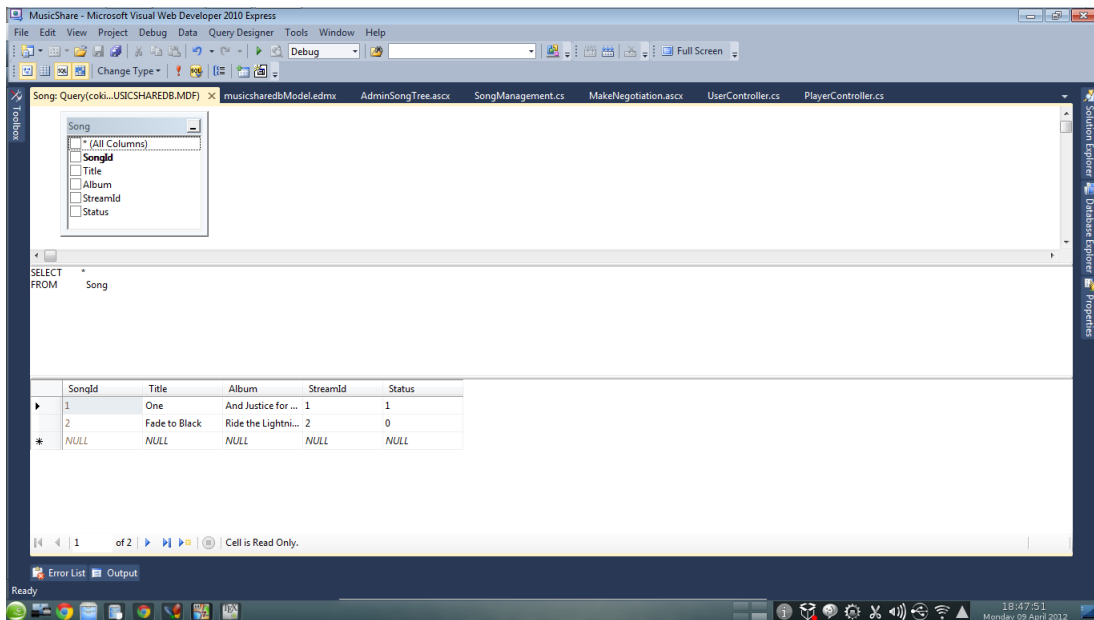


Figure 2.20: Visual Studio 2010 SQL query tool.

### 2.3.2 Web browser

We have chosen Google Chrome as web browser, not only because it is the only one which currently supports all the HTML5 features that we use but also because we find its developer tools very helpful.

Google Chrome developer tools offer the possibility of setting breakpoints in the JavaScript code pausing its execution when the breakpoint is hit, allowing you to analyse the values of whichever variable you want to, or to see what flow the script follows. This functionality gets more important as the script grows since debugging big scripts without such tools becomes a really tedious work. These tools also provide us with a network sniffer which shows the HTTP messages exchanged between the client and the server. This sniffer became quite useful for debugging our AJAX messages, made us able to inspect the actual sent and received data and it also made us able to locate where the errors were taking place.

Figure 2.21 shows the Google Chrome developer tools.

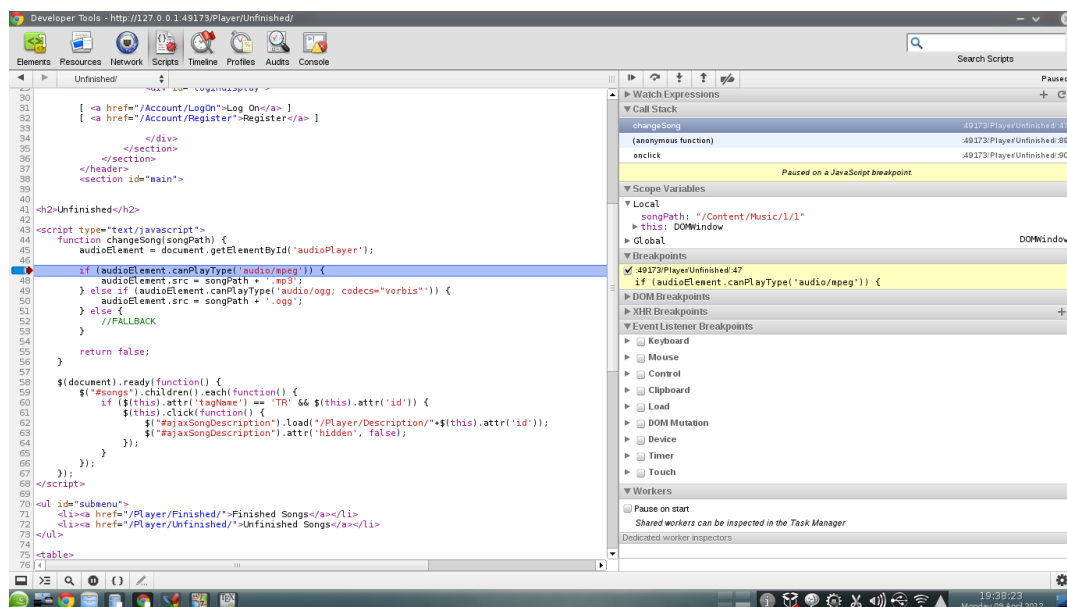


Figure 2.21: Google Chrome developer tools debugging JavaScript Code.



## 2.4 Chapter summary

The objective of this chapter has been to provide the reader with enough knowledge about what this project is based on. In order to do so, in section 2.1 we have presented a comparison between our web-site and others web-sites which have a similar function as ours. As a result, we found out that although services for music sharing are quite developed, services for creating music in a collaborative way, like in our project, have been not.

Section 2.2 gives a brief introduction to the technologies used in this project. It is intended to make the reader able to understand the design and implementation chapters.

Finally, in section 2.3 we give an overview of the main tools used for developing the server and client side code.

## 3 Design of the system

This chapter discusses the design decisions that have been taken for the program and for the database design. First, however, it is necessary to present the features that we were required to implement, since the design had to be done according to them.

The whole project follows the Model-View-Controller pattern. Hence, we will also explain what this architectural pattern is, requires and offers.

### 3.1 Requirements

The goal of the project was to offer a web-site where the users can upload their own music tracks, allowing other users to make suggestions for completing a song in exchange of money, a share of the profit made by the song, or just for fun. Therefore, since money is involved in the system, we were also required to take a close look of the web-site security and personal data confidentiality.

A negotiation system should also be built – users must be able to debate what they can offer and what they want in exchange for their tracks. Suggested tracks shall not be used if the song’s administrators and the person who has uploaded the suggestion have not reached an agreement.

The system has to keep track of which suggestion belongs to which track, forming a track tree. The tree is not restricted to suggestions; the song’s administrator could also use this tree to upload each instrument played in the song independently. Thus, users can focus on the instruments they want to make suggestions of.

The system also has to mix all the tracks and their suggestions, allowing the user to listen to each track independently or to listen to the whole set of tracks.

In addition to the suggestion system, the system should provide the users with contests, where the song’s administrators will set a prize that will be won by the user with the best suggestion, chosen out of public votes or by the song’s administrators.

## 3.2 Model-View-Controller

Model-View-Controller (MVC) is a software architectural pattern which separates a program into several components. As we can see in Mehran Nikoo's [14] diagram of MVC (see Figure 3.1), there are three components: the Model, the View and the Controller.

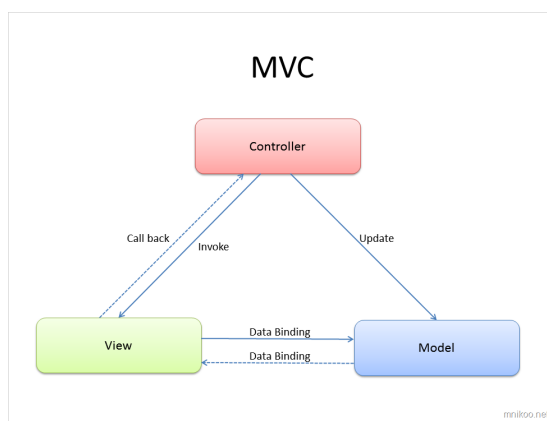


Figure 3.1: MVC pattern diagram.

The Model takes care of the storage of the data in the system and the Model is also the component that knows how to work with the data.

The View is the representation of the system which is given to the final user. It is the component that makes it possible to work with the system. The View gets its data from the Model which is generated from the controller.

The Controller is the component that initiates each action taken in the system. Each event made in the View is forwarded to the Controller. In the Controller the event is processed and it takes all the needed data from the Model. The Controller also sends all the required messages to update the Model if necessary.

We decided to use this pattern because we thought it would be a great advantage to have these components separated. Not only because everything gets more structured but also because a single change in one of the components no longer affects the whole project.

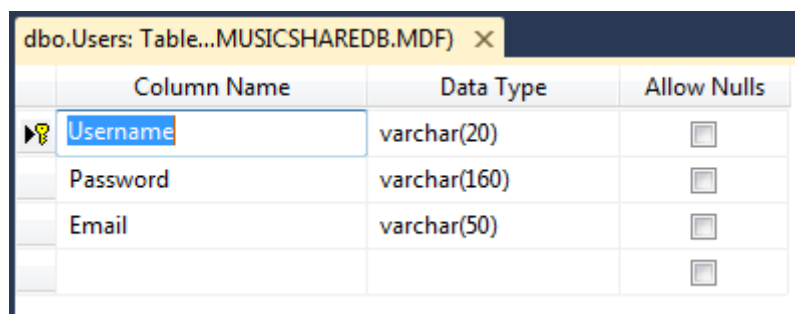
### 3.3 Database Design

In this section, we describe which tables and fields were needed in the database and why we decided to create each and one of them. Our database is bound entirely to the Model component of the Model-View-Controller pattern. Each one of the following subsections corresponds to a requirement of the project.

The whole Database diagram can be seen in Appendix A.

