SPEECH CONTROLLED MULTIMEDIA SYSTEM

By
Joseph Mc Dermottroe

B.A. (MOD.) CSLL
TRINITY COLLEGE
DUBLIN
MAY 2003
Supervisor: Johnathan Dukes
I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university.

Dated: May 2003

Signed: Joseph Mc Dermottroe
# Table of Contents

Table of Contents iv

List of Figures viii

Abstract x

Acknowledgements xi

1 Introduction 1

1.1 Overview ......................................................... 1
1.2 Background ...................................................... 3
1.2.1 Speech Technologies ........................................... 3
1.2.2 Multimedia System ............................................ 4
1.3 Aims ................................................................. 4
1.4 Outline ............................................................ 6

2 Technologies 7

2.1 Introduction .................................................... 7
2.2 ASP.NET .......................................................... 7
2.2.1 What is ASP.NET ............................................... 7
2.2.2 Why use ASP.NET ............................................. 8
2.2.3 Static versus Dynamic Web Pages ............................ 8
2.2.4 Why use Dynamic Web Sites? ................................. 9
2.2.5 What Programming Language to Use ........................ 9
2.3 Web Services ...................................................... 10
2.3.1 What are Web Services? ...................................... 10
2.3.2 Why I used a Web Service ................................... 11
2.4 Database .......................................................... 13
2.4.1 What is a Database? .......................................... 13
List of Figures

3.1 Speech Controls .................................................. 21
3.2 Visual Studio Grammar Tool .................................. 26
3.3 Sample Sentence .................................................. 27
3.4 Basic Grammar .................................................... 28
3.5 QA Input Section ................................................. 29
3.6 Semantic Map ..................................................... 31

4.1 Now&Next Programs ............................................ 38
4.2 Toolbar with Custom Server Controls ....................... 39
4.3 Select the User ................................................... 40
4.4 Code to get a list of Users .................................... 41
4.5 Code for Semantic Tag ......................................... 41
4.6 User’s Main Menu ............................................... 43
4.7 Show New Services ............................................. 44
4.8 Shows a User’s CDs ............................................ 45
4.9 A List of Tracks of a Specific CD ............................ 47
4.10 Client Code to get Track Position ......................... 48
4.11 Select either Free or PPV Videos .............................. 50
4.12 Show all the Videos which cost nothing .................. 51
4.13 Choose between a Video List and Purchased Videos .... 53
4.14 Shows all the Videos purchased by the User ............ 54
4.15 Shows a List of Videos the User can buy .................. 55
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.16</td>
<td>Confirm Video Purchase</td>
<td>57</td>
</tr>
<tr>
<td>4.17</td>
<td>Show all the Television Channels</td>
<td>58</td>
</tr>
<tr>
<td>4.18</td>
<td>addUser Method</td>
<td>61</td>
</tr>
<tr>
<td>4.19</td>
<td>Get the User’s ID Number</td>
<td>63</td>
</tr>
<tr>
<td>4.20</td>
<td>Complex Tracks Table</td>
<td>68</td>
</tr>
<tr>
<td>4.21</td>
<td>Standard SQL</td>
<td>68</td>
</tr>
<tr>
<td>4.22</td>
<td>Call Stored Procedure</td>
<td>68</td>
</tr>
<tr>
<td>4.23</td>
<td>Call deleteVideo Stored Procedure</td>
<td>69</td>
</tr>
<tr>
<td>4.24</td>
<td>Basic Stored Procedure</td>
<td>70</td>
</tr>
<tr>
<td>4.25</td>
<td>listTracks Stored Procedure</td>
<td>71</td>
</tr>
<tr>
<td>4.26</td>
<td>addUser Stored Procedure</td>
<td>72</td>
</tr>
<tr>
<td>4.27</td>
<td>updateUser Stored Procedure</td>
<td>73</td>
</tr>
<tr>
<td>4.28</td>
<td>deleteCD Stored Procedure</td>
<td>74</td>
</tr>
<tr>
<td>4.29</td>
<td>AddUserCredit Stored Procedure</td>
<td>75</td>
</tr>
<tr>
<td>4.30</td>
<td>A Marshalled C# Structure</td>
<td>79</td>
</tr>
<tr>
<td>4.31</td>
<td>How to Declare a DLL within C#</td>
<td>80</td>
</tr>
<tr>
<td>4.32</td>
<td>Initial Page of the Configuration Tool</td>
<td>82</td>
</tr>
<tr>
<td>4.33</td>
<td>Add a User to the Database</td>
<td>83</td>
</tr>
<tr>
<td>4.34</td>
<td>Select a User account</td>
<td>84</td>
</tr>
<tr>
<td>4.35</td>
<td>Delete Music from Database</td>
<td>85</td>
</tr>
<tr>
<td>4.36</td>
<td>Modify a Video’s Attributes</td>
<td>86</td>
</tr>
<tr>
<td>A.1</td>
<td>CDList and CDs Tables</td>
<td>100</td>
</tr>
<tr>
<td>A.2</td>
<td>Channels and MainPage Tables</td>
<td>100</td>
</tr>
<tr>
<td>A.3</td>
<td>Media and NewServices Tables</td>
<td>101</td>
</tr>
<tr>
<td>A.4</td>
<td>PPV and Television Tables</td>
<td>101</td>
</tr>
<tr>
<td>A.5</td>
<td>Tracks and Users Table</td>
<td>101</td>
</tr>
<tr>
<td>A.6</td>
<td>VideoList and Videos Table</td>
<td>102</td>
</tr>
</tbody>
</table>
Abstract

This project uses speech recognition to eliminate the need for a remote control to control a TV, CD player and DVD player. Using only voice control, a user can navigate a web site which lets the user watch TV and play Videos and Music. At the same time, it should be an easy-to-use system which explains why all those devices have been unified into an integrated web-interface.
Acknowledgements

Firstly, I would like to thank my parents. Without their constant help and support throughout my life I wouldn’t have been able to get this far.

I would like to thank my supervisor, Johnathan Dukes, for all his great help and advice throughout the year.

I would like to thank Carl for his guidance throughout my four years in CSLL.

Joseph Mc Dermottroe
1st May, 2003
Chapter 1
Introduction

"’Tis better to be silent and be thought a fool, than to speak and remove all doubt.”
—Mark Twain

1.1 Overview

A modern-day home comes complete with a television, CD player, DVD player, satellite or cable system and so on. In America the phrase *Entertainment Centre* has been coined for this. We all know what it looks like. In one corner of our sitting rooms lies a case where all this appliances are stacked one on top of the other. It may look pretty on the outside but there are a multitude of cables jumbled behind it all.

Then, of course, we come to the remote control or should I say the remote controls. Every device has to have one. There’s one for the video, the TV, the CD player and the television and the DVD player. For those with satellite there’s even a satellite remote control. That makes six remote controls to control them all. Of course you could always purchase a universal remote control which can control everything, well maybe not the fridge. But they are expensive to buy and complicated to control. If you can’t program a video recorder what chance have you to program one of these
gizmos. How many parents have to ask their children to program the video or to set up the channels on the TV? There has to be another way.

By the year 2000 more than half of American homes had a computer [10]. A great advantage of a computer is that it can do lots of things. A person can watch DVDs, listen to their favourite songs. Even watch TV. That sounds like an “Entertainment Centre”. It should be much easier to set up than the TV, DVD player et al. Wouldn’t it be great if we combined all those things into the one computer? That is what my project aims to do. Eliminate the need for the separate TV, separate DVD player and all. They all come built into a standard cheap computer. My project allows you to watch television, listen to music, watch videos and more. It can even be upgraded. Can you really upgrade your DVD player or TV without throwing out the whole unit?

That sorts out the problem with the all the devices. Now we’re left with the problem of remote controls, or specifically the lack thereof. Using a computer is probably the stuff of nightmares for most people. A large percentage of the population do not and cannot use one. That would then nullify the advantage of my one box idea to replace the TV etc.. What we need to do is make the computer accessible to everyone. The most obvious way, would be to use your voice.

Most sci-fi films show people controlling their TVs, robots etc using their voice. Such technology has always been the work of sci-fi or maybe only accessible to those in labs. With computers nowadays becoming so powerful such fanciful technology is now built into all new computers. Most do not know it but speech recognition is built into Windows 2000 and Windows XP. That is the fundamental component of my project. The interface on the screen is navigable simply using your voice. There’s no need for a remote control. Lost or broken remote controls become a things of the past. Of course, it’s much more intuitive to use your voice than a remote control.
1.2 Background

Searching the internet I could not find any information about a properly integrated voice-controlled multimedia system. Instead, this section will be split into two. The first section will deal with speech technologies. The second will discuss integrated entertainment systems.

1.2.1 Speech Technologies

There are a huge range of uses of speech technologies. These range from small devices like mobile phones or remote controls to systems which can answer phones and turn of lights. The common factor of all these is that they can be controlled by voice.

Since the Nokia 8890 was released nearly every Nokia mobile phone has a voice-activated dialling feature. The user stores a voice tag for some phone numbers. Later, when they say one of these tags, the phone automatically calls that number. This is quite a basic function but it does show that voice-technology can be used in such small devices.

Philips developed a remote control which could be voice-controlled. By saying channel numbers the user could change channels. Unfortunately, the product has since been discontinued.

Speech technology could be extremely important to those people with sight impairments. Many companies now have products which allow appliances, telephones etc. to be controlled using their voice. A company called Cintex3 \(^1\) has a product which can answer phones and control any IR (infra-red) controlled devices. If the person has X-10 \(^2\) home automation technology then the software can be used to control these devices.

---

\(^1\) [http://www.freedomofspeech.com/cintex3ecu.html](http://www.freedomofspeech.com/cintex3ecu.html)

\(^2\) [http://www.x10.com](http://www.x10.com)
Microsoft Office XP and Microsoft Windows XP both have speech recognition built in. A user can navigate around Windows using their voice. They can, for example, open programs and control the menus within those programs. A user can dictate a document within Word and this will then be converted into text. This uses the speech technology built into Windows XP.

### 1.2.2 Multimedia System

Some TV companies have been introducing VOD (Video on Demand) to consumers. One such company is Homechoice\(^3\) which provides this service in London. This streams TV channels and videos across a phoneline using ADSL. This is similar to my project in that it streams the media files. However, unlike my project, it does not allow the user to customise their selections. For example: They cannot create personalised audio collections. Instead, they can listen to the collections provided to them by Homechoice.

Recently Microsoft introduced a customised version of Windows XP called Windows Media Centre Edition. This comes closest to my system. It allows a person to watch TV, record videos and look at photographs. According to their press release it provides “one unified place for your entertainment”. This, along with speech control, is the main aim of my project.

### 1.3 Aims

My project has two fundamental aims:

1. Replace the need for devices such as the tv, cd player, dvd player etc with just one box. This makes things look much simpler. Computers now even come with flat screen displays with look very impressive. Much more so that the humble

\(^3\)http://www.homechoice.co.uk
television in most of our homes. A computer is also cheap than buying all the extra components separately.

2. Make it easy to control the system. This is where speech recognition comes into it. Speech is a very natural way for people to communicate. If you can extend this to a speech interface for a computer then it would make such a system much more accessible to the old, the young as well as those with sight difficulties.

These two aims were very important from the beginning but there were also many other aims for the project.

- From the beginning I wanted to make sure that the system was scalable. In other words, that it will work for one user but can be scaled to work for many more. The system should be built from the start to incorporate the features needed to make this possible.

- One of fringe benefits of the speech recognition system used is that it also allows text-to-speech processing (or speech synthesis). This could be used so that anything displayed on the screen can be read out to the user. This has immediate advantages to those with visual impairments. One of the aims of the project was to make the system more accessible to such people. Using text-to-speech allows such people to be able to hear a spoken description of the screen.

- In the news we often hear about people cracking into computer systems. Therefore, security has become very important. I wanted to try to make sure that my project would not have any glaring security holes. I also wanted to make it easy to add in security features if required. It would be time consuming to add such features in myself but I would like to make it easier for others to do so.
1.4 Outline

This project is described in five chapters. Chapter 2 discusses the various technologies used in the project. Reasons as to why I used the various technologies are given in this chapter. Chapter 3 discusses the speech technology used. The chapter will deal with the creation of grammars, the use of the various speech controls and how the controls turn spoken commands into actions on the web page. In Chapter 4 I will show how I implemented the project. Using the technologies described in chapters 2 and 3 the various parts of the project will be built-up. IE: the Graphical user Interface (GUI) and the speech interface. My own personal thoughts on how the project turned out along with the problems that I encountered will be discussed in Chapter 5. I will also use this chapter to show how other users fared with the system. Finally, I will conclude this discussion in chapter 6 when I will show what aims were achieved and what future work could be done on the project.
Chapter 2

Technologies

2.1 Introduction

In this Chapter I will discuss the underlying technologies of my Project. I am separating the underlying theory of the underlying technologies from my implementation of these technologies. My implementation will be described in the next chapter.

Why I used these technologies is quite important. Using a technology just for the sake of it is not just a bad idea, it's a waste of time. In each of the sections below I will give reasons as to why I used each particular technology. The main reason why I would use them is because they fit in with the aims of my project. Another reasons may be that it might simplify the development project.

2.2 ASP.NET

2.2.1 What is ASP.NET

ASP.NET is a Web development platform that provides dynamic web page context to users. A developer can author applications in any .NET compatible language (described in 2.2.5), including Visual Basic .NET, C#, and JScript .NET. Additionally, the entire .NET Framework is available to any ASP.NET application.
An ASP.NET written in a .Net compatible language is stored on the web server. When a user requests the page, the web server loads it and executes the program associated with the page. This code can perform calculations, access databases or call web services. This code displays HTML code in the user’s browser.

2.2.2 Why use ASP.NET

Going back to my fundamental aims for this project I said that I wanted to the interface to be navigable by speech. Since the interface uses Microsoft speech controls I had to use ASP.NET to program these. Microsoft created an excellent tool, Visual Studio.NET, to aid ASP.NET development and to create a speech controlled web site. This really helped me to build the sites more easily than by using manual means.

Later I will describe the database system I used. This system provides easy access to this database server. Debugging the web pages is a feature of Visual Studio which means errors can easily be found. Using other systems I would have had to manually do most of the work which would have taken much longer and would have created pages with more errors in them.

2.2.3 Static versus Dynamic Web Pages

Initially, web pages started out as static pages. The author wrote HTML and this was displayed on the client. The web pages only changed when the author changed them. This is very useful in some circumstances. But there has been a trend towards sites which show information depending on what the user wants etc. For this we need pages which are dynamically created. Many systems are available to do this. My project shows information which can vary over time or by user. Hence, this requires a dynamic approach. I decided to use ASP.NET.
2.2.4 Why use Dynamic Web Sites?

One of the main motivations of programming using ASP.NET was to separate the code into the client/server model. As described earlier we can write code that is executed on the server with the results displayed on the client by using HTML code. This has many advantages. Since the code created is standard HTML, this can be viewed in any Internet browser.

Since all the complexity lies on the server, administration is made much easier. Rather than having to create applications on every client or rolling out code to every client and hence updating every client when needed, all we have to do is change the code on the server. The only configuration needed on the client side is simply the URL of the server. Obviously, this makes configuration quite trivial on the client side.

Since the client only needs to display the HTML code, then the client does not have to be extremely powerful. Obviously, video and music place their own demands on the system. The only computer that has to be powerful is the server. However, this fits in with my aim for the system to be scalable. If more users are added to the system then all that has to be done is to use a more powerful WebServer or cluster many WebServers together.

2.2.5 What Programming Language to Use

Obviously, for a page to be dynamic we have to do some sort of programming. The advantage of ASP.Net is that it is language neutral. All that is required is a language which is .Net compatible. A language is defined as .Net compatible if can be compiled into Microsoft Intermediate Language (MSIL). This intermediate language can then be run in any system which has a .Net runtime. For example: on Windows and Unix. Many languages now are available that will work with .Net: managed C++, C#,
Perl, Eiffel etc.. Programming the web pages can be done in any of these languages. I decided to use C# for many reasons:

- C# is very like Java. Its syntax is very similar and since I know Java quite well then this made the change much easier.

- Visual Studio.Net comes with an environment for programming in C#. This means that all C# programming is integrated within the environment. For example: Help, debugging etc...

- Examples of code in most of the Software Development Kits (SDKs) that I use are in C#. This made programming the Speech Controls etc.. much easier since I could understand and use the code more easily.

An excellent book which explains how to code ASP.NET using C# is *Teach Yourself C# Web Programming* [9]. This book explains how to program using C#, how to create and access Web Services and how to access databases.

## 2.3 Web Services

### 2.3.1 What are Web Services?

A web service is a new buzzword in the computer industry. According to W3 a Web service is a “software system identified by a URL, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.” [16]

In other words a web service provides developers with publicly accessible methods which can be invoked from across a network or the internet. This can provide developers with very useful tools from many different sources. For example: if imdb.org,
a movie resource providing information about films, provided web services that pro-
vided access to its database on films then someone could create an application, be it
a web site or a standalone program, which could present the data in a specific form.
Developers can call web services from many different sources to create their own ap-
plications. Simply put, if someone has already solved your problem why reinvent the
wheel?

2.3.2 Why I used a Web Service

As previously said web services are one of the latest buzzwords of the IT industry.
This does not mean, however, that they are used in this project for the sole reason
of demonstrating their use. Web services were very helpful when building the the
project.

One of the main aims of my project was to make it as scalable as possible - from
one user to many users. Web services help attain this goal. Normally the web services
run on the same web server as the main code of the project. With few users there
is no need to use many web servers. However, when the system is used with many
users then the web server could easily be swamped by user requests to display pages.
Without using web services then one would have to set up complex clusters of servers.
If web services are used then the services can simply be put on separate machines.
The only configuration that needs to be changed are the URLs of these services. Of
course if even more users are added then clusters can be used for each web service.
For example: if the Video web service is in heavy use, then a web server can be
devoted to it exclusively. On the other hand, if it is found out that both Services and
MusicServices are rarely used, then they can let run on one server. This makes the
system scalable to its requirements.

When the user calls up a web page, which is, in fact, what the user interface
is, then the web server invokes the code on the webpage. This code provides the
link between the interface and the data. It also provides a separation between the data and implementation of the interface. This is important because it means that although the data might not change much over time the interface used to present the data might undergo many changes. By separating the two we can reduce the amount of development time taken to change or improve the interface whilst not having to deal with changing the original data code. This can lead to great savings in time and money. It also makes debugging the code easier. For example: if everything worked previously and now that changes have been made to the interface and something doesn’t work, then the problem cannot lie with the web service but with the interface. On the other hand if we want to change how the data is accessed and stored then only the web service has to change, not the interface. The interface will be oblivious to the changes. For example: if we move the database to another server then just the URL of the database in the web service has to be changed. The developer of the interface doesn’t have to know or have to update the interface to take these changes into account.

A important word now in the computer industry is security. How secure is the data on your system? By using a web service I have introduced a middle man between the user and the data (which lies in a database). The user does not have direct access to the data. Instead, he or she access the web service which in turn accesses the database. This is very important because a direct connection to the database could lead to attempts to try to break into the database. To further strengthen security the database could be protected behind a firewall. Only the web service would have access to it.
2.4 Database

2.4.1 What is a Database?

A database is a collection of data that is organised so that its contents can easily be accessed, managed and updated. A database can store data but, more importantly, a database can then query and manipulate this data.

2.4.2 Why I used a Database?

A central concept of this project is that every user should have their own individual profile. Originally I thought about storing this information in a text file but it soon became clear that this would cause problems. It is quite easy to add and delete data in a text file. However, if we want to update data or add data to a certain part of a text file then this quite be quite difficult.

Obviously I wanted to store data but I also wanted to do something with this data, not just store it and view it later. I wanted to manipulate the data and search it. This is far from trivial in a text file.

My project assumes not just one user but many. When the program is accessing data for one user then it locks the file. No other program can access it. This is a feature of nearly all filesystems and is very difficult to get around. If a second user is trying to access the file then access will either fail or be delayed. For example: Someone watching a video might not find it acceptable that they cannot change videos because someone else is buying a pay-per-view video.

A step up from this would be to store the data in XML files. This would have the advantage that XML describes the data it stores which could come in useful. However, this would still be a text file and would have the same disadvantages inherent in text files.
The only option left was to use a database. I was initially wary of using a database because databases can be quite complex. However, the advantages of databases outweighed their disadvantages.

**Advantages of databases:**

- A database is designed for a multiuser environment. Therefore, the data can be accessed simultaneously by many users.

- A very powerful feature of databases is that they can query the data. This means that they can retrieve the raw data, process it using some rules and return the results. This is a very powerful feature and one which I made heavy use of it.

- Databases are also very scalable. One of the main aims of my project was to make sure that the system can be scaled up as more users are added. Databases provide this feature.

2.4.3 Access or SQL Server

Once I decided that I needed to use a database the next decision I had to make was which database system to use. ASP.NET can connect to many different databases but particularly to Microsoft Access and Microsoft SQL Server. Many people are familiar with Access because it is bundled with Microsoft Office. I decided not to use this since it does not scale very well [14]. That left only SQL Server. I decided to use this over other databases such as MySQL since it fits in better with .Net [12].

A big problem with SQL Server is that it's not free (it's a very big commercial program) but they do provide a 120 day trial of the program on their website \(^1\). Once over this hurdle the next step was to set up the database. SQL Server is an extremely

---

\(^1\)Available at http://www.microsoft.com/sqlserver
CHAPTER 2. TECHNOLOGIES

complex program and I had to spend much time trying to set up databases, tables etc.. Much of this had to be done manually because the wizards didn’t either suit the task or they went into too much detail. However, once I had used the system for a while it became much easier to do things and I now believe that the time I spent at the beginning exploring the system led to huge productivity increases at the end.

2.4.4 What is SQL?

SQL stands for “Structured Query Language”. It allows a user to question or query a database. It is a query language. It works on relational databases which store data in tables which a related to one another. There are many different versions of SQL but most are very similar. In this project I will use Microsoft’s Transact-SQL. This has some extensions which will be needed later in the project.

2.4.5 Plain SQL or Stored Procedures

Initially, I accessed the data directly from the database tables using SQL commands. I wrote the SQL commands as strings and then these were invoked on the Server. However, debugging these strings was close to impossible. It was difficult to find out where problems lay. If the SQL command didn’t invoke properly was it because the string was not correctly formatted, the object was not set up correctly, the SQL syntax itself was wrong etc.. This led to much much frustration at the beginning. There had to be a better way.

A feature of SQL Server is that it can store the SQL statements in the Server as functions or methods which can then be invoked by outside sources. What is important to realise is that now outside sources do not directly access the database tables but instead go through a middle layer - the stored procedures. This has numerous advantages.
• **Speed:** The stored SQL are compiled by the Database and this compiled form is used at runtime. This leads to speed gains over plain SQL.

• **Debugging:** By creating the SQL statements within the Server I could use the Server’s debugging tools on the statements. This ensure that the SQL is valid. Figure 4.25 in the next chapter shows the debugging tool. After entering the SQL code into the dialog box, the SQL can be checked using the button “Check Syntax”. This saved a lot of time. There are also tools available to optimise the SQL, to check processing time etc.. but I didn’t make use of them since my SQL was not very complex.

• **Consistency:** Within the procedures one can write code to ensure that all actions will leave the database in a consistent state. By this I mean that if something is changed then other changes can be made in the database to ensure that these new changes are reflected throughout the database. IE: It is a form of housekeeping on the database. For example: When a User is being deleted from a database, all their entries in other tables are deleted at the same time.

• **Security:** Every user has to be given execute permissions on the stored procedures before they can be invoked. This has obvious security implications because we can stop certain users from doing certain things to the database while granting permission to others.

• **Flexibility:** I often had to change the database structure when I added in new features to the program. These changes ranged from adding new fields to a table to creating whole new tables. I only had to update the stored procedures to accommodate these changes, not classes and programs. This meant that the database could radically change while my services saw no change.
2.5 Media Services

2.5.1 Introduction

A Media Server streams media to a number of clients. It sounds like a web server, which does a similar job. However, a Media Server has many advantages over web servers when it comes to streaming media:

- A Media Server is designed to stream media, be it an audio or video file, from one computer to another over a network connection. This means that it is optimised to stream the media in the most efficient way possible.

- A Media Server allows you to stream from a particular point within the media. For example: from the 7th minute of a video. This would not be possible if you used a Web Server.

- Media can be arranged in playlists on a Media Server. These can be arranged in a specific way or can loop. The Media Server allows a lot of control over how the media is played.

These lead me to think about using a Media Server over a Web Server, but which Media Server should I use?

2.5.2 Windows Media Services 9

Most of my project was created using Microsoft development tools, especially Visual Studio.Net which was an excellent tool. The advantage of using tools from one provider is that they tools are normally integrated well with one another.

AVI files do not stream well. No Media Server streams avi files, instead a special format is used. You always have to convert to this format. Since I was using Windows PCs as clients then the obvious media type to use would be Microsoft Media, both
the wma (audio file format) and wmv (video file format). To stream these files you can use Media Server on Windows 2003.

**Advantages of Windows Media Services:**

1. WMS (Windows Media Services) was designed to be scalable. This fits in well with one of the main aims of my project.

2. WMS provides fast streaming. Unlike most Media Servers you do not have to wait while the client buffers data before playing the media. This is very important since the user would not like to wait a few seconds every time they change channel etc...

3. WMS provides Live and On-Demand streaming. When streaming television channels then the stream would have to be a broadcast stream. However, when showing videos or playing music then the media is streamed on-demand.

4. WMS provides support for advertising. Although I did not include this in my project I did try to show the commercial possibilities of my project. Obviously, advertising is very important in a commercial environment. WMS allows you to place advertising practically anywhere. For example: At the beginning of a file or to create advertising breaks.

5. WMS can be programmed. Although it provides a GUI interface WMS also provides a programmable interface. I use this later in a configuration tool.

As can be seen WMS provides a lot of benefits when streaming media. I used many of the features of the server. These would not have been possible using a Web server. Also, all Windows PCs are bundled with Windows Media Player which means that a media player does not have to be installed on each client. This can save a considerable amount of time since an administrator does not have to install and configure a media player on every client. It is already installed and configured (by Windows).
2.6 Summary

The aim of this chapter was to describe the various technologies used. I did not want to use the technologies for the sake of it. Instead, I had reasons for using each technology.

The graphical interface had to be dynamic and viewable in a web browser. I used ASP.Net to create the GUI of the project. It also fits in with the speech technologies described in Chapter 3.

To make the project scalable I used web services. These also increased the security of the database since the web page does not directly access the database but instead calls a web service which in turn accesses the database.

Using databases I could allow many users to use the system. All their settings are stored here. Since it is a database it can easily be configured to scale up as more users are added to the system. Using databases also allowed security constraints to be added to the project. The database tables, where all the data is stored, is not accessed directly from outside the database. Instead, calls are made to stored procedures which in turn query the database. This adds to the security of the system.

A media server is used since it optimised for handling audio and video files. It can also be accessed programmatically. This is very useful if one has to access the server from a program. Since a media server is designed from the ground up to be scalable, this fit in nicely with the aim of my project that it should be scalable.
Chapter 3

Speech Technology

3.1 Introduction

Earlier, in section 1.3, I gave reasons why speech control is a fundamental aim of my project. In order to do this some sort of speech programming tools were needed. Luckily, Windows provides built-in speech recognition and synthesis services. Microsoft are developing speech tools which can take advantage of these services. The tools used are Microsoft Speech Controls which are in beta. Since these tools are in beta there are many problems which cropped while creating the system which will be detailed later.