#### 3.3.1 Table Users

The system will manage user data since each song is bound to a specific user and users will also be able to negotiate songs between each other. Therefore, it becomes clear that we need to store some user information and we also need to authenticate them in our system by using some password, since their information and identity must remain safe.

The image shows a screenshot of a SQL Server Enterprise Manager window displaying the table definition for 'dbo.Users'. The window title is 'dbo.Users: Table...MUSICSHAREDB.MDF'. The table has four columns: 'Column Name', 'Data Type', and 'Allow Nulls'. The columns are: 'Username' (varchar(20)), 'Password' (varchar(160)), 'Email' (varchar(50)), and an empty column with 'Allow Nulls' checked. The 'Username' column is highlighted with a blue selection box.

Column Name	Data Type	Allow Nulls
Username	varchar(20)	<input type="checkbox"/>
Password	varchar(160)	<input type="checkbox"/>
Email	varchar(50)	<input type="checkbox"/>
		<input checked="" type="checkbox"/>

Figure 3.2: Users database table.

In Figure 3.2 the table Users and its fields are shown. The table contains the Username as the primary key of the table since the Username will work as the identifier for the user and we do not allow two users or more with the same Username. Furthermore, the table stores the password that is used for authentication and the email which may be used for contacting the user.

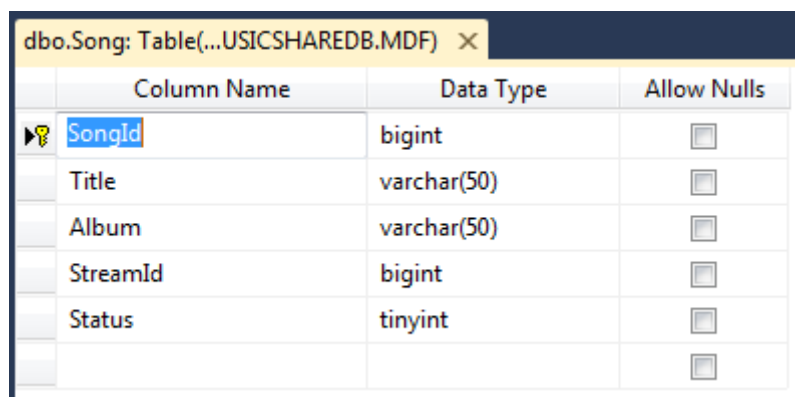
Additional user data may be required in the future. In that case, it should be easy to add

the data to the database if needed.

### 3.3.2 Table Songs

Songs are at the core of the database design and involves several tables that we discuss below.

The first table we discuss is the Song table, which can be seen in Figure 3.3.



Column Name	Data Type	Allow Nulls
SongId	bigint	<input type="checkbox"/>
Title	varchar(50)	<input type="checkbox"/>
Album	varchar(50)	<input type="checkbox"/>
StreamId	bigint	<input type="checkbox"/>
Status	tinyint	<input type="checkbox"/>
		<input type="checkbox"/>

Figure 3.3: Songs database table.

We use SongId as the primary key for each song, since we want to allow several songs with the same title or album. Therefore, SongId is an auto-incremented number that identifies each song.

The Title and Album fields are just strings for naming each song. Status describes whether the song is open to suggestions, in negotiation or if it is published.

StreamId links the song to a stream database table which describes its main track, that is, the final finished song track, see Figure 3.4.

Column Name	Data Type	Allow Nulls
StreamId	bigint	<input type="checkbox"/>
Name	varchar(50)	<input type="checkbox"/>
Path	varchar(255)	<input type="checkbox"/>
MixedTrackPath	varchar(255)	<input type="checkbox"/>
Length	int	<input type="checkbox"/>
Description	varchar(500)	<input checked="" type="checkbox"/>
Branch	varchar(255)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Figure 3.4: Streams database table.

Since each song can have several tracks, such as suggestions and different instruments, for composing of the final track to get released, we introduce the Stream table. Each Stream represents an audio file. Therefore, the field Path and MixedTrackPath stores the physical path to the audio file and to the track mixed with all the previous tracks in its branch of the tree. We also include the branch name where the stream belongs, which is essential for, later on, extracting the hierarchy of the Streams that compose the song. The Length field stores the track duration in seconds, which is shown to the final user on the web-site. The tree-structure of the song is stored in the table SongFolder, see Figure 3.5

Column Name	Data Type	Allow Nulls
SongID	bigint	<input type="checkbox"/>
FolderName	varchar(255)	<input type="checkbox"/>
		<input type="checkbox"/>

Figure 3.5: SongFolders database table.

This table relates the songs with their folder which builds the Stream tree. The field FolderName contains the name of the folder following the standard Unix file systems path naming policy, that is, /folder1/folder2/folderN/file1. It is important to realise that these paths describes logical paths – the song structure– they do not describe the physical path of the audio files.

We also needed a connection between Songs and Stream. There is already one link between them in the field StreamId of the Table Song, which represent the final composed song track, but it is still needed to connect all the Streams that compose the song with the song itself. For this purpose, the table SongStreams was created, see Figure 3.6.

Column Name	Data Type	Allow Nulls
SongId	bigint	<input type="checkbox"/>
StreamId	bigint	<input type="checkbox"/>
		<input type="checkbox"/>

Figure 3.6: SongStreams database table.

This SongStreams table is just an ordinary relational table, which link songs and streams with each other.

There is still one last piece of information to be stored. Right now, we could load all the songs with all their Stream hierarchy and be able to find all the audio files, but we have not established relations with their owners and their permissions. The followings tables solve this issue.

Column Name	Data Type	Allow Nulls
UserName	varchar(50)	<input type="checkbox"/>
SongId	bigint	<input type="checkbox"/>
Permissions	tinyint	<input type="checkbox"/>

Figure 3.7: SongUsers database table.

Column Name	Data Type	Allow Nulls
StreamId	bigint	<input type="checkbox"/>
Username	varchar(50)	<input type="checkbox"/>

Figure 3.8: StreamUsers database table.

The SongUsers table shown in Figure 3.7 makes a connection between songs and users and also adds a permissions field with indicates whether the user has administrator rights or not.

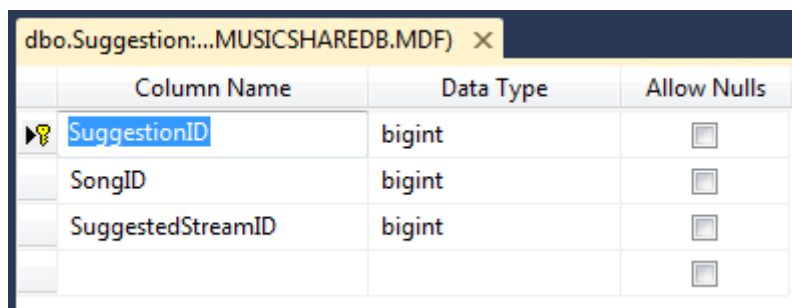
The StreamUsers table in Figure 3.8 connects streams and users. The permissions field is not needed here because the song’s owners already have all the rights , otherwise the song’s administrator would not be able to publish their song.

### 3.3.3 Table Suggestions

The suggestions table that can be seen below in Figure 3.9, is a simple table. A new stream is used as a suggestion, therefore we just need to link a suggestion with the stream and with the song. If a stream is included in any suggestion, we will treat that stream



as a suggestion. By doing so, rather than making a new big suggestion object with all the information that a Stream has, we are making use of the Stream table, which already exists and we are not complicating the final design.



Column Name	Data Type	Allow Nulls
SuggestionID	bigint	<input type="checkbox"/>
SongID	bigint	<input type="checkbox"/>
SuggestedStreamID	bigint	<input type="checkbox"/>
		<input type="checkbox"/>

Figure 3.9: Suggestions database table.

### 3.3.4 Table Negotiations

The negotiation process is based on a special type of messages which are exchanged between the negotiation parties. The messages states whether they want to give or get the music track for free, for an amount of money or for a share of the song’s profit. The message type is indicated in the field Type of the table Negotiations, see Figure 3.10. The field Value store the amount of money or the percentage of song’s profit, if needed.

Who has created the negotiation, when he or she has created it, who is the receiver and who is the sender, is stored in the fields Creator, Date, ToUser and FromUser, respectively. The negotiation is also linked to a suggestion which is what is being negotiated in the field SuggestionId and it is also linked to a comment object that the sender may leave. The field ResponseTo is a link to another negotiation object. This field makes it possible to renegotiate and to make a negotiation history.

Finally, the Status field tells us if the negotiation has been rejected, accepted or renegotiated.

Column Name	Data Type	Allow Nulls
NegotiationID	bigint	<input type="checkbox"/>
Type	tinyint	<input type="checkbox"/>
Value	money	<input checked="" type="checkbox"/>
CommentID	bigint	<input checked="" type="checkbox"/>
Status	tinyint	<input type="checkbox"/>
FromUser	varchar(20)	<input checked="" type="checkbox"/>
ToUser	varchar(20)	<input checked="" type="checkbox"/>
SuggestionID	bigint	<input type="checkbox"/>
Date	date	<input type="checkbox"/>
ResponseTo	bigint	<input checked="" type="checkbox"/>
Creator	varchar(20)	<input type="checkbox"/>
		<input type="checkbox"/>

Figure 3.10: Negotiations database table.

The Comments table as shown in Figure 3.11 is used to send messages between users or to link a message to another object, like was the case in the Negotiation object, where the user who sends the offer or the counteroffer can leave a message by using a Comment object.

Column Name	Data Type	Allow Nulls
CommentID	bigint	<input type="checkbox"/>
UserName	varchar(20)	<input type="checkbox"/>
Comment	varchar(2048)	<input type="checkbox"/>
ResponseTo	bigint	<input checked="" type="checkbox"/>
Date	date	<input type="checkbox"/>
		<input type="checkbox"/>

Figure 3.11: Comments database table.