The tools integrate into a WebServer to provide web pages which can be speech controlled. The tools are fully programmable. On the client side all that is needed is a plug-in for Internet Explorer. It could work with other browsers as long as they support ActiveX plugins. At the moment this has to be installed manually, but I assume that when the tools develop further then the installation should be an automatic when the user visits the webpage.
Figure 3.1: Speech Controls

Figure 3.1 shows how the Speech controls work with the project.

1. An event is caused on the Server which causes control to be passed to the Speech Control.

2. The Control may give a prompt to the user. The User utters something. The utterance is evaluated against a grammar within the Control. If the utterance is not recognised then the control loops giving the user another chance to say something recognisable.

3. If, what the user says is deemed valid, then control is passed over to a Semantic Map. This maps the semantics of the sentence to a control on the webpage.

How control is passed to the speech controls will be describe in section 3.2. Section 3.4 will describe the different types of grammars and what the grammars output. In section 3.5 I will discuss how the control deals with the semantics of the input sentences and how this links in with the server.
3.2 Voice-only vs Multimodal

A voice-controlled web page can be described as being either voice-only or multimodal. What speech controls are used and the flow of execution of the page depend on what mode you use. The mode of the page is set by appending the following codes to the end of the page URL: ?mode=voiceonly, in the case of a voice only site, or ?mode=multimodal, in the case of multimodal mode.

A voice-only application does not provide a Graphic User Interface (GUI) to the user. It is typically used over a telephone where the only means of communication is verbal. The computer asks questions which the user answers. Any feedback is provided verbally.

A multimodal application provides the user with a GUI. The user can use the application using a keyboard or by using their voice. This is sometimes called “tap-and-talk”. Feedback is usually visual. To use speech recognition the user has to click on a button or some other control and then speech recognition takes over. The important fact here is that a control triggers speech control.

In this project neither voice-only or multimodal modes were suitable. What was needed is a Visual interface which is navigable (only) by voice. The Multimodal mode only allows voice control AFTER clicking on a control. What I wanted was complete voice control without clicking on anything. On the other hand, voice-only does not use a GUI, instead it prefers to be used over systems which do not or cannot provide visual information. For example, over a telephone. It does, however, allow complete control using speech. My project uses a mixture of the two: A GUI which is navigable by speech without using user-clickable controls. As far as the controls are concerned they are in voice-only mode.
3.3 Speech Controls

The Speech SDK [8] provides numerous speech controls. Speech Controls are a collection of ASP.NET controls designed to speech-enable Web applications. Examples of these are the Command control and the QA (Question/Answer) control. Nearly every control can be used in either voice-only or multimodal mode. However, the properties of the control changes depending on the mode. For example: in multimodal mode a prompt is not given. (What Prompts will be used for will be described later in section 4.2.3)

Many of the controls deal with numbers, dates, telephone calls etc.. None of these suited the purposes of the project. This left only two which could be used: a Command control and a QA control. At the start both controls seemed to be quite similar, if not identical. They both allow dynamic grammars to be added. They both can be used to navigate a page. However, in the end I used the QA control.

I decided to use QA controls for the following reason. A QA control can be used in multimodal mode (described in section 3.2), whereas a Command control cannot. This makes the QA control more flexible since it can be easily changed to suit a multimodal mode, if needed. A command control does not work in a multimodal mode.

One of the features of this control is that allows the use of prompts. A prompt is a question or information spoken by a speech application. In this project it has three purposes:

1. It asks the user questions. For example: "What track would you like to play?". When someone hears this they will be prompted to respond. This can make things easier for the user because they will be able to formulate an answer based on the context of the question.
2. One of the aims of the project is that it should be more accessible to users with visual impairments. Prompts can be used to describe the screen to the user so that the system can be used without even looking at the screen.

3. It can inform the user about the current state of the application. For example, tell the user that the video is paused.

3.4 Grammars

3.4.1 Introduction

It’s all very well having a computer convert spoken text into text on a screen. The question is what we do with this input string. We have to extract a meaning from the string and do something depending on that meaning. There are some problems with this:

- What does the string actually mean?

- The string might just be garble (in the context where it is said). For example: If we get a string such as ”the big big”. This might have absolutely no meaning in the current context. We have to try to allow for this by asking the User to repeat what was just said.

- The string might not be valid in the current context. If the User says ”I want to watch Teletubbies” when they are in the Music menu. Although this sounds like a good idea it would be extremely difficult to implement because every page would have to know about the features of every other page.

- The string might be valid in the current context but there could be ambiguity.
For this we use a grammar. “A grammar is a formal definition of the syntactic structure of a language, normally given in terms of production rules which specify the order or constituents and their sub-constituents in a sentence” [4]. This can be implemented in a computer. However, in this project, the computer’s grammar will not be as sophisticated as a human’s due to processing requirements, grammatical complexity etc..

In the Speech SDK [8] you have to provide each speech control with a grammar. The grammar decides what the speech SDK will accept as input and what it should reject. If an utterance is rejected then the control will ask again until it gets a valid reply. The control can use two different types of grammars - static grammars and dynamic grammars.

3.4.2 Static Grammars

This type of grammar is a so-called static grammar because it is created by a developer during the development phase. It is then compiled and used during runtime. It cannot be changed while it is being run. Initially, I thought that this was the only type of grammar you could use.

This type of grammar is very useful in certain circumstances. It is used to build the menu system of the project. For example: ”Go to main menu” or ”Buy this video”. These could not be created dynamically by a computer.
Figure 3.2 shows the tool provided by Visual Studio to visually create a grammar. The various elements have to be dragged onto the screen and ordered correctly. Each element can then be given a value (word or phrase) which it will accept. Additionally, each element can be given one or many semantic values which are useful later. An interesting point to note here is grammars in the Speech SDK [8] are case-dependent. When a user utters a sentence then the sentence is converted into text with the first word capitalised and all subsequent words begin with small letters. For example: When the user utters "I want to watch videos" then this will be converted to "I want to watch videos". Likewise, when the user says "Watch Video". This explains why
both “Watch” and “watch” have to be used in the grammar.

Figure 3.3: Sample Sentence
Once a grammar has been created and saved the grammar can be tested to see what it will accept and reject an input. Figure 3.3 shows a sample sentence which the grammar accepts. Note how a ”path” in blue is given along the grammar. At the bottom of the screen the test returns whether it succeeded or not, along with any semantics which are part of the grammar. In this example, the semantic tag is /SML/Value and its semantic value is Videos.

### 3.4.3 Dynamic Grammars

This type of grammar is created or built up by the application during execution. This means that grammatical rules can be created on-the-fly. When I first started this project I thought that this was impossible. However, it later became clear that it is a feature of the Speech SDK [8], the details of which are not described in the documentation.

```xml
<grammar xml:lang="en-US" tag-format="semantics-ms/1.0" version="1.0" root="Rule1" mode="voice" xmlns="http://www.w3.org/2001/06/grammar">
  <rule id="Rule1" scope="private">
  </rule>
</grammar>
```

Figure 3.4: Basic Grammar

To see how to build up a dynamic grammar you have to first look at how a grammar is coded. Figure 3.4 shows the XML code needed to generate an empty grammar. Using this code we can then create a grammar which will accept any input which we want. All that has to be done is add in tags such as `<item> </item>` . A computer program can use this to build up the grammar. Hence, it becomes a dynamic grammar.
CHAPTER 3. SPEECH TECHNOLOGY

3.5 Semantics

3.5.1 Introduction

Once a speech control was chosen the next step was to link the input that the control gets to the application. If the User says "I want to watch RTE1" what does this mean? This is where the semantic tags in the grammar come into play.

In figure 3.5 you will see the Input section of the QA control. This deals with two main areas: Grammars and Semantics. At the top there is a section which allows you to add a static grammar to the control. In this control there is no static grammar. The grammar will be created dynamically. Below this is the semantics section.

Figure 3.5: QA Input Section
Each semantic value has five properties. The first property, *Semantic Item*, can be thought of as a short name for the value. It will be used later by the Semantic Map. The next property, *XPath Trigger*, tells the control the trigger’s name. The trigger is the semantic tag within the grammar. In figure 3.5 the tag is `/SML/Value`. When the user speaks the spoken utterance is converted to text. Then the control checks to see if it is grammatically correct. If it is grammatically correct then the parser will return the semantic value of the sentence. This comes from the tags in the grammar which I described earlier in section 3.4.3. The next property is not important but it can be used to normalise the input. The recognition confidence level is given by the control when it recognises an utterance. This level is given a floating point value between 0 and 1. The final two properties set the thresholds which allow the control to either accept the input or not depending on the confidence level. They are set to 0 in this control because I did not want the control to be too specific about the utterances it accepts because many people will use these system and their voices all vary. When the utterance is accepted then the speech control finishes it execution by outputting the semantic tag of the utterance. This tag is then picked up by a *Semantic Map*.

### 3.5.2 Semantic Map

Once the speech control has evaluated the sentence it outputs the semantic tag associated with the sentence. This is where the *Semantic Item* label is important. The control outputs the tag and the *Semantic Item* which is picked up by another control, the *Semantic Map*.

The *Semantic Map* maps the semantics of the sentence to an event which ASP.Net can then process. Figure 3.5 showed the semantic section of a QA control. If the control parses the sentence and finds the grammar outputs the semantic tag or trigger `/SML/Value` then this will be given a *Semantic Item* value of *Value*. Control is then
passed to the *Semantic Map*.

![Semantic Map properties](image1.png)

**Figure 3.6: Semantic Map**

In the *Semantic Map* in figure 3.6 we can see a *Semantic Item* called *Value* with its appropriate properties. When a *Semantic Item* whose value is *Value* is found, then the *Semantic Map* will send the value of the Semantic Trigger, `/SML/Value`, to `TextBox1.Text`. The *Semantic Map* also allows client functions within the client’s browser to be evaluated. I did not use this feature. Once the *Semantic Map* is finished `TextBox1.Text` will contain the semantics of what the user said. This is then used by the application to play music etc..
3.6 Summary

The main aim of the project is that the graphical interface should be navigable by voice. Using the speech controls of the Speech SDK made this possible. To use the speech controls is quite an involved process. In this chapter I gave an outline of what has to be done in order to speech-enable a web page.

The mode of the page has to first of all be set before any recognition can be done. This mode also defines what speech controls can be used. I gave my reasons as to why I had to choose a QA speech control which converts the User’s speech to text. The control would not be able to do this without grammars. The two types of grammars used, static and dynamic, were explained to show the differences between them and the situations when they could be used. Finally, I described how the text obtained from the speech controls and grammars was converted to actions on the web page by using a Semantic Map.
Chapter 4

Implementation

4.1 Introduction

In this section I will describe how I used the technologies described in the previous two chapters to create my project. The main part of the Project can be broken into five stages: Create the User Interface, Call Web Service, Query Database and Media Services. To make configuration of the project easy I created the configuration tools described in the final section of this chapter.

4.2 Stage 1: Create User Interface

4.2.1 Introduction

The most important thing, as far as the User is concerned, is the interface of the project. This project’s interface is divided into two: the visual interface and the speech interface. The visual interface is built up using server-side code which ASP.Net compiles into HTML for the client. The Speech controls also use server-side code but most of the code generated is SALT which can then be used by the speech controls on the client. Both interfaces can also be coded on the client side by directly editing the pre-created HTML file.
CHAPTER 4. IMPLEMENTATION

It should not be forgotten that ASP.Net divides code into two parts: The client-side code which is the HTML file, eg: index.html, and the server-side code which has the same filename as the client file but with the extension of the programming language, eg: index.html.cs. These two files are merged together after the server-side code is compiled into HTML.

4.2.2 Server Controls

When I started to create the look or interface of the web site I noticed that a lot of my code remained the same from page to page. This was to be expected if a consistent interface was to be created. But the problems started when I tried to make a change to the interface. The change had to be reflected throughout the whole site. This would lead to problems because sometimes I would forget to make the changes on one page. This created both interface inconsistencies and errors on the webpage. Luckily, ASP.Net has a solution to this problem: Server Controls.

According to Microsoft “server controls are components that run on the server and encapsulate user-interface and other related functionality” [3]. They have two advantages:

1. Code changes to a control are reflected in each instance where the control is used. IE: Over the whole web site of this project. This made it easy to build up the site and reduced bugs and inconsistencies within the interface.

2. These controls can be used natively within Visual Studio.Net which allows them to be dragged on the web page and its methods and properties can be invoked and queried using Visual Studio.Net tools.

In the project there are six such controls. All of the controls have a base set of methods and properties which create the basic feel of the interface. All the other
controls inherit from this base control. The advantage of using inheritance is that the Controls can provide a consistent interface to the developer and use common methods to achieve common aims. For example: the `select()` method described earlier.

**MenuControl**

This is the base Control in my hierarchy of server controls. This provides the basic look of my whole project. The name of the control is an apt description of what this control does - it shows a menu structure. Most of the interface is created when the `onInit()` method is invoked. I have called my custom method, `InitializeComponent()`, from here. All that this does is set up the colour scheme used by the control as well as providing a heading for the web page. One in five Irish households now have SKY Digital [13] which provides a very intuitive EPG (Electronic Program Guide) [15]. My colour scheme is based on this.

Since the control shows a list of items in a menu then we would have to provide a method to add items to the menu. This is done using the `addMenuItem()` method. This method, which I created, simply adds a new Label to the page with a caption.

One of the more important methods of the control, which will be used extensively later on, are the `select()` and `unselect()` methods. I created these methods to allow a developer to highlight certain items on the menu or to deselect other items. This is quite important because most people are used to seeing their choices displayed more prominently in menu structures. The selected item is displayed in a yellow colour. To unselect it the colour is set back to blue.

Since this project uses Text-To-Speech as well as Speech Recognition I decided early on to make the system friendlier to those with visual impairments. The Text-To-Speech system simply takes in a string and then reads this string out loud. Using this system I made it possible to describe each screen using a method called `describe()`. In this control this method creates a text description of the Menu screen. This can
then be used by the speech controls. All this method does is enumerate through each Label in the menu and add its caption to the string. The resulting string is a textual description of the menu items.

**TracksControl**

This Control displays a list of tracks to the User. It shows the track number along with the track title on the screen. It inherits from *MenuControl*. Since there are some differences between these two controls some of the methods in *MenuControl* had to be overridden. At the same time new methods were added to deal with the new features of this control.

This control is first of all constructed using the constructor of *MenuControl*. This gives the control the basic look-and-feel of the project. Only then are new elements added to the Screen. For example: Labels showing *Track Number* and *Track Title*.

Unlike *MenuControl* each item on the screen is actually made up of two Labels - the track number and the track title. This would cause problems with the *select()* and *unselect()* methods since they assume just one Label. They had to be changed to reflect this fact. *describe()* was also changed to take this into account.

When a new track is to be added to the page then the *addTitles()* method is invoked. I created this method which just adds two labels to the screen showing this new information. The new title is added below the previous title.

**MusicControl**

This control is used to show CDs or the titles of music collections. It only differs from *MusicControl* in that it has to account for this fact. For example: the headings and labels are oriented towards CDs, not tracks. To do this I created a method called *addCdNumAndTitle()* which is used to add this information to the page.
TeleControl

To show television channels on the screen the *TeleControl* is used. Whereas, the Music and Track-Controls did not have to show a media player, this control will have to show television in a preview screen. This had to be taken into account when positioning elements in this control.

This control displays its data in two different parts of the screen. In the top left hand corner of the screen details about the current television programs will be shown. This includes the name of the program, when the program begins or has began, the channel name and the description of the program. This leaves room on the right side to show a preview screen. Below this the list of channels, along with their programs, are shown.

To add channels to the screen, along with their programs, the method *addChannels()* has to be used. Unlike the other add methods which I created this can take in an arbitrary number of programs. However, for the purposes of the project I have limited this to two programs because of time constraints on the project. IE: the interface shows Now&Next program information. The list of channels and programs can be dynamically added to the screen. For example, if there is updated list of programs for a channel then they can be added to the page. This depends on whether the application can provide the updated list of programs. In this project that has not been implemented but it could be added.

As described earlier the top left corner of the screen shows the information of the current program. I had to create methods which could access and change this information. This is done by using accessor methods such as *setProgramName()* , *setDescription()* , *setTime()* and *setChannel()* . Obviously, I created appropriate *get* methods to get the current displayed information.

This control also allows the channel names to be highlighted, separate to the
programs. This is to allow either the programme shown now or the next programme to be highlighted, as shown in figure 4.1.

**VideoControl**

This control is very similar to the Television control. Instead of showing channels it shows a list of videos. This means that the information section to the left side has to show the appropriate video information. For example: the cost of the video and the rating of the video.

The `describe()` method here is more involved than the `describe()` methods of other controls. The method reads out the information about the current program and then lists out the videos on-screen. This would be quite useful for people who cannot see the information about the current video.
**ConfirmControl**

One of the features which I added to the project is a PPV section. It would not be a good idea to let a user simply say “buy video” and then charge them for the video. They could do this by mistake. To prevent this problem from arising the User should be presented with a confirmation dialog or page. This control does this.

The developer uses methods to display the relevant information about the Video to the User so that they can be sure that they want to buy the video. The developer can show the cost of the video using `setCost()` which I have provided. Obviously, the User has to know how much credit they have in order to make a decision. To add this and any other pertinent information I created `addInfo()`.

**Control Designers**

![Figure 4.2: Toolbar with Custom Server Controls](image)

Each of the above controls has a class which inherits from Control Designer. This
is used when adding the control to the Visual Studio toolbox, as shown in figure 4.2. It simply allows the control to be dragged onto the canvas and its height, width etc. to be changed using a mouse and/or keyboard.

4.2.3 Main Pages

There are three “main” menus in this project: the first menu which displays the list of users who can use the system; the main menu of each individual user; and a menu which shows the new services available to each user

User Selection Page - Startup.aspx

![User Selection]

Figure 4.3: Select the User

This page shows a list of users (figure 4.3). This section shows the two important parts of every web page in the system: creating grammars and how the project works out how what to do with the input.
CHAPTER 4. IMPLEMENTATION

Like every web page of the project this page uses Server Controls. It uses the MainMenu server control to build up the interface of the page. As described earlier this control will provide the template for the page. This template will be built up using data retrieved from a web service.

```java
service = new localhost1.Services();
users = service.listUsers();
```

Figure 4.4: Code to get a list of Users

The web service, Services, will be described in detail in Section 4.3. The code used to create this web service and to get the list of users is shown in figure 4.4. The web service is initialised by creating a new instance of it. The listUsers() method of Services is then invoked which returns a list of users.

We now have the data needed to build up the page. First of all each user name is added to the menu using the addMenuItem() of the server control. This will show the list of users on the web page. After the menu has been built up we have to create the grammar which will be used by the QA speech control. I created a method called createGrammar() to build up the grammar.

```java
grammar += "<item>" + titles[i];
graham += "<tag>$User=" + titles[i] + "</tag>";
graham += "</item>";
```

Figure 4.5: Code for Semantic Tag

Dynamic Grammars were described in Section 3.4.3. All that needs to be defined now are the semantic tags of the grammar. Figure 4.5 shows the code needed to create the semantic tags. The user name is put within the <item> </item> tag. This is what the speech control recognises within a spoken utterance. The second line defines the semantic tag. The tag has been named $User. When the speech control recognises the word it creates a semantic tag called /SML/User with the
value \( titles[i] \) (the user name). When all the tags of every user name have been created, the grammar is added to the QA Control by using the \( \text{Reco.Add()} \) method provided by the QA control.

The User interface, both Visual and Spoken, has now been created. What happens when the user says something? When the user says something, assuming it is accepted by the grammar (if not then the control asks for a response again), then the value of the semantic tag is put into a \( TextBox \). This is a control on the web page. When the value of a \( TextBox \) changes then an event, \( \text{TextChanged()} \), is raised which can be caught by the server. I customised this method to perform an action on the screen.

In this page there are two types of semantic tags which can be put into the \( TextBox \):

1. The User might request a description of the page. This will result in the semantic value \( Describe \).

2. The user name uttered by the user.

If the User wants a description of the web page then the \( \text{describe()} \) method is invoked on the Server control. This will give a string description of the page. By setting the \( InlinePrompt \) property of the QA control to this string, the control can given a spoken description of the page.

If the User says a user name then we use the \( Services \) web service again. We use the method \( \text{getUserID()} \) provided by this web service to give us the user id of the User. This is very important because this is used throughout the rest of the menus to show all the user’s videos, tracks, channels etc.. To do this we store this id in a \( Session \) variable which is stored as part of the User’s browsing session. From here we transfer the User to the next page - \( Index.aspx \).
Main Menu Page - Index.aspx

This page shows the main menu for each user (figure 4.6). It is fully dynamic. By this I mean that the contents of the page can vary from user to user. This uses the Session variable described in the previous section. It also uses the Services web service to display the User’s menu items. Each menu item can be thought of as a service.

The type of Grammar used by the speech control in this page is a static grammar. This is quite important because a satisfactory dynamic grammar cannot be built-up here. How can you get the server to turn a keyword into a phrase? For example: From a keyword Music create phrases like “Do you want to listen to tracks?”, “Do you want to listen to CDs?” etc.. So these phrases have to be created beforehand. The web service provides the URLs of the grammars which are then loaded by the QA control.

Once the speech control completes its actions it passes the semantic value of the sentence to the TextBox control. Like in the User Menu, the user can request a
description of the menu. Otherwise, depending on what menu item the user selects, the system chooses the correct page URL. By using the web service we can get the URL of the page whose user id we already know and whose description is the menu item.

**Services Menu Page - Services.aspx**

![New Services](image)

Figure 4.7: Show New Services

When a new service is to be added to the system then it is first added to this page (figure 4.7). It is not initially added to the main menu. The user must decide whether they want this added to their own menu or not. I decided to do this for two reasons:

1. The Main menu is a personalised menu for the user. The user should have the choice of what they want on this menu and what they do not need.

2. It is conceivable that there could be a charge for each service. This feature could easily be added to this page so the user would pay to add the new service to their menu.
Like most of the pages of the system this page uses a dynamic grammar to allow the user to select the services. Each item is added to the grammar. The page also uses a static grammar. This is the same grammar which will be used in most of the pages. It allows the user to go back to the previous page and go the main menu.

### 4.2.4 Music Pages

There are two pages in this section: a page to show the list of CDs (or more generally, to show music collections) and a page to show the list of tracks from the selected CD or collection.

**CD/Collection Page - Music.aspx**

![Figure 4.8: Shows a User’s CDs](image)

The page in figure 4.8 shows the list of CDs or the list of music collections. Unlike the previously described pages this page uses the *MusicControl* server control. The
list of all the music collections is added to the page using the `addCDNumAndTitle()` method of the control.

The User can select the CD using either the CD number or by the title of the CD. Both dynamic and static grammars are used here. A dynamic grammar is used when the user wants to select the title by the title name. If the user wants to select the CD by number then a static grammar is used. This is because it is more efficient to use static grammars because they are precompiled. It is easy to create a grammar of numbers at design time.

Like every page this page can be described verbally to the user. The server control has the `describe()` method which provides the text representation of the page. The user can also go back to the main menu from this page. If the user selects a CD then the `MusicService` web service is used to get the ID of the CD. We set the last played CD to be this CD (the reason for this will become apparent later). Finally, the server transfers over to the `tracks.aspx` page.

**Tracks Page - Tracks.aspx**

The tracks page in figure 4.9 lists the tracks of the CD which the user selected in the previous page. This page uses the `TracksControl` to display the list of tracks. This is a complicated page because, along with speech recognition, the page must also allow the user to listen to music.

How does the system know which CD the user has selected? Earlier, in the CD selection page, we set the last played CD to be the CD which the user selected. The tracks page uses this to list the tracks of the last played CD.

When the page initially loads no track has been played because `Session["Playing"]` has not been set yet. If this is the case then the first track is played. The first item of the list is also highlighted by calling the `select()` method of the `TracksControl`. 
There are two types of grammars used here, just like in the previous page. The dynamic grammar lets the user choose videos by the video title. A static grammar lets the user choose videos by number. Of course the system allows the user to go back to the previous page as well as go to the main menu. The third static grammar used lets the user control the tracks. For example: go to the next track, pause and resume the track.

To play a track the `play()` method is called. I created this method to allow a developer to easily play music files. This takes three parameters: the track number, the position within the track at which the track should be played and if the track should be played automatically (ie: whether it is paused or not). The HTML used to create the media player is generated by the `MediaPlayer` class. This class will be
described in detail in Section 4.5.2.

1. The User can choose the next track in the list.

2. The User can pause the track currently being played.

3. The User can play or resume the paused track.

4. The User can select a specific track from the list.

If the User wants to play the next track in the list the program gets the number of the currently being played track which is stored in the session variable, *Playing*. The next track number is either the current track number plus one or, if that number is greater than the number of tracks on screen, it is set to zero. The currently selected item on screen is deselected and the newly selected track is selected (or highlighted). The `play()` method is then called with the new track number.

```javascript
function SavePos(){
    if (this.Player1 != null)
        Tracks.Position1.value = this.Player1.controls.currentPositionString;
}
```

Figure 4.10: Client Code to get Track Position

When the user wants to pause a track then we need to get the position at which the track is currently at. This is done in the HTML at the user’s browser. Figure 4.10 shows the code needed to do this. It gets the current position of the track being played and puts in into the *Value* field of a hidden control called *Position1*. This is then sent to the server. This is then stored in a *Position* session variable. The `play()` method is called again except that the *playMode* of the track is set to 0 (it is paused).

If the track is paused then the user must be able to resume it. When the User says *Play* then the track can be resumed. If the value of the hidden *Textfield* is empty
then we look at the Position session variable. We use the play() method to play the track from the paused position.

Finally, if the user wants to select any track on the screen then they just have to say the name of the track. This grammar returns the number of the track. This is then used to play the track.

A question which should arise is whether this page is useful to those who cannot see it? IE: Can a visually-impaired person use this page? As with every page in this project this page makes use of the describe() method of the TracksControl server control. When such a user asks for a spoken description of the page then the output will be a list of all the music tracks on that page. This means that the visually-impaired user will now know what music tracks are available.

An important thing to note here is that the speech control will now consider itself to be in a completed stage. This means that it will accept no more input. For all intents and purposes it has finished listening. To get it to start to listen again we have to reset it. When the speech control is finished it sets a state variable in the SemanticMap. To reset the control we have to set all the Semantic states to empty. The speech control will then restart and will allow further speech input.

4.2.5 Video Pages

This is the largest collection of pages within the site. There are six pages in total. There are such a large number of pages because this aspect of the site has to deal with PPV movies, as well as free videos.
Video Type Page - Videos.aspx

![Figure 4.11: Select either Free or PPV Videos](image)

This is probably one of the simplest pages of the project (figure 4.11). There are no dynamic aspects to it. The purpose of this page is to allow the user to decide which type of movie do they want to watch: a free movie or a PPV movie.
Free Video Selection - FreeVideos.aspx

Figure 4.12: Show all the Videos which cost nothing

The page in 4.12 shows the list of videos which the user does not have to purchase. It uses the VideoControl server control to display the list of videos onscreen along with a more in-depth description of the video currently being played. Instead of a MusicService web service we use a VideoServices service to list the videos and get information on the videos.

When the page is first loaded the browser plays the first video in the list. This uses a similar play() method to that of the tracks.aspx page. However, the media being played in tracks.aspx were only audio files. In this page video files are being played. This meant that the size of the video screen had to be taken into account as well as whether the video is to be played full-screen or not. I changed my play() method to take this into account. We use the same MediaPlayer class as in the tracks.aspx
This can implement the HTML needed to display the video files correctly.