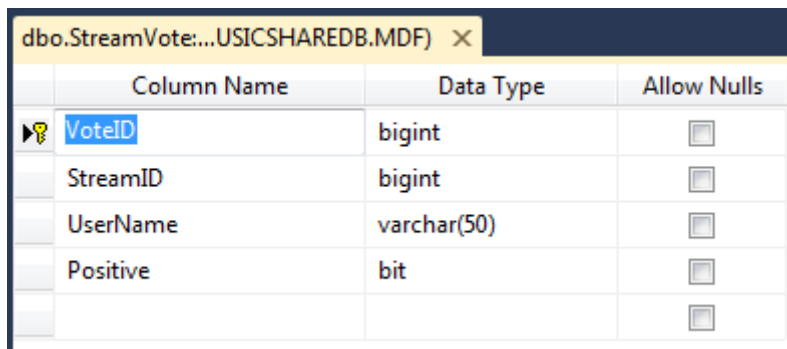
### 3.3.5 Table Contests

Column Name	Data Type	Allow Nulls
ContestID	bigint	<input type="checkbox"/>
StreamID	bigint	<input type="checkbox"/>
Type	tinyint	<input type="checkbox"/>
Status	tinyint	<input type="checkbox"/>
CreationDate	date	<input type="checkbox"/>
StartDate	date	<input type="checkbox"/>
EndDate	date	<input type="checkbox"/>
Prize	money	<input type="checkbox"/>
Description	varchar(255)	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Figure 3.12: Contests database table.

The Contests table (see Figure 3.12), has the fields Status, Type and Prize, which are analogous to the Status, Type and Value fields in the Negotiations table (see Figure 3.10).

CreationDate, StartDate and EndDate are used to open and close the contest when the creator specifies. StreamId is a link to a Stream upon which the contest is made.



The screenshot shows a table named 'StreamVotes' in a database. The table has five columns: VoteID (bigint), StreamID (bigint), UserName (varchar(50)), Positive (bit), and an unnamed column (bit). All columns are marked as 'Allow Nulls'.

Column Name	Data Type	Allow Nulls
VoteID	bigint	<input type="checkbox"/>
StreamID	bigint	<input type="checkbox"/>
UserName	varchar(50)	<input type="checkbox"/>
Positive	bit	<input type="checkbox"/>
		<input type="checkbox"/>

Figure 3.13: StreamVotes database table.

The StreamVotes table shown in Figure 3.13 is made for keeping track of the votes that the public have made to a certain stream. This table is used later on to decide who is the winner of a certain contest when the administrator has chosen the contest to be decided by the public.

### 3.4 Program Design

Since we followed the MVC pattern commented before in this section, the design of the program is, of course, ruled by it. The main responsibility relays on the Model component, since the Model is the component in charge of the data and its management – this was also the biggest and hardest part of the project.

Therefore, this sub-section is focused on the Model component of the code, although a few details of the Controller and View components are also discussed. A class diagram of the Model is shown in Appendix B.

The big picture of the whole system is shown in Appendix C.

### 3.4.1 Code - Database Interaction

One of the biggest design decisions made for the model was how to make the interaction between the database and the code. This interaction is of utmost importance since the simplicity of the code for the rest of the project depends on how this interaction is designed. Furthermore, by offering simplicity you usually offer an easy-to-maintain code.

Since we wanted to implement a very simple management of the database, we came with the idea of not making the users aware of that they are working with a database at all. Therefore, we have not explicitly implemented any "save to database", "load from database" or "update the database with" method, in any of the classes of the Model component. Instead, we let the language semantic to do the work for us.

The constructors of each class of the Model Component take care of – besides creating a new object in local memory – storing of the new object in the database and, if necessary, the constructor also stores files into the file system (usually the different audio files that each song uses).

When you want to load an object into memory that has been previously created, the class that load the object implements a FromID static method which loads the object into memory. This method will also populate the new object with data taken from the database.

To delete an object, the class that needs to do this also implements a delete method. The Delete method will delete the information stored in the database and also the files in the file system if any. It will not delete the object from memory since the .Net platform uses a garbage collector, which keeps track of every .Net object and decide when to remove the object. A deleted object should no longer been used – calling its method can trigger an exception.

Other methods in the class implements searching into the database if needed, but the classes outside does not have knowledge of this. Searching is transparent to the upper classes.

As can be realised now, this design offers an abstraction layer to the users of the class since

they do not know that they are actually working with an underlying database. The design provides simplicity since you do not longer have to be worried about managing a database at a certain layer.

### 3.4.2 Model - UserManagement

The UserManagement class handles the information about the users. Whenever a user is registered on the web-site, the controller accesses this class instancing the User class contained inside the UserManagement class. The User class' constructor will validate all his or her information and store it in the database. Inside the UserManagement class we find two inner classes called User and Comment.

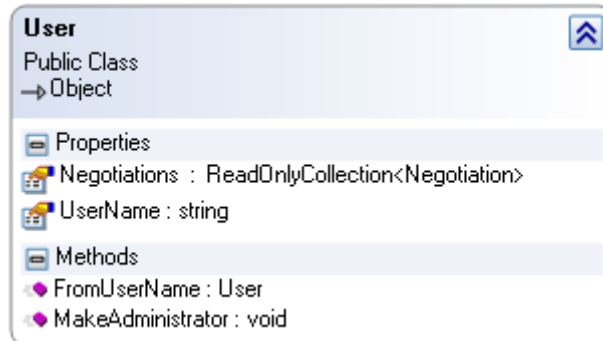


Figure 3.14: User class.

The user class, as seen in Figure 3.14, includes the methods to authorize the user in the system, to check their permissions and to access the negotiations which they are taking part of.



Figure 3.15: Comment class.

On the other hand, the inner class Comment, see Figure 3.15, holds the fields that are stored in the database and its methods allow the creation of a new comment and also to respond to a previous comment.

### 3.4.3 Model - SongManagement

The SongManagement class is composed of all the classes which implement concepts related to the songs. Most of the classes have a clear link to the database tables. Thus, we will not discuss those classes' properties that have been already discussed in the Database sub-section. Instead, we will discuss the methods of the classes mostly.

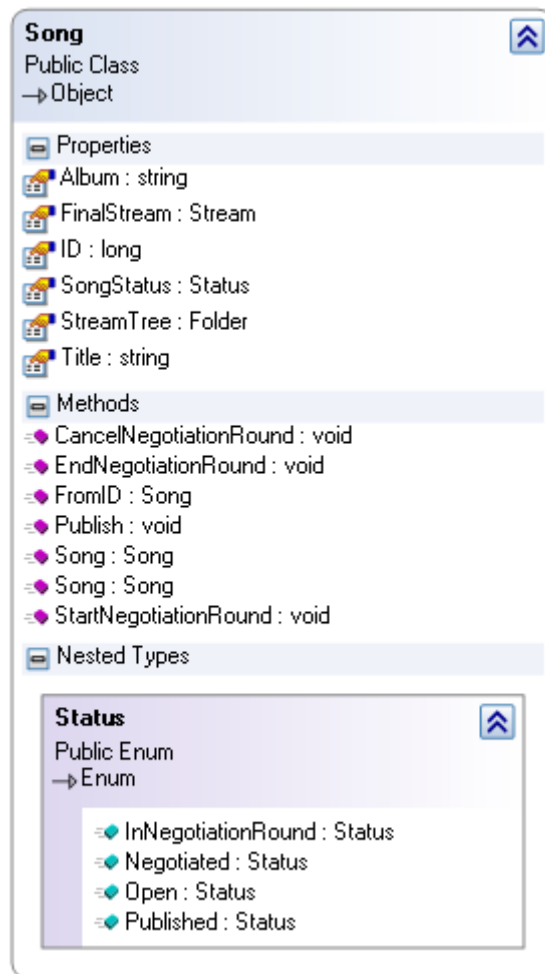


Figure 3.16: Song class.

In the song class, see Figure 3.16, we made methods for starting and ending negotiation rounds as well as for publishing the song. When a negotiation round starts, the song's administrators can negotiate the percentage of those who asked for a share of the song's profit. If the negotiation round is ended, not cancelled, no further negotiation round can be set for that song, since the percentages are already agreed.

The StreamTree property gives access to the root Folder of the song, allowing navigation through all the streams the song is composed of.



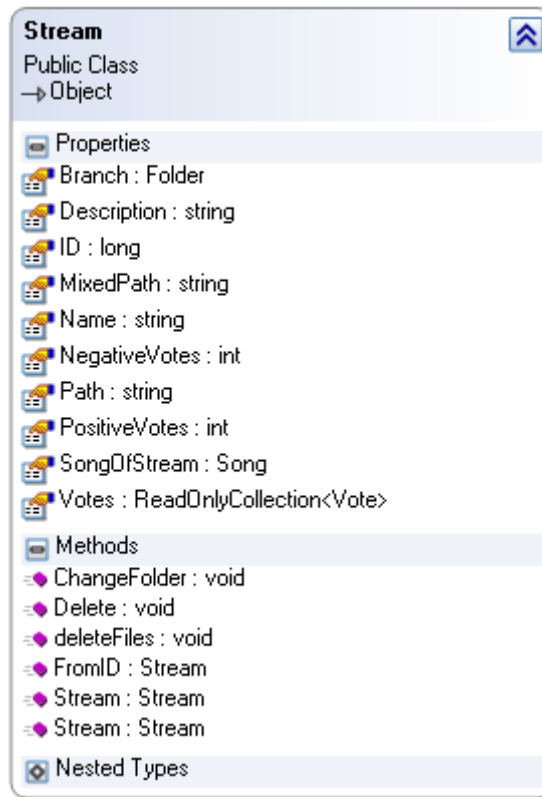


Figure 3.17: Stream class.

The Stream class (see Figure 3.17) holds the data needed in order to find the audio streams which the song is composed of. The Stream class provides properties and methods to change its name, and to change the folder in the song's folder hierarchy where the stream belongs.

More importantly, the class provides the necessary methods to convert the audio stream formats, required to support the different web-browsers that exist in the market. Also, there is a method inside the class which is in charge of mixing the audio tracks. We explain the implementation of those methods later in Chapter 4. We also designed a property to access the votes of the stream, which can be used later to decide which stream should win a contest or just to display the vote count to the public. This class is used by

the class `Song` (see Figure 3.16) to refer the final audio stream of a song and it is heavily used in the `Folder` class (see Figure 3.18).

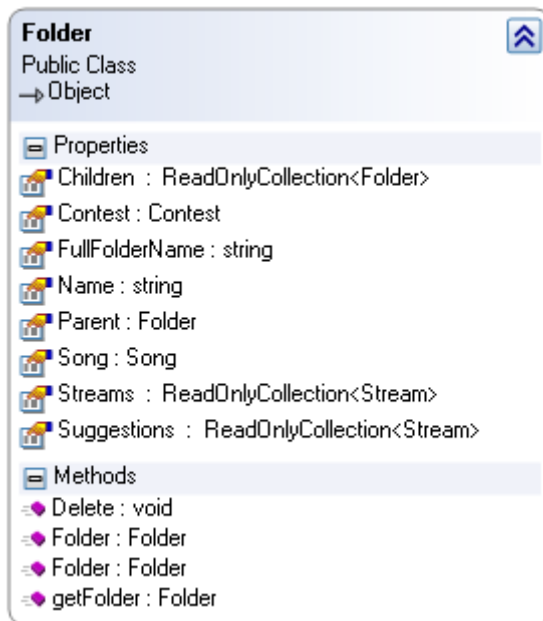


Figure 3.18: Folder class.

The `Folder` class generates a hierarchy by having a property, `Parent`, which points toward the parent folder and also a property, `children`, which points toward a list of children in the folder achieving a folder hierarchy which we can navigate in. Each folder has a list of streams and suggestions. If the property `Contest` of the folder is set, then all the suggestions stand for the contributions in the contest.

The `FullFolderName` property builds the folder's path by navigating through the folder's parents. Finally, we made the `GetFolder` method which searches for a folder given its path. The `Negotiation` and `Contest` classes have methods for starting them, cancelling them and responding to them. They are not shown here since they are quite big classes. You can see them in Appendix B.

### **3.4.4 Controller**

The different controllers implemented are basically waiting for a GET or POST HTTP request. When the request occurs, an action (a method) of the controller is triggered. Those methods are just bound to parse the parameters sent in the GET or POST request and to call methods in the Model component for making the requested actions. At the end, the controller will generate a view, usually with the data provided by the Model as well.

### **3.4.5 View**

The views are written in HTML5 and generated by ASP.NET code, but the interesting part resides in the several scripts that we have implemented using Javascript and JQuery. The JQuery code in the views performs asynchronous data and views requests and provides some real-time interactivity between the user and the web-page. We comment some of the implementation details in Chapter 4.

## 3.5 Chapter summary

In this chapter, we have described, in overall, the important aspects of the design of the application. Thus, first we commented the requirements of the customer with all the details that we were given by him. Then, we discussed the Model-View-Controller software architectural pattern, since we based our system design on it.

Later, the two most important design components of the system, the Database and the Program are presented.

The Database is discussed along its tables and the most important fields of the tables, so the reader can understand their need and meaning. We explained the connection bridge between the code and the database and how they work together.

Moreover, we commented and showed the most important classes of the code, mainly by describing their methods and structure since their fields can be inferred from the database subsection.

Finally, we briefly discussed the Controller and View components.

## 4 Implementation

This chapter discusses those implementation aspects which are particularly interesting, namely the security, the audio processing and the web-pages-user interaction.

### 4.1 User Identity Security

The user authentication has been something that we have taken special care of. Our system is prepared to make monetary transactions in the future and in this stage of development the system already gives an interface for negotiating with money and songs between users. Therefore, we must assure that the system is trustful. Both identity and user information must remain safe.

Currently, we store the user's nickname, email and password in the database. If we were not careful enough and we left an open door through which attackers can use to access our database and steal the data, the users' passwords would really compromise the users' security if they got stolen. Many users will use the same password they use for their email accounts, e-bank accounts and many other services. Thus, we are not only talking about security within our system but also in many other aspects.

We addressed this problem by storing passwords using a one-way hash encryption algorithm. We do not store the actual password, instead, the one-way hash algorithm, SHA1, generates a different string out of the password given. The special thing about these kind of algorithms is that by not even knowing exactly how the password was encrypted you can go back to original password from the encrypted one. No attacker could ever figure out the original password since not even we ourselves know it. In order to compare passwords when the user tries to log in, we also encrypt the given password with the same algorithm and compare them directly.

On the other hand, once the user has been authenticated, the web-server must know which client is which. This is achieved by using cookies but you have to be very careful while doing this because there are plenty of ways to tamper with the cookies.

A first approach would be to store the username directly in the cookie. This should not be done since, to steal the identity of someone, you just have to create a cookie with the name of the victim and that is all. Secondly, you can think of storing both the username and the password and check them on the server every time. In this case, if the cookie is stolen, the attacker will know the user's password and keeping the password secret was one of the reasons for not storing the actual password in the database.

Instead, we created an authentication ticket, which consists of the authenticated user's name and the expiration date. Then this authentication ticket is encrypted – this time, in contrast with what we did when storing the password in the database, we use a two-way encryption algorithm; the algorithm can encrypt and also decrypt. The ticket is stored encrypted as a cookie in the client's web-browser and decrypted and checked at the server side. Note that it is not necessary to store the user's password to check the authenticity of the ticket in the server, since the server, which is the only one who knows the encryption key, is also the only one which can generate a valid ticket. If the ticket can be decrypted and the control data is okay, then the ticket is valid.

In order to implement these encryption techniques we have used the `System.Web.Security` namespace of the .NET platform which can encrypt and decrypt our data using algorithms that have been proven in real applications.

## 4.2 Audio Conversion and Mixing

Since the different web-browsers available for the Internet can only reproduce the audio format that they have chosen, our web-site has to provide several audio formats in order to reach as much people as possible.

The conversion is achieved by using the free source project FFMpeg[5] which can decode and code several hundreds of audio and video formats and codecs by using the library libavcodec. Since this library is cross-platform, we can use it in our Windows server machine but FFMpeg could be used in linux or unix as well, if the customer decides to execute the web-site by using Mono in the future.

We use FFMpeg by launching a new process and making a pipe between the web-server and the FFMpeg process where we send the audio data as input and the FFMpeg process stores the converted audio file for us. Since FFMpeg is not a software developed by us, the customer will have to be aware of updates for security, performance or bug reasons.

We also need to mix the audio between different audio streams. For that purpose we chose SOX[4] – A free source project for audio editing. Among other things, SOX allows mixing several audio tracks in a very easy way.

## 4.3 JQuery Asynchronous Requests

Since the customer wishes the web-site to be as dynamicly loaded and interactive as possible, this project makes an intensive use of Asynchronous Requests to the web-server by using the JQuery Javascript library.

There are several cases where a user's action does not imply a web-page refresh. For instance, when a user uploads a new track, he or she can see the upload process in real-time and when the process is completed the track is immediately displayed in his or her track list. In order to achieve this and provide interactivity, we must download and upload data to the server asynchronously. In Figure 4.1 we can see a commented piece of code extracted

from our project for managing the an asynchronous POST request.

```
/*The variable data holds the information that
we need to pass to the web-server*/
var data = new FormData();
data.append('audioFile', event.dataTransfer.files[0]);
data.append('branch', liUploading.getAttribute('id'));

/*Initiates the POST request with our selected data
giving the url written below and providing
a callback function progressUpload, where we trace
the progress of the upload and uploadSuccessful where we receive
and treat the server's response. This function is non-blocking.*/
$.ajax({
    url: '/User/SongUpload/<%: ViewData["songId"] %>',
    xhr: function () {
        uploadXhr = $.ajaxSettings.xhr();
        uploadXhr.upload.addEventListener('progress',
                                           progressUpload, false);
        return uploadXhr;
    },
    data: data,
    cache: false,
    type: 'POST',
    success: uploadSuccessful
});
```

Figure 4.1: JQuery asynchronous POST request.



When we receive the server's response we need a way to indicate which kind of response it is, because we will not always want to perform the same action. When we upload a track, the usual response would be to add the received html code, which would be the new track item, to the current html code. However, there is another type of response that could be sent. For instance, the upload or conversion process may fail and in that case the server would send us an error message. We address this problem by using html comments in the response and at the client side we search for these comments to decide what we should do with the response. Figure 4.2 provides an example.

```

/* the variable event is the server's response as HTML code. */
if (event.search("<!--Replace-->") > -1) {
    /* Replace type response.
        We should add or replace the given HTML code */
} else if (event.search("<!--FullScreenMessage-->") > -1) {
    /* Full screen message type response.
        We should display the given HTML code as a
        message over the current web-page. */
} else if (event.search(.....) > -1) {
    .....
}

```

Figure 4.2: JQuery asynchronous POST request.

## 4.4 Chapter Overview

In this chapter we have commented the most relevant aspects of the implementation of the project's features. We thought it would be worth to explain the implementation of the security in detail, since otherwise it could have passed unnoticed because the security aspects tend to be ignored until someday the security is broken.

The conversion and mixing are two processes which will demand a huge amount of computational resources. It was important to define how these processes are performed and which libraries are used.

Finally, the main JQuery techniques uses are described, which have been quite important for providing a nice user interaction with the web-site.

## 5 User Interface

This section shows the final user interface running the basic features that the web-site provides.