Unlike in tracks.aspx this page shows more in-depth information about the videos. This is displayed in the top-right corner of the page. The VideoControl provides methods to get and set the labels which give the User information about the video such as its title, rating, cost and a description. These are all taken from the database using the web service which will described in detail later.

The speech controls allow the user to select videos and to pause videos, just like in tracks.aspx. However, because we are now dealing with video media, we have to provide the user with the ability to watch the video in full-screen. A new static grammar is used to let the user manipulate the videos. As well as allowing the user to move around the videos it also lets the user watch a video in full screen or in the preview screen.

When the speech control hears that the user wants to watch a video in full screen then the semantic value it produces is fullScreen. When the TextBox returns this value to the server then the server sets a variable in the MediaClass which tells the class to produce the HTML code for a full-screen media player. Conversely, if the user wants the preview screen then the server will generate the appropriate code.

Since we now have to be able to tell whether a video is to be showed in full-screen or not, we have to have a way of storing the current state of the player’s size. This is implemented by using a Session variable, FullScreen. By setting this variable to whatever its appropriate value should be we can maintain this state as the page is refreshed.
Initial PPV Page - PPV.aspx

Figure 4.13: Choose between a Video List and Purchased Videos

PPV movies can be classified here into two groups - those which the user has purchased and those which the user can purchase. This page allows the user to select between the two types (figure 4.13). The user can look at the list of PPV movies currently being offered as well as the videos which they have already purchased. This makes it a simple page because we just have to have a grammar which can account for this. If the User wants to watch a list of their purchased videos then they can say something like “Go to Purchased Videos”. On the other hand if they want to see what’s available then they can ask to see the “Video List”.

Purchased Videos - PPVPurchased.aspx

![PPV Videos]

**Figure 4.14**: Shows all the Videos purchased by the User

This section will describe the page which shows the user their list of already purchased videos. Figure 4.14 shows this page. It is very similar to the *FreeVideos.aspx* page. The only difference is that it uses the web service to get the list of the videos which the user has purchased. If no videos were purchased then a message is displayed on screen informing the user that they have not purchased any videos. 24 hours after purchasing the video the video is deleted from the user’s list and they will have to buy it again. The exact details of this will be described later.
This is the page (figure 4.15) which allows the user to buy a video. It has the same look as a standard Video page because it also uses the VideoControl control. There are three main differences:

1. Each video has a price which is displayed when the user selects the video. The cost is displayed along with the additional information about the video in the right corner of the screen.

2. The preview screen can show either a preview of the film or a trailer of all the films. To make things simpler I have set it so that it shows a preview of all the
videos. By this I mean that it uses a broadcast stream, instead of an on-demand stream.

3. The user cannot pause the preview video.

The web service gives the cost of the video. By using the `getCost()` method of the `VideoControl` the cost can be displayed on screen. One URL is used to show the video screen. Since it is a broadcast stream there is no need to deal with the position of the video within the stream as this is impossible. A broadcast stream can be thought of as a television channel. You cannot move to a position within a television channel. It also means that the preview video cannot be paused because you cannot pause standard television programs. I decided to do this because there is no need to pause a preview video since they are only very short and will be repeated often.

The main aspect of this page is that it allows the user to see a list of videos and then purchase this video. A static grammar is used to do this. It contains a list of possible phrases which a user might say to purchased a video. For example: “Buy this video”. When the speech control hears this phrase it sets a session variable, `Buy`, to the currently selected video id and transfers the user to a confirmation screen. However, if the user has already purchased the video then they will be unable to buy it again until it expires.
Purchase Video - Purchase.aspx

Figure 4.16: Confirm Video Purchase

This screen (figure 4.16) functions as a confirmation screen to allow the user to decide if they really want to buy the video. Hence, it uses the ConfirmControl server control.

On the screen the user sees the name of the film and its cost. The current amount of credit the user has is displayed along side this. If the User wants to purchase the video then they have to say “Buy Video”. They can, of course, go back to the video list page or go to the main menu. If they buy the video then they will have 24 hours to watch it before it is deleted. The cost of the video is deducted from their credit amount.
4.2.6 Television Page

This page (figure 4.17) allows the user to watch tv. It uses the TeleControl server control to show the channels and their programs. Its interface is similar to the pages used to show Videos except that the list of videos is replaced by a list of channels and their corresponding programs.

Due to time constraints, a system to extract Now& Next Data was not implemented into the project. Instead, arrays were used to populate the program fields. Like every other page which shows a list, I created an add method to fill the list. In the case of a TeleControl that is addChannel(). When the page is first loaded then the in-depth program data is given for the the currently being played program on the first channel on the list.

The speech interface to this page is exactly like that of the video pages. To the left side is a video screen. The user can use this as a preview screen to watch the program.
Using voice commands they can watch the television channel in full screen. They can also choose to watch a specific channel or else the next channel. However, unlike in the video pages, the current channel can not be paused. This is because the video is coming from a broadcast stream which can not be paused. Pause functionality could be implemented if the stream is an on-demand stream. All that would need to be changed is the URL of the channel which is stored in the database.

4.3 Stage 2: Call Web Services

4.3.1 Introduction

The server created the User Interface in Stage One of the project. However, the interface needs to display data to the user. The previous stage can be thought of as showing a template to the user. This stage fills the template with data. To do this the server-side part of the web page calls web services which link the database to the webpage. Earlier I gave reasons as to why this should not done directly. Instead we should use web services. These web services contain a list of methods which I created to access the database. This section is split into three. First of all, what web services are there? Secondly, how do we call web services? Finally, how do we use these web services?

4.3.2 The Web Services

Three Web Services are used in this project:

1. Services: This Service deals with Users. I created methods to list Users, add Users, delete Users and update User details. It also provides methods which allow you to access user details, for example: the age of a user.
2. **MusicServices:** This service provides my custom methods which list CDs and their tracks. New CDs and tracks can be added to the Database. Information on those CDs and tracks within the database can also be accessed from here. If they have to be deleted then the appropriate methods are also provided.

3. **VideoServices:** This provides the largest amount of methods of the three services. It has to deal with Videos as well as Television channels. Within Videos there are PPV videos and free videos. Therefore, it must provide methods to list videos, buy videos, delete videos etc.. The appropriate methods are also provided to show details of television channels etc..

### 4.3.3 Invoke Web Services

Before using a web service it has to be located and referenced. Obviously, the locations of the various web services which we use are known to us. This means they only have to be referenced. In Visual Studio there is an *Add Web Reference* menu to do this. This allows you to browse the web service’s methods to make sure that the correct service has been located. When the service has been successfully added to the project then it will automatically be given a *namespace* which allows it to be referenced from within a programming environment.

Namespaces for the Web Services:

- **MusicServices:** localhost.MusicServices
- **Services:** localhost1.Servicess
- **VideoServices:** localhost2.VideoServices
4.3.4 How to Use Web Services

It would be tedious to go through all the methods of the services. Because of this, a list of all the methods is provided in the Appendix. Appendix B.1 shows the methods of Services, Appendix B.2 shows those of MusicServices and Appendix B.3 shows those of VideoServices. The methods themselves fall into four categories: Adding, updating, deleting and extracting information from/to the database. I have divided it up this way to show how to use SQL, in particular how to call stored procedures. The Stored Procedures themselves are described in Section 4.4.3.

Adding Information to Database

```csharp
sqlCommand1.CommandText = "addUser";
SqlParameter myParm = sqlCommand1.Parameters.Add("User", SqlDbType.VarChar, 50);
myParm.Value = userName;
myParm = sqlCommand1.Parameters.Add("Age", SqlDbType.Int, 4);
myParm.Value = age;
myParm = sqlCommand1.Parameters.Add("Credit", SqlDbType.Money, 8);
myParm.Value = credit;
sqlCommand1.CommandType = CommandType.StoredProcedure;
sqlConnection1.Open();
sqlCommand1.ExecuteNonQuery();
sqlCommand1.Parameters.Clear();
sqlConnection1.Close();
```

Figure 4.18: addUser Method

Figure 4.18 shows the code needed to add a User to the Database. In the first line we give an SqlCommand object the name of the stored procedure we want to call. You can also set this to be an SQL statement, if desired. We then have to add the parameters needed by the procedure. There are three in this example: userName, age and credit. The type of these parameters is very important along with the order in which they are added. Then we make sure that the SqlCommand knows
that it is dealing with a stored procedure. The database connection is then opened. Because we are not querying the database, but instead adding to it, we call the `ExecuteNonQuery()` method. Finally, the parameters are cleared and the connection is closed. All methods in this web service follow those series of steps if they are adding to the database.

**Updating Information to Database**

This looks exactly like adding information to the database. The only thing that changes is the name of the stored procedure which will carry out the update.

**Deleting Information to Database**

Again this is exactly like adding or updating information in the database with the difference being the name of the stored procedure.

**Getting Information From Database**

Getting information from the database is more complex than adding it. Figure 4.19 contains the code needed to get the User ID of a User by the user’s name. Setting up the `SqlCommand` is the same. The difference is that we are now querying the database. This means that instead of using `ExecuteNonQuery()` we use `ExecuteQuery()`. Because we expect to receive data back we have to use a `DataReader`. Data is read according to its type. In figure 4.19 the type which we expect the user’s id to be is `int`. The query returns a table which has only one column. This column is column 0. So we are getting one value from a single table. After the data is read the connection is closed and the user’s id is returned by the web service.
sqlCommand1.CommandText = "getUserId";
SqlParameter myParm = sqlCommand1.Parameters.Add("UserName",
SqlDbType.VarChar, 50);
myParm.Value = user_name;
sqlCommand1.CommandType = CommandType.StoredProcedure;
sqlConnection1.Open();
rdr = sqlCommand1.ExecuteReader();
while (rdr.Read())
    user_id = rdr.GetInt32(0);
sqlCommand1.Parameters.Clear();
sqlConnection1.Close();
return user_id;

Figure 4.19: Get the User’s ID Number

4.4 Stage 3: Query Database

4.4.1 Introduction

A major part of my project is the database and the connectivity issues associated
with it. It is here that all the user data is stored. EG: the location of the various
media URLs, the list of users etc.. For the purposes of the project the database
should not be considered as simply one entity, but two. The database consists of
stored procedures and the database tables. The database tables are not viewable
from outside the database, whereas the stored procedures are. The reasons for this
are given in the previous chapter. To query the database one must first call the stored
procedures which in turn query the database.

4.4.2 Database Tables

All of the data is stored within tables in the database. In this project there are 12
tables. I will first of all describe the basic tables. These tables are then referenced by
the more complex tables which I will describe later.
CHAPTER 4. IMPLEMENTATION

Users

This table stores the details of each user. It stores their ID, their name, how much credit they have and the age of the user.

Each field has a type which allows certain values to be stored. Obviously, the amount of credit a user can store should be of type money. Their age should be an int. The length of a person’s name varies from person to person. To incorporate this fact within the database we use the varchar type. This allows a variable number of characters to be stored. For the purposes of this project I assume that no one has a name of more that 50 characters. Typing is very important because it is used later when we invoke the stored procedures since parameters have to be given a type.

The table has one primary key - Users\_ID. This has to be unique within the database. To ensure this, the database is set to automatically assign a value for this. I have set this to be one more than the previous assigned value. A user cannot enter a value here. The database does it instead.

CDList

This table stores the data of each CD within the database. It stores the name of the CD or collection, the number of tracks of the CD and whether it is being played. Similar to Users the primary ID of the table is the ID value. It is automatically given a value when a cd is added to the database.

Channels

The list of channels that can be viewed within the web site are stored in this table. The ID, the name, the URL and the Age of the channel are stored here. Its primary key, Channel\_ID, is automatically assigned a value.
In this table we come across a new data type, *text*. What is the difference between *text* and *varchar*?

- A *text* field can contain a lot more characters than *varchar*.
- A *text* column cannot be indexed.
- It is more difficult to query a *text* field.

**Media**

This lists all the media or audio tracks within the project. It stores the author, title and URL of the track. Each track is given an automatic ID value which is the primary key of the table.

**VideoList**

All the videos within the project are listed here. This contains quite a bit of data - the title, the description, the rating, the age certificate, the cost and the URL of each video. An important thing to note here is that the rating, age and cost of each video is given a default value - 0 (zero). This means that they do not have to be explicitly added when adding to the table.

**NewServices**

This table lists all the New Services which a user can then add to their main menu. Each Service has a Label, which is displayed on the menu, an Id, which allows it to be identified by the database, a Page, which gives the URL of the web page and finally the location of the Grammar file which will be used by the speech recognition controls.

There are two important things to note in this table:
1. The primary key of the table, which identifies each entry in the database uniquely, is composed of both the User_ID and ID fields. Up until now all the primary keys of the other tables were composed of just one field. We had to use two here because one would not uniquely identify each entry.

2. The Users_ID field references another field in another table. The Users_ID field is called a foreign key. It references the Users_ID field in the Users table. This means that only a valid User ID can be entered here.

Main_Page

This table is used to store the Main menu for each User. This has the same structure as the NewServices table. The only difference is that this allows the order of each menu item to be stored in the database. EG: If Music should be shown before Videos.

CDs

The List of all the CDs belonging to each User is stored here. It lists the ID of the User who can play the CD, the ID of the CD and whether it is being played. Like in previous tables Users_ID references Users_ID in the Users table. This is a foreign key. We also have a second foreign key, ID, which references ID in the CDList table. Both of these combine to form the primary key of the table. This table looks simpler than the previous but it can show more information than it displays.

One of the features of a Database is that it can use joins to join tables together. If you replace the ID value with all the entries associated with it in the CDList table then you get a more complex table. This has the advantage that we do not have to reduplicate information. This leads to savings in storage space. You also have to only update each CDList entry once and then these changes are reflected throughout the database.
Television

This table lists all the channels which each user can view. It is a simple table made up of two foreign keys: Users_ID and Channel_ID which references the Channels table. Like the CDs table this table can be expanded using joins to show all the details of the channels which each user can view. This is done using SQL statements.