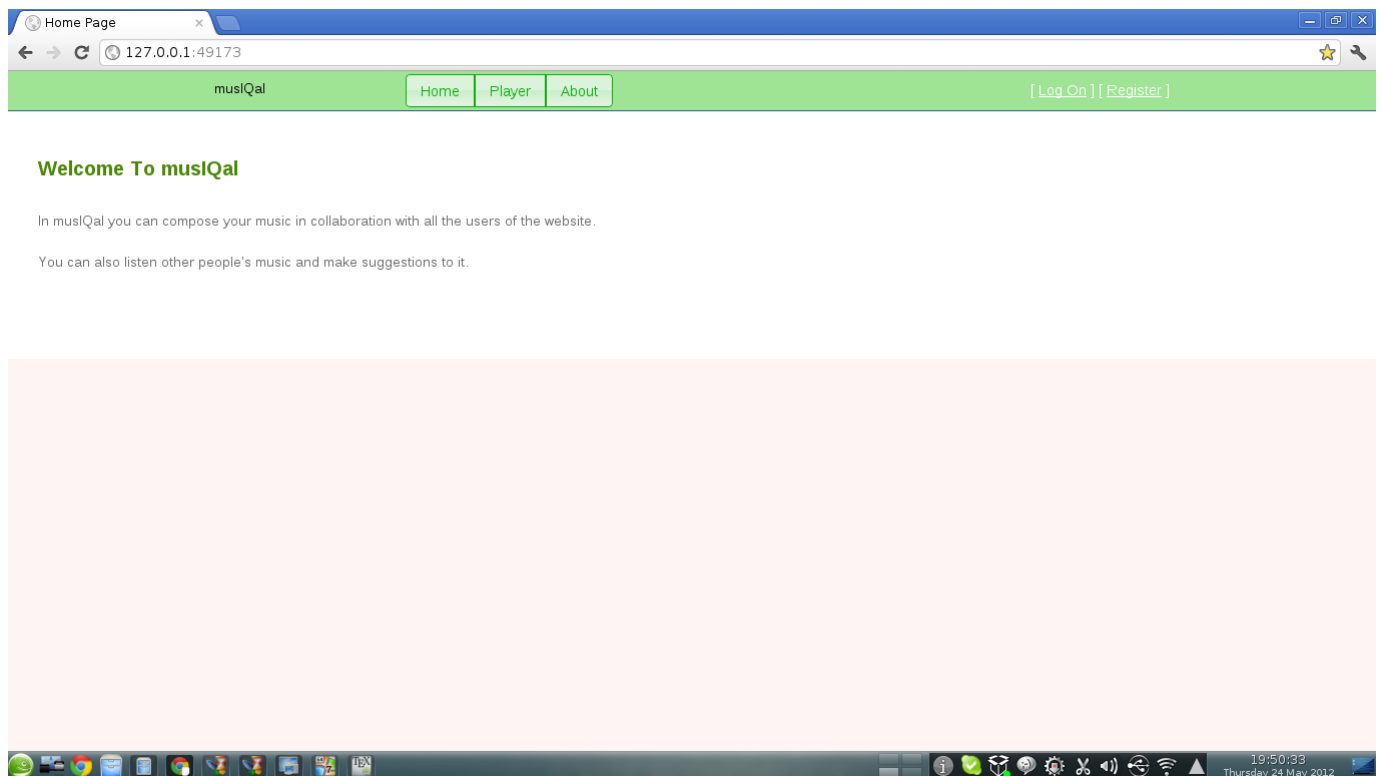


Figure 5.1: The main web-page of the web-site. This page is still to be filled by the customer.

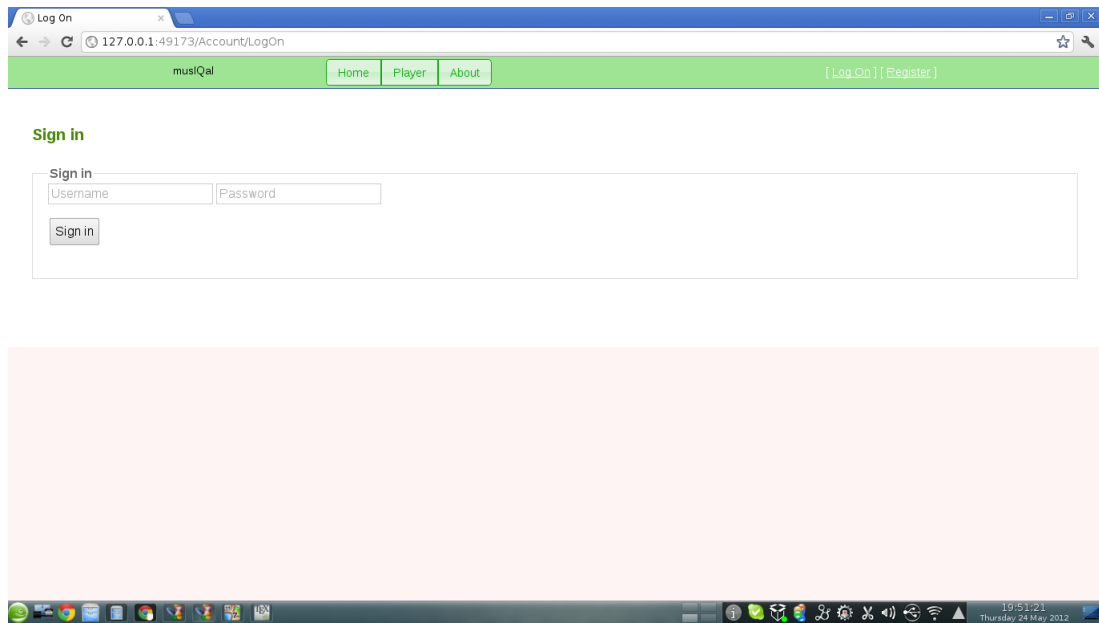


Figure 5.2: User's Login web-page.

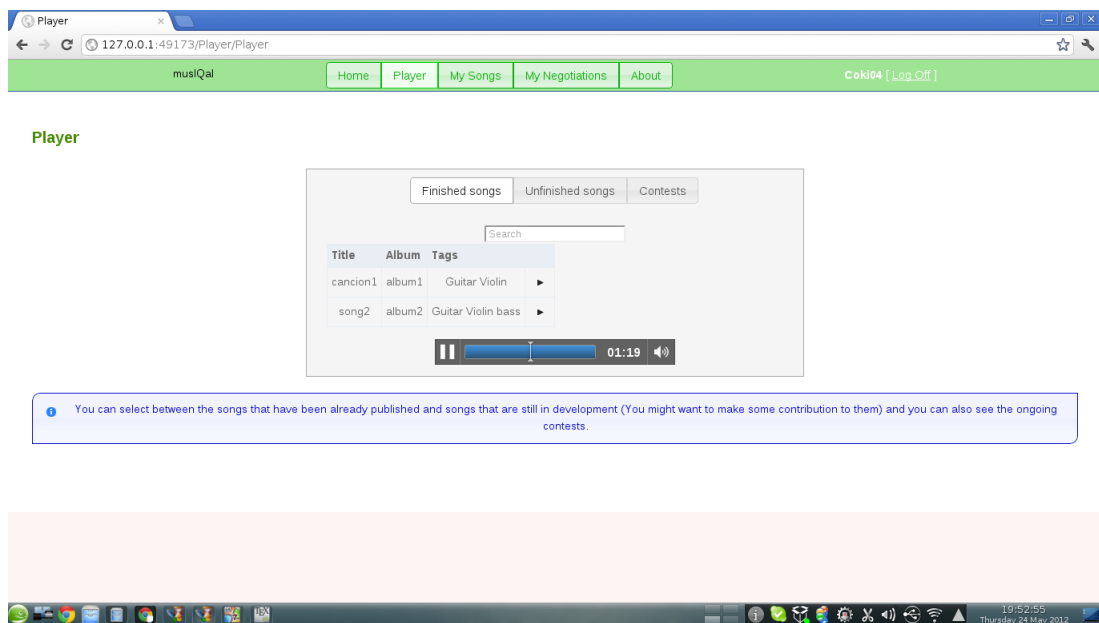


Figure 5.3: List of published songs in the system.

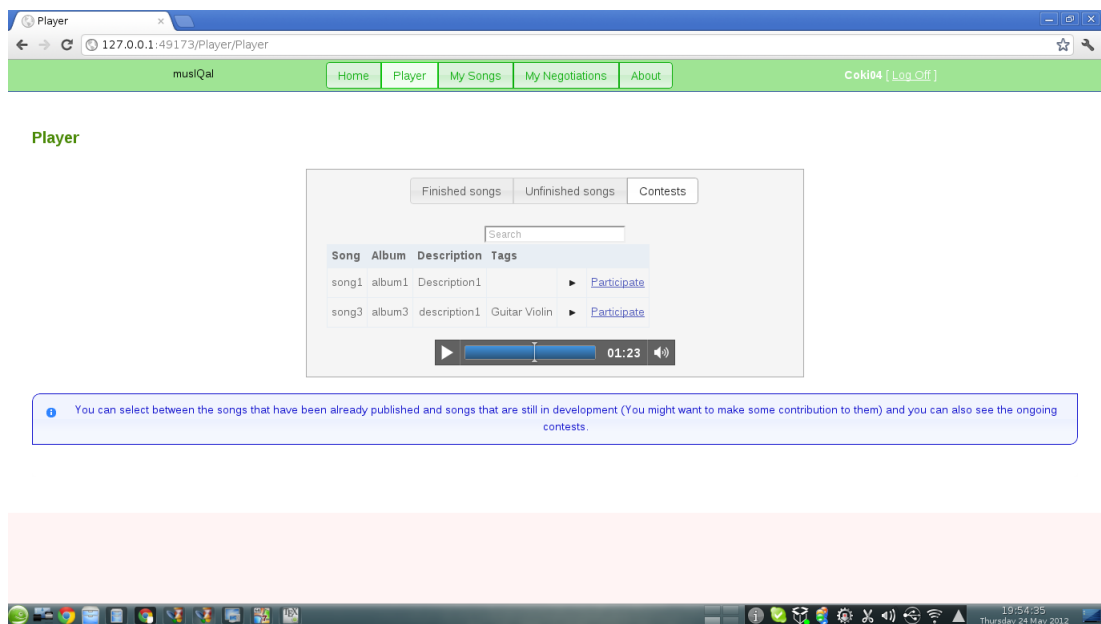


Figure 5.4: List of ongoing contests in the service.

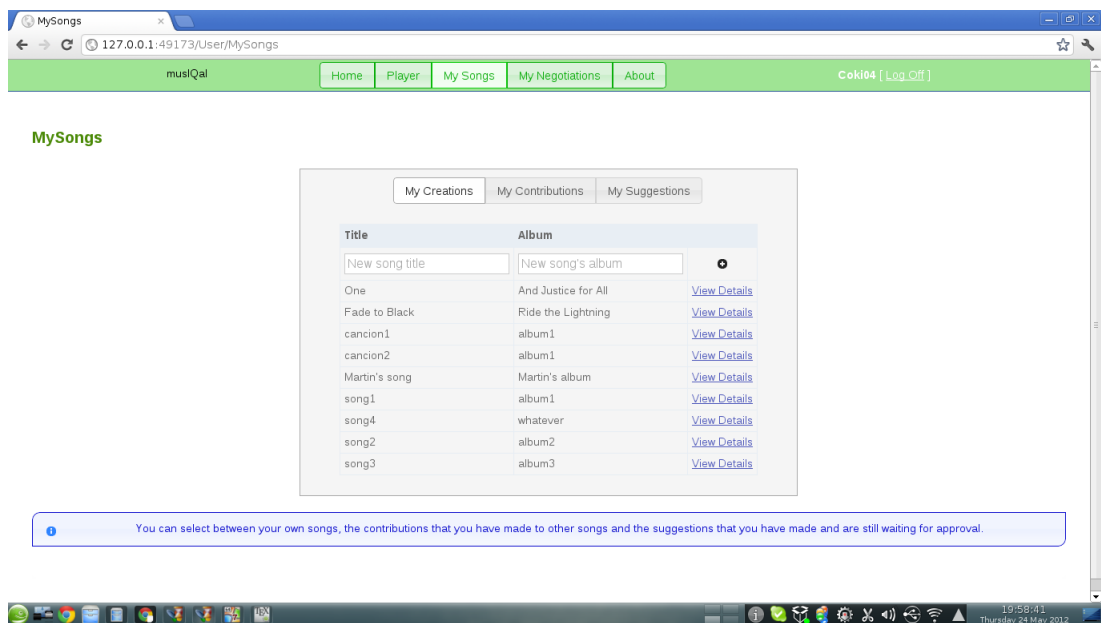


Figure 5.5: List of the songs created by the user.

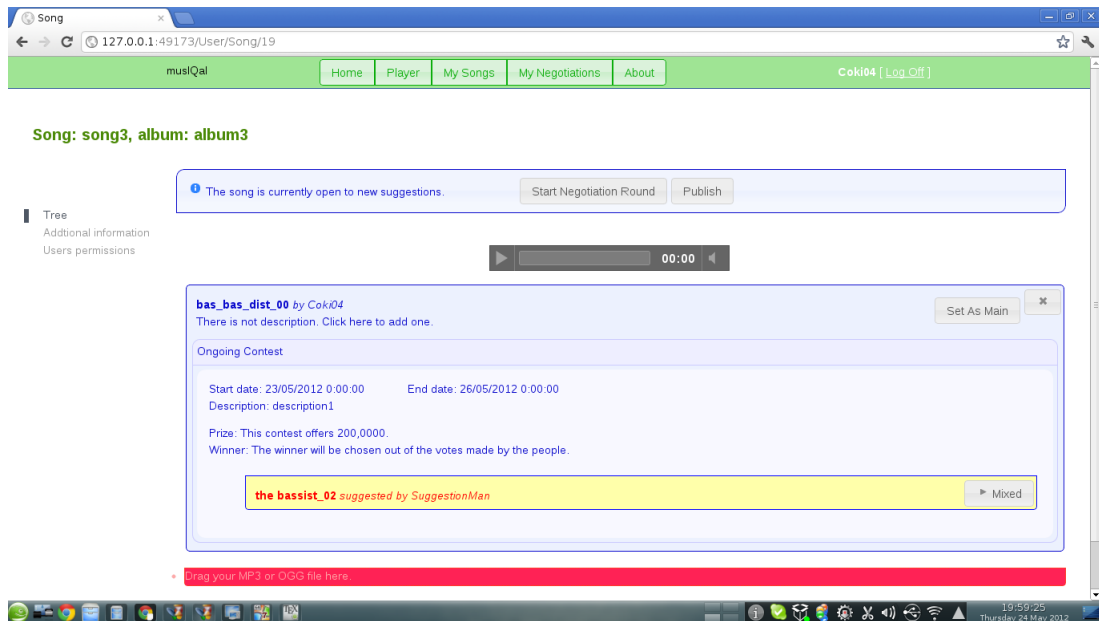


Figure 5.6: Page used to upload tracks and keep track of the song status.



Figure 5.7: Page to make suggestions to other people's songs.

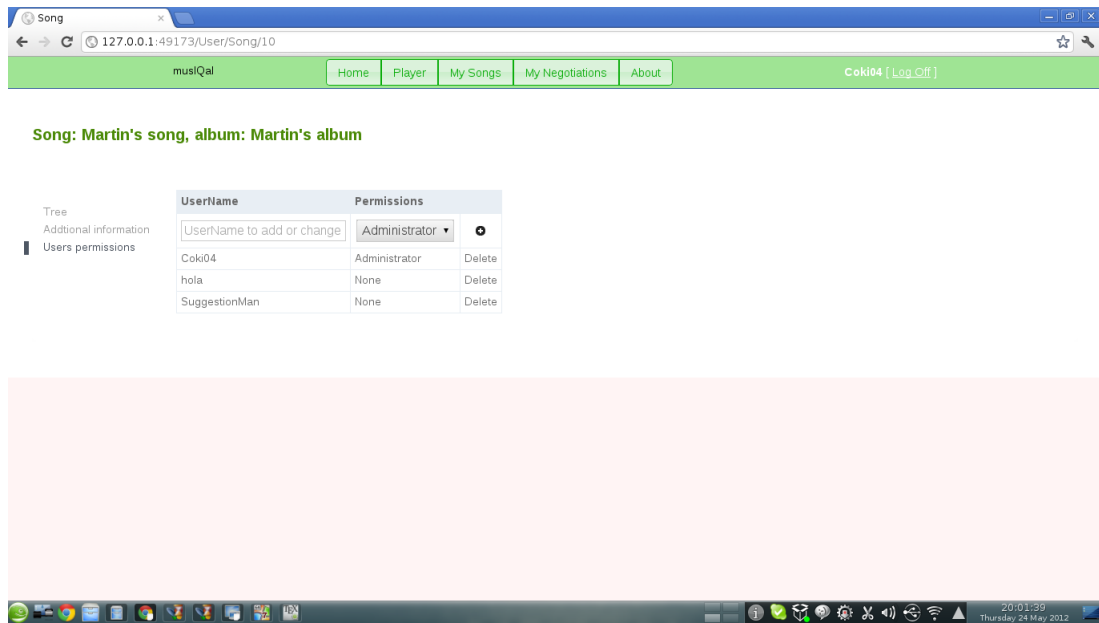


Figure 5.8: Users permissions of a song.

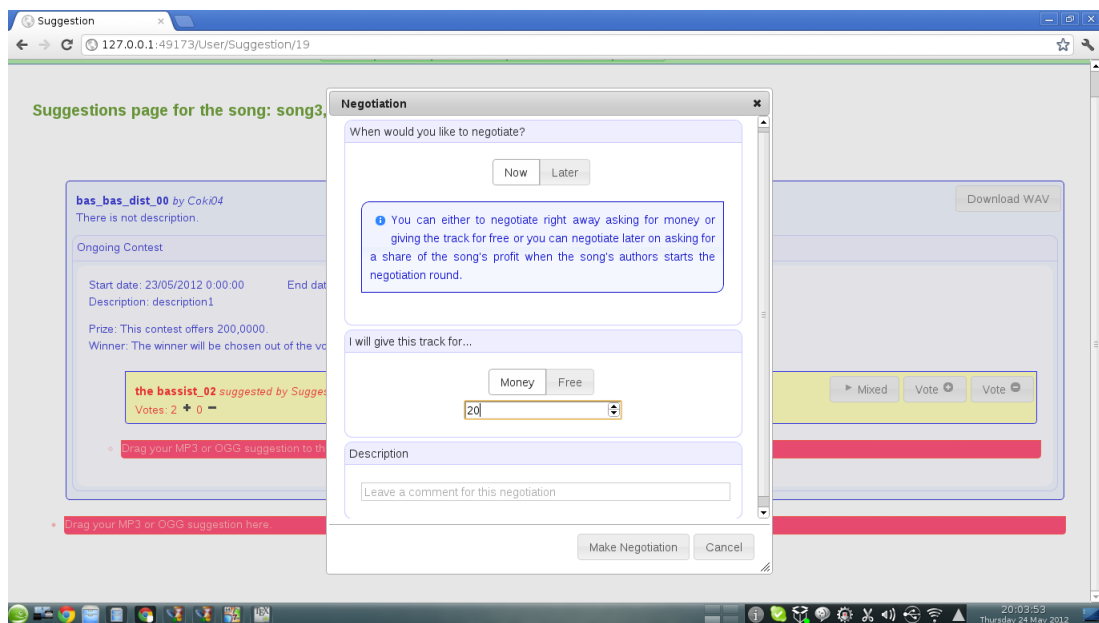


Figure 5.9: Negotiation proposal.