Videos

This presents a list of all the Videos which the User can view. It has the same form as the Television table. The only difference is that, instead of a list of Channels displayed by Channel_ID, we get a list of Videos by Video_ID.

PPV

This table contains a list of PPV movies associated with each User. The table can again be expanded by using joins. The interesting point to note here is that there is an expiry time associated with each movie and user. This has type datetime which is quite a complex type. This type can represent dates in almost any format.

Tracks

The purpose of this table is to show all the tracks each user can listen to. Each track is also part of CD collection. All this had to be represented within the table. This makes it look quite complex as shown in figure 4.20.

There are three foreign keys in the table. Users_ID, which is used in many tables, ID, which references the CD and Media_ID, which references the track. The primary key of the table is made up of the three fields: Users_ID, ID and Track (number). Using this table we can find out the track URL for each CD according to each User.
4.4.3 Call Stored Procedures

```csharp
string test = "SELECT dbo.CDs.ID FROM";
    test += "dbo.CDList INNER JOIN dbo.CDs";
    test += "ON dbo.CDList.ID = dbo.CDs.ID";
    test += "WHERE (dbo.CDList.Title = " + title + """);
    test += "" + title + "" + ")";
```

Figure 4.21: Standard SQL

Figure 4.21 contains SQL code to get the ID of a CD based on its title. As you can see, it could get quite complex if complicated SQL statements are used. What makes it even more complex is that the statement has had to be spread over five lines. Quotation marks have had to be specially encoded because they could not be used directly within C# strings. It is also very difficult to see if there are errors with the SQL statement.

```csharp
sqlCommand1.CommandText = "getCDID";
SqlParameter myParm = sqlCommand1.Parameters.Add("@Title", SqlDbType.VarChar, 50);
    myParm.Value = title;
```

Figure 4.22: Call Stored Procedure

In contrast to this, the code in figure 4.22 is used to get the ID of a CD using the
CD title. This makes use of stored procedures. One of the reasons the code for my web services looks quite accessible is because I use these stored procedures. I did not have to code the SQL directly in C#. Someone could look at this method and see that a stored procedure, getCDID, is being invoked by sqlCommand1. They could also see that it takes a parameter, title. Obviously there is less of a chance to make a mistake here than with the code in figure 4.21.

```csharp
sqlCommand1.CommandText = "deleteVideo";
sqlCommand1.CommandType = CommandType.StoredProcedure;
SqlParameter myParm = sqlCommand1.Parameters.Add("ID", SqlDbType.Int, 4);
myParm.Value = ID;
sqlConnection1.Open();
sqlCommand1.ExecuteNonQuery();
sqlCommand1.Parameters.Clear();
sqlConnection1.Close();
```

Figure 4.23: Call deleteVideo Stored Procedure

Figure 4.23 demonstrates in code how to invoke a stored procedure. The steps below are what are needed to do this in general.

1. Set its CommandType property of a sqlCommand object to CommandType.StoredProcedure.

2. Set its CommandText property to the name of the stored procedure in the database.

3. Add any parameters using the Parameters.Add method. Note: Add them in the order they are declared in the stored procedure declaration.

4. Open the Database.

5. Use a DataReader Object to get any return data.

6. Close the Database.
This is all that is needed to query the database. You do not have to know the names of the database tables or fields. All of that is taken care of by the stored procedures. A list of stored procedures can be found in Appendix C.

4.4.4 Query Database Tables

The Stored Procedures are stored within the database itself. Each stored procedure has the form shown in figure 4.24. The project contains 74 of these Stored procedures. They can be broken up into 5 different types: Query procedures, Add procedures, Update procedures, Delete Procedures and a mixture.
Query Procedures

These procedures simply queried the database using certain criteria. An example of this is *listTracks* which is shown in figure 4.25. This has two parameters, *ID* and *User* (the sign signifies that they are parameters) both of which are of type int. What follows is a *SELECT..FROM..WHERE* SQL statement.

What we want to return from this procedure is the Track Name. This is what we *SELECT*. We select this from two tables, *Media* and *Tracks*. They are joined because we want all the data from both tables. However, this would return all the tracks in the database. We only want certain tracks from certain users. To do this we add an extra criteria to the *SELECT* statement using the *FROM* keyword. We only want tracks with an id of *ID* and whose user id is *User*. The selected tracks are then return when the procedure is invoked.
Add Procedures

Figure 4.26: addUser Stored Procedure

To query data from the database there has to be data within the database. This can be done by directly updating each table or by using stored procedures. Earlier I said that my project uses stored procedures to do this because I do not want people to directly access the tables. Figure 4.26 shows a stored procedure which adds a user to the database using the `INSERT INTO .. VALUES` SQL statement.

This statement is quite simple. We add the data or the `VALUES` to the database table with the appropriate field names. What is important to note here is that you have to explicitly give the field names. The Users table actually has 4 fields. To insert the data into their specific field we have to explicitly name the fields. Incidentally, the 4th field, `user_id`, is given an automatic value when the data is inserted into the database.
Update Procedure

![Stored Procedure Properties - updateUser](image)

Figure 4.27: *updateUser* Stored Procedure

Sometimes we might want to update the data in the database. In figure 4.27 we see a stored procedure which updates user information. This is accomplished using the `UPDATE..SET..WHERE` command.

First of all we have to give the table name which we want to `UPDATE`. In the case of figure 4.27 this is User. We then have to tell the database which fields we would like to update along with the new updated values. This would update every entry in the User table with this values. To be more specific we use the `WHERE` part of this statement to tell the database to update the User whose `Users_ID` field is `ID`.

Delete Procedures

Sometimes we might want to actually delete data from the database. This is not as easy as updating or adding to the database. The database is a relational database.
Some of the data that we might want to delete might actually be referenced elsewhere. If this is the case then the data cannot be deleted until the references are deleted.

![Stored Procedure Properties - deleteCD](image)

Figure 4.28: deleteCD Stored Procedure

In figure 4.28 we want to delete a CD from the database. The CD data is stored in the CDList table. It is entirely possible that this CD is referenced in the CDs table. To delete this CD the reference to it must first be deleted in CDs before finally deleting it from CDList.

Mixed Procedures

Some procedures are a mixture of all the above procedures. AddUserCredit in figure 4.29 first of all selects a user and then updates the user with more credit.
The syntax of both commands was explained earlier but there is an important thing to note in this statement and that is the `declare` keyword. Unlike, in normal SQL, stored procedures can have local variables. This can immensely improved the expressivity of SQL. In figure 4.29 we use this to get the current user’s credit and then later add the new credit to the old credit. This would be much more difficult to do in normal SQL statements.

### 4.5 Stage 4: Media Services

#### 4.5.1 Introduction

The main aim of the project is to let the user view videos, television and listen to music. The videos, music etc. are stored on a media server. The URL to this media
is stored on the database which is then embedded in the HTML code used by the User’s browser. When this code is invoked the User’s media player contacts the media server directly and should start to play the media.

4.5.2 Media Player HTML

The Microsoft Player SDK [6] contains the details of how to embed a player within a webpage using HTML code. I have provided a class which can generate this HTML for the User’s browser to show the videos or to play the music. There are 5 attributes of this class which define the player’s appearance and its state.

1. **URL** - the location of the media.

2. **autoStart** - if the player should begin immediately to play the media.

3. **position** - the position at which the player should start to play the media.
   Position should be in seconds. If not then `setPosition()` does the conversion.

4. **full** - if the player should be set to display the media in full screen or not.

5. **uiMode** - this defines how the player should look. There are two uiModes which I use: `none`, which means that the player should not display the player controls and `invisible`, which means it is hidden.

When the class is instantiated the developer can set any of these variables. The `toString()` method converts all the variables to the HTML code required to create a media player.
4.6 Configuration Tools

4.6.1 Introduction

All the other sections assumed that the Media is on the Server and that all the database entries are in place. But how did they get there in the first place? How do you add users and media to the system?

One of my main aims was to make the system secure. So far the only way to update the system to include the relevant information was to manually update the database. This has, naturally, security implications. As few people as possible should have direct access to the database. Adding entries to a database might not be intuitive so it would be better to create programs which would provide the user with an easy interface to do these tasks.

One of the main reasons I used Web services was to ensure that there is no need to directly access the database. Instead, users and developers use the web service to do this. From the start I created methods on the web service to allow people to add and update information in the database. My configuration tools use the web services to update and add new information to the database. These tools provide an easy interface for the user to do these tasks.

4.6.2 Media Server Tool

Every media file in this project has tags which give information about the type of media being played. For example: the title, author and rating of the media. This information had to be extracted from the file and entered into the database where it would be used later. I created a program which runs on the media server to extract this information and to save it to the database.

I had to use two SDKs to do this. The first, the Media Services SDK [7], allows you to connect to a Media Server and programmatically control it. From this I could
get a list of files. Then the Media Format SDK [5] allows you to extract all the relevant information from the media files.

The use of the two SDKs caused a major problem initially. Although you could use C# to control the Media Server, the Format SDK does not use C#. Instead, you have to use C++. You cannot mix both languages within a program. I wanted to continue to use C# because this would allow easy access to an Internet Server which I would use later. This meant that the program had to be split in two.

The main section of the program was created using C#. This connected to the media Server to get the file names of the media files. It would then send each file name to a C++ Dynamic Link Library (DLL) which would extract the relevant tags from the media file and pass it back to the main program. From here the file information would be added to the database using Web Services.

**Media Server**

All of the music and video files are stored on the music server. To access these files in a programmable way the Windows Media Services SDK [7] had to be used. This provides objects and methods which allow you to access the media server. Since this program runs on the Media Server the security and connection issues associated with connecting to remote Media Servers could be avoided.

The program connects to the local media server. The program connects to each Publishing point within the server. A publishing point can contain media files and playlists. A playlist contains a list of media files. Now the program has a list of files on the server.

If the file is a playlist then the program goes through the playlist listing each media file. Once the program has the filename for each media file it passes this to the DLL. This was initially a major problem since C# is managed code and C++ is unmanaged.
• **Managed code** is code which runs on the CLR (Common Language Runtime). This is similar to Java’s JVM except that any .Net compatible (see section 2.2.5) can run on the CLR. E.G: C#.

• **Unmanaged code** is any code that does not use this runtime. The code created is normally faster than unmanaged code cause it does not deal with garbage management, run-time type checking and reference checking. E.G: C++

```
StructLayout(LayoutKind.Sequential, CharSet = CharSet.Unicode)
```

```
public struct media
{
    [MarshalAs(UnmanagedType.LPWStr)]
    public string fileName;
    public int type;
    public audio audioInfo;
    public video videoInfo;
};
```

**Figure 4.30: A Marshalled C# Structure**

Because of the differences between them you cannot directly pass variables between the two. Instead, the variables have to be *marshalled* so that they become compatible. Figure 4.30 shows how a C# structure is defined using marshalling so that it can be used by the C++ DLL. The structure will be passed into the DLL in exactly the way it is ordered. Hence the use of the property `LayoutKind.Sequential`. The equivalent of a string in C++ is `LPWStr`. To convert or marshal as C# string you have to tell the compiler how this is to be done. In figure 4.30 the following code is used to declare this conversion:

```csharp
[MarshalAs(UnmanagedType.LPWStr)]
```
[DllImportAttribute("MediaDLL")]
public static extern void getFileInfo(ref media mediaStruct);

Figure 4.31: How to Declare a DLL within C#

Once all the variables have been declared in such a way they have to be passed into the DLL. To make the DLL accessible in C# you have to declare it. Figure 4.31 shows that we want to use an external (hence extern) method called getFileInfo. This takes a reference to a media structure. IE: a pointer in C++. The name of the DLL is MediaDLL. This can then be used like any normal method within the program.

The DLL passes back the structure which should contain the relevant data. The structure contains two sub structures: a video struct to store video information and an audio struct to store audio information. An integer called type tells us which type of media it is. If it is a video file (type=2) then we call the addvideo() method. This calls the VideoServices web service to add the relevant information to the database. If it is an audio file then the MusicServices service is called to add the information.

Media DLL

This section deals with the DLL which extracts the media information from the file. The DLL is written in C++ to take advantage of the Media Format SDK [5]. It exports my three methods: getFileInfo(), getAudioInfo() and getVideoInfo().

How to get media information from the file:

1. Create a metadata editor object, WMCreateEditor.

2. Using this editor open the file.

3. Retrieve header information from the file using the QueryInterface() method.

4. Choose the attribute you want to get the information from using the GetAttributeByName() method. For example: g_wszWMTitle gives the title of the
CHAPTER 4. IMPLEMENTATION

5. Get the size of this attribute then call the GetAttributeByName() method again. This should return the required attribute value.

getAudioInfo() and getVideoInfo() differ in respect to the attributes they return. GetAudioInfo() returns the title, author and rating of the file. GetVideoInfo() returns the title, author, rating and description of the video file. These can be called from outside the DLL if it is already known what format the file is.

If the format of the file is not known, IE: if it is unclear whether the file is a video or audio file, then getFileInfo() is used. This can look at the properties of the file and determine if there is a video stream within it.

How to get file properties:

1. Choose whether you want to view the file locally (IWMSyncReader) or over a network (IWMReader). IWMSyncReader will be used here.

2. Create a file reference using WMCreateSyncReader.

3. Open the file reference.

4. Get a list of the file’s properties using GetOutputProps().

Since we want to check what type the file is we use getType() on the properties we retrieved from the file. If it is found that there is a video stream within the file then we get back a value WMEDIAETYPE_Video. Otherwise, it is an audio file. Using this information either getAudioInfo() or getVideoInfo() can be called.
4.6.3 Project Configuration Tool

Figure 4.32 shows a screenshot of the initial page which allows a user to configure the database entries without directly accessing the database. Instead, the web services described in section 4.3 do this. The features of this tool are:

1. Add TV channels to project.

2. Add videos and music tracks to each User’s account. Section 4.6.2 describes how they are initially added to the database.