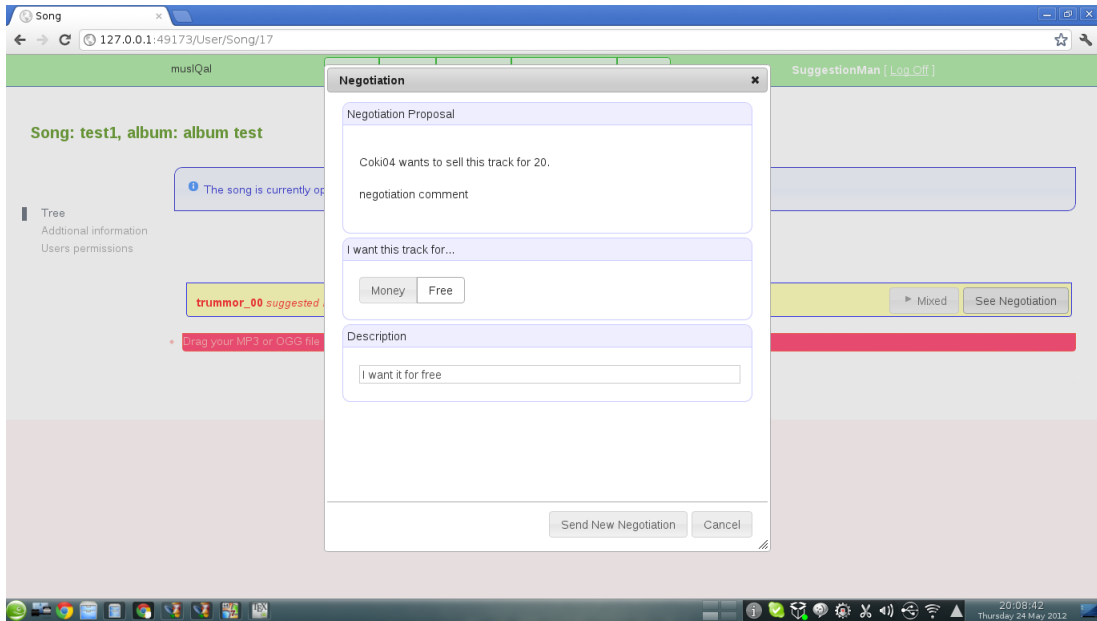


Figure 5.10: Negotiation response.

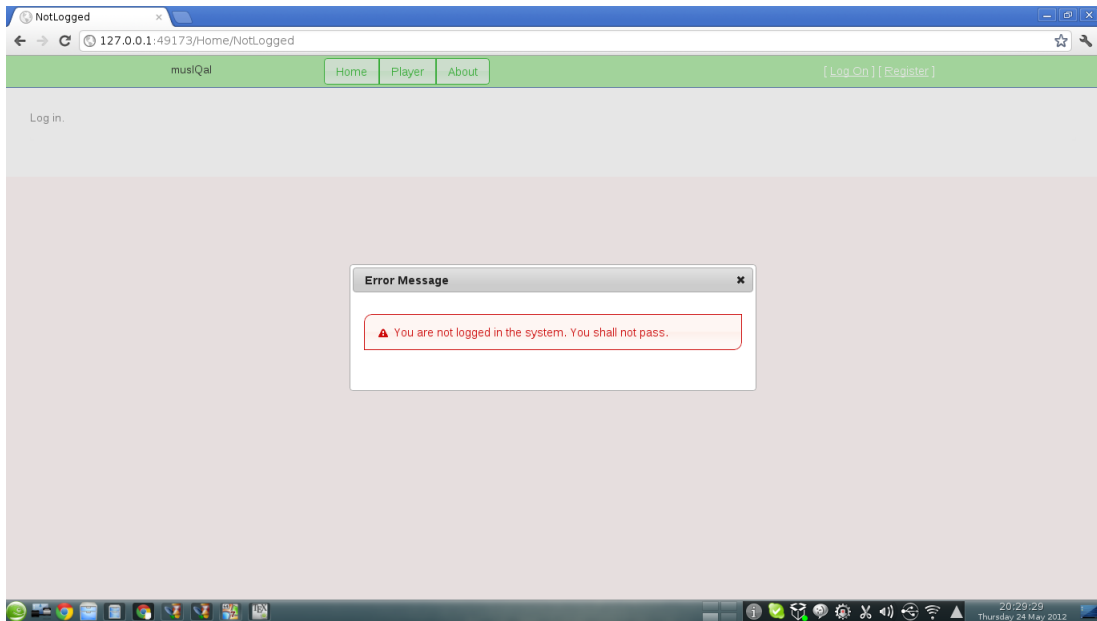


Figure 5.11: Error page.



## 6 Evaluation

This chapter evaluates different aspects of the project in retrospect, analysing what have worked out, what have not and what could have been better. This is an essential part of every project or work and the best way to improve and make the work meaningful.

### 6.1 Time Planning

We followed the time table proposed by the coordinator of the dissertation with several adjustments to better fit our project demands, such as incrementing the time used for research about the different technologies and tools used and extend the time for writing of the Design chapter and reducing the time for writing the Implementation chapter since it fitted our dissertation better. These adjustments were made possible thanks to our supervisor and her experience also helped us with the timing aspects.

### 6.2 Acquired Knowledge

Just by looking at the table of contents of the background chapter you can realise that this project has been made using quite a lot of technologies and tools. Moreover, one of the most important technologies, HTML5, is in continuous development since it has not reached a final version. This means that we have learnt a good number of technologies which are extremely much used by countless number of companies. Furthermore, some of them, such as HTML5, JQuery and the entity framework, are quite new and extremely powerful and their demand in the market is increasingly unstoppably.

We had previous experience in C# and SQL programming but almost no knowledge in web related technologies. This project made us learn about the languages used for web development almost from scratch. Knowledge in web development also has been beneficial for us since this field is constantly expanding and will really help us in our future as computer engineers.

### **6.3 What could have been done differently**

Thanks to the fact that we had enough time to research and think about the technologies to use, we have not gone into any problems because of the technologies used.

The communication with the customer has been also great. We were working in two-weeks iterations where, at the end of each iteration, we showed the work done to the customer and he provided us with feedback and new requirements. These iterations helped us to stay on the right track.

The adaptation to the new technologies was also quite successful since we started to have results quite soon. We were also happy with the interleaving between the writing of the dissertation and the coding. About the visual style (CSS), we were never actually good at it and the result is not as good as would be desired. This fact was acknowledged by the customer and in consequence he decided that we should focus on designing and implementing all the required features rather than spending time on the visual aspect. What we would improve is the internal group coordination which could have been a bit better. Specially, sharing the code by using SVN did not work exactly as expected. Additional research on SVN or similar system would be needed for further projects.

## 7 Conclusions

We have succeeded in implementing all the required features by our customer, Martin Blom, providing the clients with a web-site which stores their music and allow them to develop the music collaboratively with other users. Furthermore, their music can be listened to and mixed on the web-site itself, without requiring any additional software. Finally, the users can negotiate for tracks with other users and they can also start contests for their songs to encourage other people to collaborate with them.

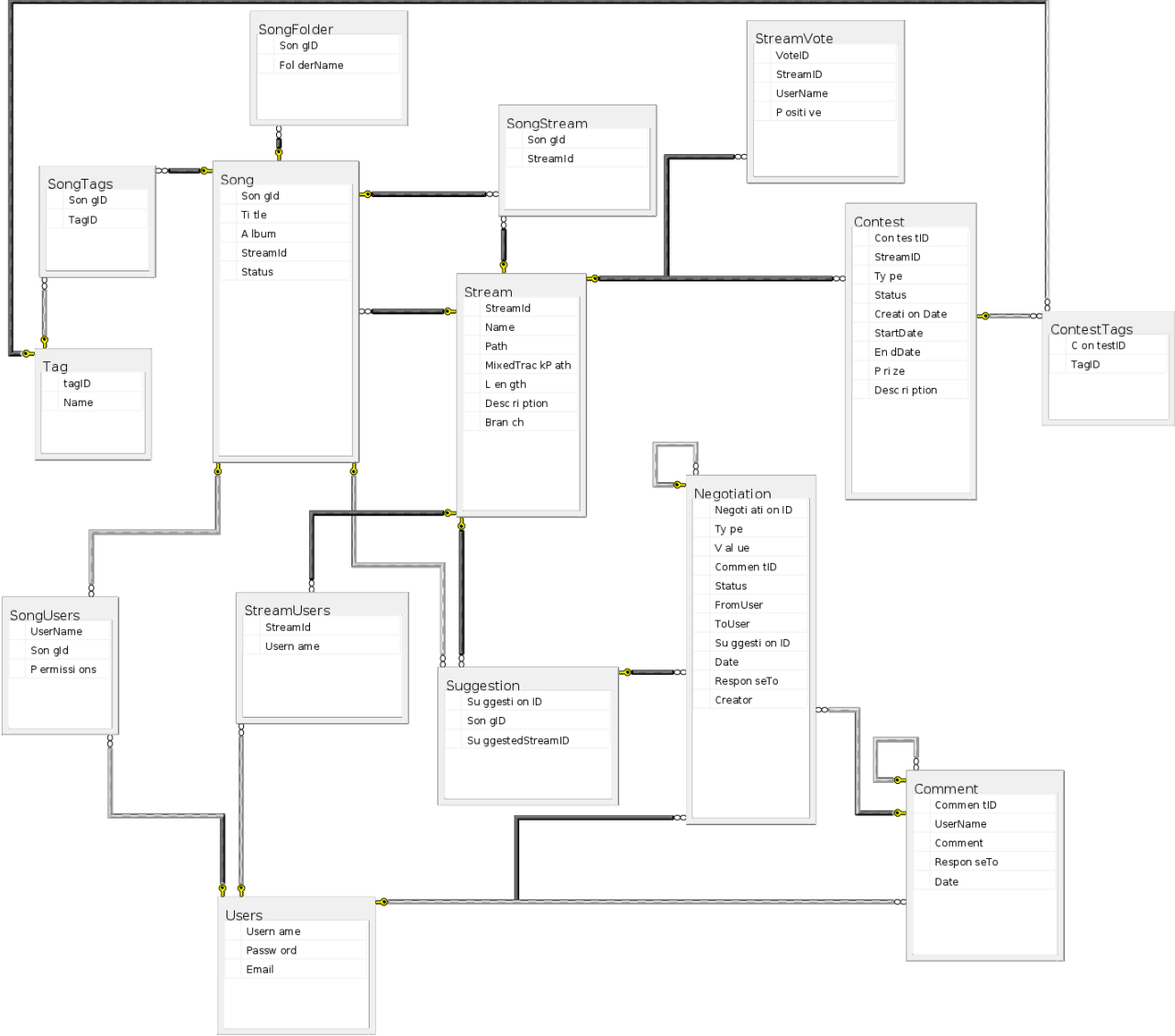
All these features have been implemented using the latest technologies (HTML5, JQuery) as demanded.