3. Either delete videos, music and television channels from each user or completely from the database.

4. Rename or update video files, music tracks or TV channels.

This section will be structured into three subsections. The first section will describe how to add entries to the database. The second will detail how to delete entries
from the database. Finally, the third section will show how to update the database entry.

Add Entries to Project

![Add User](image)

Figure 4.33: Add a User to the Database

If someone wants to add a User to the system they will be presented with the page shown in figure 4.33. This page makes use of the Services web service. Once all the fields have been filled my addUser() method of my the Services web service is called to add the user to the database.

To add a channel to the database a person has to fill in the name, age and URL of the channel. When these have been filled in then my addChannel() method of VideoServices is used to add this to the database.
If the administrator wants to add a video or music track to the database they will shown the page in figure 4.34. It lists all the users in the database excluding the Admin account. The list of users is retrieved using my listUsers() web method. Once the desired user is selected the next page is shown. If the administrator wants to add music to a User’s account they will then be shown a list of CDs not in the User’s account. They can then select the CD they want to add using my addCDToUser() method of MusicServices. If they want to add a video to a User’s account they will be shown a list of videos not in that User’s account. They desired video is added using addVideoToUser() of VideoServices.
CHAPTER 4. IMPLEMENTATION

Delete Entries From Project

If an administrator wants to delete music completely from the database they are shown figure 4.35. The list of CDs is retrieved using my web method listCDs(). This page is used to delete not only a CD completely from the system, but to delete the CD from a user’s account. To delete the CD completely a Session variable, Next, is set to DeleteMusic. When this is detected my web method deleteCD() is used to completely delete the CD. If they want to delete a video completely from the database they will be shown a page of video titles. deleteVideo() is called to completely delete the Video.

If the administrator wants to delete a CD only from a User’s account then the Session variable is set to DelMusic. Figure 4.34 is displayed to show a list of Users. Once a user has been selected then figure 4.35 is shown which lists all the CDs belonging to the User. Because the session variable has been set to DelMusic my web method deleteCDFromUser() is used to only delete the CD from the user, not from...
everyone. To delete a video from a user `deleteVideoFromUser()` is called.

To delete a User is a simpler task. The administrator is presented with a list of Users and can then select the user which they want deleted. To delete the User my `deleteUser()` method of `Services` is invoked.

Update Entries in Project

![Modify Video](image)

Figure 4.36: Modify a Video’s Attributes

Figure 4.36 shows the page used to modify a Video’s settings. Each field can be individually modified. When finished `updateVideo()` is called to update the video’s entry in the database.

To modify a User’s settings a similar page is shown which allows an administrator to change the name, age and credit details of the User. When all the fields have been
filled in then my \texttt{updateUser()} method of \textit{Services} is called.

4.7 Summary

This was the longest chapter in this write-up and also, probably, the most important. It describes, using the technologies in chapters 2 and 3, how the project was built up. The main section of the project was described in four stages and then finally the two tools used to configure the project were described.

The first stage of the implementation was to build up the web interface. This was first of all created using server controls which defined the general look-and-feel of the project. These were then used in all the pages of the project. Then I discussed how the individual pages were created and what methods were used to build these up. This was important because it set up the GUI for the User. It has to show all the relevant information to the user without being overly complex. The interface had to also be consistent from page to page to maintain the integrated-feel of the project. This was difficult to do because some pages just showed menus, others let the user listen to music whilst other pages had to show video previews.

The second stage dealt with the web services. It described how the various services were called by each of the web pages described in the first stage of the implementation. A description of each web service was given along with how the web services call the stored procedures in the database. The security implications of this were important. Since the interface does not directly access the database but uses a web service, database security could be maintained. Another advantage of this is that the ASP.NET pages described in the first stage did not have to be cluttered up with database access code. This helps with the overall readability of the code.

The third stage detailed the stored procedures and database tables. An overview of each table was given. In this the important features of each table were described.
CHAPTER 4. IMPLEMENTATION

The various types of stored procedures were then discussed and how they accessed the database tables. As with stage two an advantage of doing this is that the web services are not cluttered up with SQL code.

How the web browser displayed the audio and video was described in the fourth and final section of the main implementation of the project. This section detailed how the Windows Media Player is created by HTML within a client’s browser. The HTML code is created on the server using the media URLs.

I also provided two tools which help to configure the project. The first tool accessed the media server and saved the locations of the media in the database. The second tool allows someone to configure the Users, their video, audio and channel settings. When I configured the project these tools were quite useful since they simplified adding and removing entries to the database.
Chapter 5

Results

5.1 Introduction

This chapter will be divided into two sections. The first section will discuss my thoughts on the project and the problems I encountered while creating it. Some people tested my project and their feedback, both positive and negative, is outlined in the second section.

5.2 Personal Assessment

In this section I will detail my own views of the project. I will discuss the positive aspects of the system and also the negative aspects that I encountered while creating the system.

5.2.1 Positive Aspects

Even though I had many problems with the speech recognition system (outlined in section 5.2.2) I was surprised at how well it fared overall. The engine used is built into Windows XP and therefore can be thought of as a free speech recognition engine. Bearing this in mind the results, when the system is not used in a noisy environment,
were very good. I was consistently getting excellent accuracy. I could use the system and it would rarely misinterpret my responses and make a wrong decision.

Sections 2 and 3 describe the various technologies used by the project. Since I was using so many technologies, especially a database server, I was afraid that this would have a baring on speed. However, this did not turn out to be the case. When the database is first initialised there is a delay but after that any delay is negligible. This is important because it would not be very good if it took a couple of seconds between the user saying something and the page changing to reflect what the user said. In most cases, the delay is less than a second which I believe is very good.

I am quite pleased with how the media-end of project went. Normally, when I stream audio and video from the web, it takes a few seconds for the stream to buffer before playing. This is acceptable in a computer environment but not in a TV environment. A user would not like it if, every time they changed videos or music, there is a delay of a few seconds. I am delighted that I was able to use the Fast Playback feature of Windows Media Server which virtually eliminated this problem. Audio and video start almost instantly now. There does not appear to be any buffering.

5.2.2 Negative Aspects

One of the big problems with this project was the lack of documentation on the speech SDK. Since it is in beta the documentation is quite incomplete. The controls themselves were not very well described. It took me a while to figure out the properties of each control. EG: where could it be used, what features it has etc.. Also there was a complete lack of code examples of each feature. For example: when I wanted to create dynamic grammars there were absolutely no code examples. By chance I happened to find a property which allowed this. To actually figure out the use of the
SDK features one must experiment with them and see what does and does not work. This process of trial-and-error can be quite laborious.

A problem which I did not manage to overcome was a timeout problem. When a speech control is activated then it is given a set time after which the control will stop and then reactivate itself if the user says nothing. This means that the control’s prompt will be called out again. This can be quite annoying because after, for example, every 3 minutes the control will repeat itself if nothing is said. There is no way to set an infinite timeout value. The largest number that can be given is a 32bit integer value which will represent the timeout in milliseconds.

Another problem with the system is the speech recognition engine. The engine supplied with Windows XP has three problems:

1. The language version used is US English. Obviously, there are a number of pronunciation differences between Hiberno-English and US English [11]. This would means less accurate recognition results when Irish users use the system. The same could be same about British English.

2. To get very good results the speech recognition engine can be trained for a specific user. This can lead to problems if more than one person uses the system. If the system is left untrained then it appears to work better if many users use the system.

3. In environments with a lot of background noise, speech recognition results do not appear to be very good. However, this depends heavily on the microphone used, so a microphone designed for such an environment would lead to better results.
5.3 User Evaluation

The usefulness of this system would be limited if only one user could use the system. Because of this, I asked a variety of people to use the system. Their knowledge of computers ranged from basic to advanced. I demonstrated the system to them and then asked them to try it for themselves. After using the system they gave their feedback — both positive and negative.

5.3.1 Positive Feedback

People were quite surprised at how robust the system seemed to be. In most cases, and to my surprise, voice recognition worked quite well. This was true for both the male and female voice. I did not expect this since I was led to believe that the system was trained to my voice.

All of the people which I questioned about the system remarked about how useful it is to be able to control the system without using a keyboard. Many said that it would be very useful for older people who might not be able to see the screen properly or who would not be able to use a remote control. This fitted in nicely with my aim that the system should be useable by such people. Something that I did not think about was that the system would be useable by people who might not be physically able to use a remote control. For example, because of arteritis. A voice-controlled system would be very useful to this group of people.

The general consensus was that the system was quite fast. Moving between pages seemed very fast to some people. However, the first person to use it noticed that there was a delay while the system initialised for the first time. However, this only happens once when the database is started up. Afterwards it should run much faster, which the other users noticed.

When younger people (under 25 years of age) first used the system they simple
called out the menu item they wanted to select. E.g: “Music”. They did not realise that they could use phrases like “I want to listen to music”. Older people (above 40) tended to use the latter. This flexibility impressed both groups when it was pointed out to them.

### 5.3.2 Negative Feedback

Even though speech recognition worked remarkably well there were times when it did not. The speech control then repeated its prompt. All of the users noticed this and had to repeat their reply. Naturally, they found this one of the main problems with the system.

Tying in with speech recognition accuracy most users asked whether the system would be useful for those with speech problems or stutters. Unfortunately, recognition accuracy would drop dramatically if such users used this system. The speech recognition engine would find such spoken input very difficult to understand.

Other users queried whether the system could be used for other languages. It would not, unless the speech engine was designed for that language. Currently, the engine is a US English speech engine. If the engine is replaced with, for example, a Japanese language engine then English speakers would be unable to use the system. Only one language can be used at once.

### 5.4 Conclusion

Although I had problems with the Speech SDK documentation and the recognition errors, I am pleased with how well the system turned out overall. I was delighted with the feedback I got from the people who tested my system. They were very enthusiastic about it. Most could foresee it being very useful to a wide range of people. What was
interesting to note is that it took only a few minutes to show them how to use the
system. They found it to be quite intuitive. I find this a very encouraging response.
Although they did find some issues, with the system these were overshadowed by the
positive aspects they saw in the system.
Chapter 6

Conclusion

6.1 Assessment

At the very start I listed the aims of my project: two fundamental aims along with some secondary aims. I think I succeeded in achieving these aims.

The main aim, to make the system navigable by voice, was very important. From the start I tried to make this possible. A person can move throughout the whole website, control videos, music and TV just by using their voice. There is no need to use the keyboard. I think this is certainly a great success. The fact that the speech input does not have been in a particular form is also important. For example, rather than require the user to say music in the Main Menu, they can use normal English phrases like a “I want to listen to music”. This should make it easier for people who find computers intimidating. The speech interface is aimed at these people.

A further aim was to integrate the features of a CD player, video recorder and TV into one system. I believe this was achieved. The system presents the user with a standard webpage which can show TV, videos (be they cost-free videos or pay-per-view) or listen to music. All this using just a standard PC. The user can change channels, not just by selecting a particular channel number like normal, but by saying
for example “I want to watch BBC1”. This is then streamed to the user’s computer. Likewise with music and videos.

I didn’t try to emulate television, cd player etc.. functions in a computer environment. I tried to use the benefits of computers to make these things simpler. For example, people are used to selecting channels by number but my system works by name, which is much more natural for humans. People can selecting music by just saying the name of the CD. They do not have to go search for the CD, put it in the hi-fi and press a few buttons to play it. This makes daily tasks much simpler with my system because the user does not have to look for CDs or videos. They are all stored centrally.

The system was designed from the ground up to be scalable. A computer can be dedicated to be the WebServer, and another to be the database server etc. if more power is required. Conversely, the whole system can run on just one computer. This means that it should be able to grow from supplying entertainment to a small number of users to serving a much larger group of users. The Database server, SQL Server, is designed to server large numbers of users and request simultaneously. Web servers can be clustered together to speed things up if it appears that web requests are slowing down. All of these help to achieve the goal of making the system scalable.

From the start I tried to limit any possible security implications of using the various programs of the system. E.g: the WebServer and database server. The user does not directly contact the database. In fact there are three steps in between - the web site, the web services and the database itself. This is extremely important because if the database becomes comprised then the whole system becomes compromised. Even a developer of the user interface does not have access to the database. They should never need access. Any data they require should be requested by the web service. Even the web services have limited access to the database. They access the database through stored procedures. They do not directly access the database
tables. This has obvious security advantages. Since each stored procedure has security permissions attached then only authorised users or services can invoke them. This shows that the system can be kept quite secure. Unauthorised users would have a very hard time trying to access parts of the system which they shouldn’t.

Since I assumed that not only one user would use the system but many, it would be logical to assume that people of different ages could use it. The whole system is designed to be safe for young people. The menus will not show movies for underage people or TV channels for underage people. This has traditionally been a problem for most parents. A small but useful feature of the system.

At the beginning of the project I decided to implement a feature which has more commercial applications - that is Pay-per-view movies. The system was built up to take advantage of this. Every user has a credit level. They can purchase movies (assuming they are over age) and watch them for a certain amount of time. The cost of the movie is credited from their account. Of course all of this is achieved using voice commands.

6.2 Future Work

There is plenty of scope for future work in this project. In this section I will outline some directions which could be taken to expand the system.

Speech Recognition

The speech recognition engine in Windows was adequate for my needs. However, the engine did have some problems with recognition and noise. One of the advantages of using speech recognition within Windows is that the speech engine can be replaced with another version. There are many commercial such systems available which might be more suited to certain environments etc.. This does not affect any of the code in
the project so no changes would have to be made to accommodate this new engine.

**Updated Speech SDK**

The version of the Speech SDK [8] which I used was a Beta 2. Obviously, since it is a beta version there are many problems associated with it which I described earlier. If a new version comes out then this should cause less problems than the current version.