### 7.1 Future work

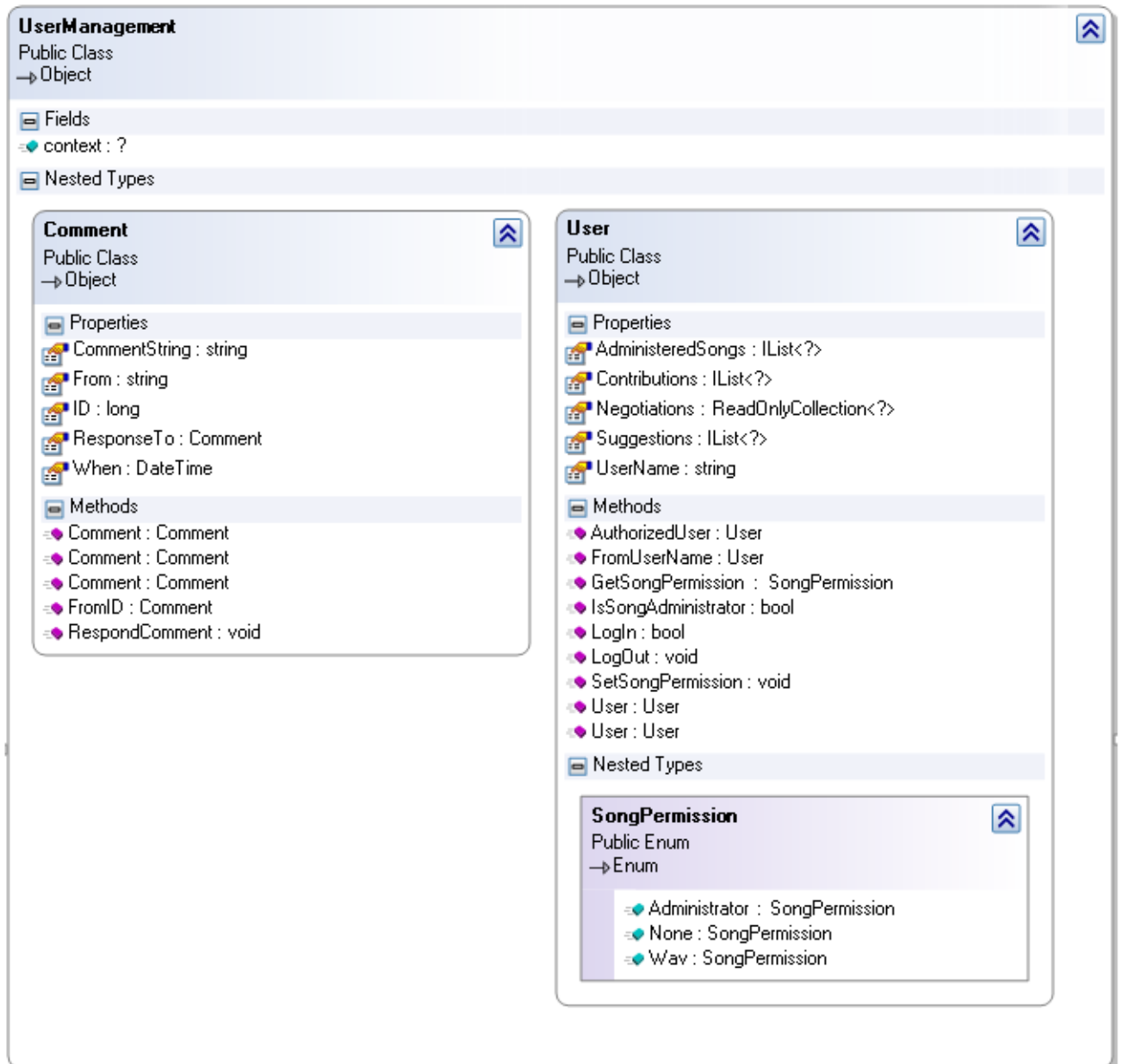
If this project gets successful on the Internet, with a high process load, this project will need to significantly reduce the processing time consumed by the audio converting and mixing. Probably a good field for researching in order to address this problem would be to process the audio by using the GPU instead of the CPU. Some encoding algorithm could be parallelized. Thus, the system could show speed improvements programming it in OpenCL[7] and executing the code on the GPU.

One more feature needed is to offer actual banking transactions. Thus, the users could actually send and receive money for their music. This would require researching the API of services such as Paypal[8] or Google Checkout[6] or any other e-banking service chosen by the customer of this project.

# Appendix A: Database Diagram

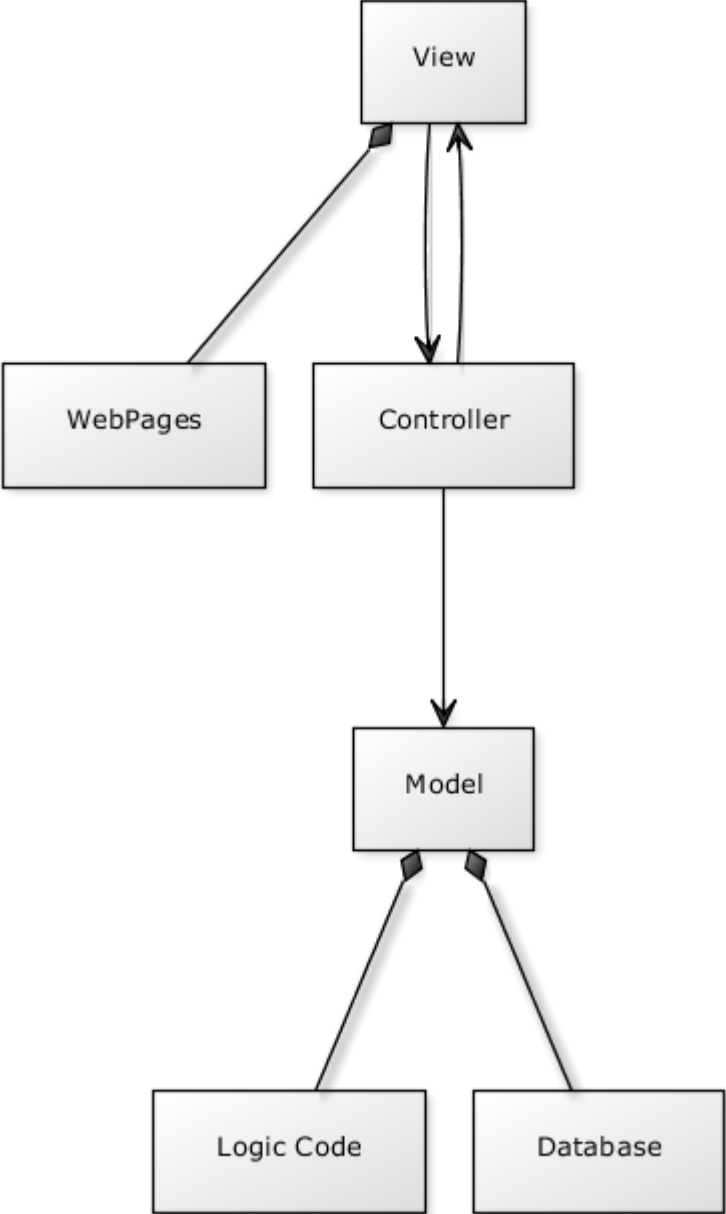


## Appendix B: Model Class Diagram





# Appendix C: Whole System Diagram



## Bibliography

- [1] Ubetoo AB. Ubetoo — digital music distribution — paid streaming — <http://www.ubetoo.com/>, 2008.
- [2] blip. Free music — listen to music online — free streaming radio — blip.fm — <http://blip.fm/>, February 2012.
- [3] blubster. Blubster free music downloads — <http://www.blubster.com/>, February 2012.
- [4] et al. Chris Bagwell. Sound exchange’s project web-page — <http://sox.sourceforge.net/>, 2012.
- [5] FFMPEG. Ffmpeg’s project web-page, 2012.
- [6] Google. Google checkout web-page — <https://checkout.google.com/>, 2012.
- [7] Apple inc. Open computing language (opencl) web-page — <http://www.khronos.org/opencl>, 2012.
- [8] Paypal inc. Paypal web-page — <https://www.paypal.com/>, 2012.
- [9] A.R. Jones. *Mastering ASP.NET with C#*. SYBEX, 2002.
- [10] jukeboxalive. Download music, upload music, listen to music free — jukeboxalive.com — <http://www.jukeboxalive.com/>, February 2012.
- [11] SoundCloud Ltd. Soundcloud - share your sounds — <http://soundcloud.com/>, 2007.
- [12] Ltd MixMatchMusic. Mixmatchmusic - because connected fans are loyal fans — <http://www.mixmatchmusic.com/>, 2007.
- [13] LLC Musiclab. Free music download - download free mp3 music - best limewire alternaative! — <http://www.bearshare.com/>, 2012.
- [14] Mehran Nikoo. Mehran nikoo’s mvc blog entry — <http://mnikoo.net/2010/06/14/model-view-star/>, 2012.
- [15] w3c. W3c html 5 elements — <http://dev.w3.org/html5/spec/section-index.html>, 2012.