**Database**

The database Server which I used was SQL Server 2000. This is an industrial database with many features, most of which I did not use. The database has many features which allow better security, easier development and features to do with scalability. By using some of these features the database part of the project could be beefed up to handle additional tasks or make current tasks easier.

**Front Page Security**

The front page is important because it is here that the User is selected. There are no security features here to only let specific users in. As long as the speech control recognises the user name then the user can use the program. This could be stopped by incorporating authentication systems a password system.

To be more in keeping with the speech recognition side of the project some sort of voice authentication or fingerprinting system could be used. Voice authentication has numerous advantages over normal authentication systems. These include:

1. This is quite a fast means of authentication. It takes less than half a second to authenticate a user [2]

2. It is an easy method to use. There are no passwords or PINs to remember.

3. There is no need to disclose personal information.
4. It is an accurate form of biometric authentication with about 0.3% false acceptance rate [1].

**User Interface**

Obviously, the User Interface could be updated. I tried to make the Interface as independent from the other systems as possible. For example: using web services to get data. The interface could be updated but the other parts of the project do not necessarily have to be updated at the same time.

**Other Languages**

The Speech Engine I used comes in three languages: US English, Japanese and German. By changing the engine language different languages could be used.
Appendix A

Database Tables

Figure A.1: CDList and CDs Tables

Figure A.2: Channels and Main_Page Tables
Figure A.3: Media and NewServices Tables

Figure A.4: PPV and Television Tables

Figure A.5: Tracks and Users Table
<table>
<thead>
<tr>
<th>Key</th>
<th>ID</th>
<th>Name</th>
<th>Data Type</th>
<th>Size</th>
<th>Nulls</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Video_ID</td>
<td>int</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Title</td>
<td>varchar</td>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Description</td>
<td>text</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rating</td>
<td>int</td>
<td>4</td>
<td></td>
<td>(0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Age</td>
<td>int</td>
<td>4</td>
<td></td>
<td>(0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cost</td>
<td>money</td>
<td>8</td>
<td></td>
<td>(0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>URL</td>
<td>text</td>
<td>16</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>ID</th>
<th>Name</th>
<th>Data Type</th>
<th>Size</th>
<th>Nulls</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Users_ID</td>
<td>int</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Video_ID</td>
<td>int</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure A.6: VideoList and Videos Table
Appendix B

Web Services Methods

B.1 Services

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addServiceToMenu</td>
<td>Add a selected Service to the User's Main Menu</td>
</tr>
<tr>
<td>addUser</td>
<td>Add a User to the System</td>
</tr>
<tr>
<td>addUserCredit</td>
<td>Add Credit to the User’s Account</td>
</tr>
<tr>
<td>deleteUser</td>
<td>Delete a User from the System</td>
</tr>
<tr>
<td>getPageWithID</td>
<td>Get the Page URL from its Label (name)</td>
</tr>
<tr>
<td>getUserAge</td>
<td>Get the Age of the User</td>
</tr>
<tr>
<td>getUserCredit</td>
<td>Get the User’s Credit</td>
</tr>
<tr>
<td>getUserID</td>
<td>Get the User’s ID</td>
</tr>
<tr>
<td>getUserName</td>
<td>Get the User’s Name</td>
</tr>
<tr>
<td>listNewServices</td>
<td>Get the New Services of a User</td>
</tr>
<tr>
<td>listSubscribedGrammars</td>
<td>Get the URLs of the Grammars</td>
</tr>
<tr>
<td>listSubscribedServices</td>
<td>Get the Names of the Services</td>
</tr>
<tr>
<td>listUsers</td>
<td>List All the Users of the System (except Admin)</td>
</tr>
<tr>
<td>updateUser</td>
<td>Update Information on User</td>
</tr>
</tbody>
</table>
## B.2 MusicServices

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addCD</td>
<td>Add a CD to the Database</td>
</tr>
<tr>
<td>addCDToUser</td>
<td>Add the CD to a User’s Account</td>
</tr>
<tr>
<td>addTrack</td>
<td>Add a Track to a CD</td>
</tr>
<tr>
<td>addTrackToUser</td>
<td>Add a Track to a User</td>
</tr>
<tr>
<td>deleteCD</td>
<td>Delete the CD Completely from the Database</td>
</tr>
<tr>
<td>deleteCDFromUser</td>
<td>Delete the CD reference only from the User</td>
</tr>
<tr>
<td>getCDID</td>
<td>Get the ID used to reference the CD</td>
</tr>
<tr>
<td>getTrackName</td>
<td>Get the Tracks Name from its ID</td>
</tr>
<tr>
<td>getURL</td>
<td>Get the URL of a specified Track</td>
</tr>
<tr>
<td>lastPlayedCD</td>
<td>Get the ID of the Last played CD</td>
</tr>
<tr>
<td>listCDs</td>
<td>List the CDs belonging to a User</td>
</tr>
<tr>
<td>listTracks</td>
<td>List the Tracks of a Specified User’s CD</td>
</tr>
<tr>
<td>listTracksOfLastPlayedCD</td>
<td>List all the Tracks of the Last Played CD</td>
</tr>
<tr>
<td>setLastPlayedCD</td>
<td>Set the Last Played CD of Each User</td>
</tr>
</tbody>
</table>
### B.3 VideoServices

#### B.3.1 Video Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addVideo</td>
<td>Add a Video to the Database</td>
</tr>
<tr>
<td>addVideoToUser</td>
<td>Add the Video to a User</td>
</tr>
<tr>
<td>deleteVideo</td>
<td>Delete the Video completely</td>
</tr>
<tr>
<td>deleteVideoFromUser</td>
<td>Delete the Video from a User’s Account</td>
</tr>
<tr>
<td>getExpiryDate</td>
<td>Get the Expiry Date of the Purchased Video</td>
</tr>
<tr>
<td>getNextFreeVideoID</td>
<td>Get the ID of the Next Free Video</td>
</tr>
<tr>
<td>getNextPPVVideoID</td>
<td>Get the ID of the Next PPV Video</td>
</tr>
<tr>
<td>getNextPurchasedVideoID</td>
<td>Get the ID of the Next Purchased Video</td>
</tr>
<tr>
<td>getVideoAge</td>
<td>Get the age rating of the Video</td>
</tr>
<tr>
<td>getVideoCost</td>
<td>Get the cost of the Video</td>
</tr>
<tr>
<td>getVideoID</td>
<td>Get the ID of the Video according to its Name</td>
</tr>
<tr>
<td>getVideoDescription</td>
<td>Get the Description of the Video</td>
</tr>
<tr>
<td>getVideoRating</td>
<td>Get the rating of the Video (1-5 stars)</td>
</tr>
<tr>
<td>getVideoTitle</td>
<td>Get the Title of the Video</td>
</tr>
<tr>
<td>getVideoURL</td>
<td>Get the URL of the Video</td>
</tr>
<tr>
<td>hasBeenPurchased</td>
<td>Has the User bought the video?</td>
</tr>
<tr>
<td>isPPV</td>
<td>Is the Video a PPV video?</td>
</tr>
<tr>
<td>listAllVideos</td>
<td>List both PPV and Free Videos</td>
</tr>
<tr>
<td>listFreeVideos</td>
<td>List the Free Videos a User can Watch</td>
</tr>
<tr>
<td>listPPVVideos</td>
<td>List the PPV Videos a User can Watch</td>
</tr>
<tr>
<td>listPurchasedVideos</td>
<td>List Videos purchased by User</td>
</tr>
<tr>
<td>purchaseVideo</td>
<td>Purchase a Video (with enough Credit)</td>
</tr>
<tr>
<td>updateVideo</td>
<td>Update Video Details</td>
</tr>
</tbody>
</table>
## B.3.2 Channel Methods

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addChannel</td>
<td>Add a Channel to the Database</td>
</tr>
<tr>
<td>addChannelToUser</td>
<td>Add the Channel to a User</td>
</tr>
<tr>
<td>deleteChannel</td>
<td>Delete the Channel completely from the Database</td>
</tr>
<tr>
<td>deleteChannelFromUser</td>
<td>Only Delete the Channel from the User</td>
</tr>
<tr>
<td>getChannelAge</td>
<td>Get the Age rating of the Channel</td>
</tr>
<tr>
<td>getChannelID</td>
<td>The ID of a channel according to its Name</td>
</tr>
<tr>
<td>getChannelName</td>
<td>Get the Name of the Channel</td>
</tr>
<tr>
<td>getChannelURL</td>
<td>Get the URL of the Channel</td>
</tr>
<tr>
<td>listChannels</td>
<td>List the Channels a User can watch</td>
</tr>
<tr>
<td>updateChannel</td>
<td>Update Channel Details</td>
</tr>
</tbody>
</table>
Appendix C

Stored Procedures

C.1 Access User Entries

<table>
<thead>
<tr>
<th>Procedure Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addToMainPage</td>
<td>Add a New Service to a User’s Subscribed Services</td>
</tr>
<tr>
<td>addUser</td>
<td>Add a User to the Database</td>
</tr>
<tr>
<td>addUserCredit</td>
<td>Add Credit to a User’s account</td>
</tr>
<tr>
<td>deleteFromNewServices</td>
<td>Delete an entry from New Services</td>
</tr>
<tr>
<td>deleteUser</td>
<td>Delete a User and their Database entries</td>
</tr>
<tr>
<td>getNewServiceDetails</td>
<td>Get the name, URL, and Grammar location</td>
</tr>
<tr>
<td>getUserAge</td>
<td>Get the age of a User</td>
</tr>
<tr>
<td>getUserCredit</td>
<td>Get the amount of Credit a User has</td>
</tr>
<tr>
<td>getUserID</td>
<td>Get a User’s ID</td>
</tr>
<tr>
<td>getUserName</td>
<td>Get the User’s Name</td>
</tr>
<tr>
<td>listUsers</td>
<td>List all the Users in the Database (except Admin)</td>
</tr>
<tr>
<td>listNewServices</td>
<td>List all the New Services of a User</td>
</tr>
<tr>
<td>NumNewServices</td>
<td>Now many New Services a User has</td>
</tr>
<tr>
<td>NumSubscribedServices</td>
<td>How many Subscribed Services a User has</td>
</tr>
<tr>
<td>PageByID</td>
<td>Get the page URL from its name</td>
</tr>
<tr>
<td>SubscribedGrammars</td>
<td>List all the Subscribed Grammars of a User</td>
</tr>
<tr>
<td>SubscribedServices</td>
<td>List all the Subscribed Services of a User</td>
</tr>
<tr>
<td>updateUser</td>
<td>Update a User’s settings</td>
</tr>
</tbody>
</table>
## C.2 Access Music Entries

<table>
<thead>
<tr>
<th>Procedure Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addCD</td>
<td>Add a CD to the Database</td>
</tr>
<tr>
<td>addCDToUser</td>
<td>Add a CD to a User’s account</td>
</tr>
<tr>
<td>addTrack</td>
<td>Add a track to the Database</td>
</tr>
<tr>
<td>addTrackToUser</td>
<td>Add a track to a User’s account</td>
</tr>
<tr>
<td>deleteCD</td>
<td>Completely delete a CD from the database</td>
</tr>
<tr>
<td>deleteCDFromUser</td>
<td>Delete a CD from a User’s account</td>
</tr>
<tr>
<td>getCDID</td>
<td>Get the ID of a CD</td>
</tr>
<tr>
<td>getTrackName</td>
<td>Get the Name of a track</td>
</tr>
<tr>
<td>getTrackURL</td>
<td>Get the URL of a track</td>
</tr>
<tr>
<td>lastPlayedCD</td>
<td>Get the ID of the last played CD</td>
</tr>
<tr>
<td>listCDs</td>
<td>List all a User’s CDs</td>
</tr>
<tr>
<td>listTracks</td>
<td>List the tracks of a particular CD</td>
</tr>
<tr>
<td>NumCDs</td>
<td>How many CDs does a User have</td>
</tr>
<tr>
<td>NumTracks</td>
<td>How many tracks in a particular CD</td>
</tr>
<tr>
<td>setLastPlayedCD</td>
<td>Set the last played CD</td>
</tr>
</tbody>
</table>
## C.3 Access Television Entries

<table>
<thead>
<tr>
<th>Procedure Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addChannel</td>
<td>Add a Channel to the Database</td>
</tr>
<tr>
<td>addChannelToUser</td>
<td>Add a Channel to a User’s account</td>
</tr>
<tr>
<td>deleteChannelFromUser</td>
<td>Delete a Channel from a User’s Account</td>
</tr>
<tr>
<td>deleteChannelID</td>
<td>Delete a Channel completely from the Database</td>
</tr>
<tr>
<td>getChannelAge</td>
<td>Get the age rating of a Channel</td>
</tr>
<tr>
<td>getChannelID</td>
<td>Get the ID of a Channel</td>
</tr>
<tr>
<td>getChannelName</td>
<td>Get the Name of a Channel</td>
</tr>
<tr>
<td>getChannelURL</td>
<td>Get the URL of a Channel</td>
</tr>
<tr>
<td>listChannels</td>
<td>List the Channels a User can watch</td>
</tr>
<tr>
<td>NumChannels</td>
<td>How many Channels can a User watch</td>
</tr>
<tr>
<td>updateChannel</td>
<td>Update a Channel’s entry</td>
</tr>
</tbody>
</table>
## C.4 Access Video Entries

<table>
<thead>
<tr>
<th>Procedure Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addVideo</td>
<td>Add a Video to the Database</td>
</tr>
<tr>
<td>addVideoToUser</td>
<td>Add a Video to a User’s account</td>
</tr>
<tr>
<td>CanView</td>
<td>See if a User can view a Purchased Video</td>
</tr>
<tr>
<td>deleteVideo</td>
<td>Delete a Video from the Database</td>
</tr>
<tr>
<td>deleteVideoFromUser</td>
<td>Delete a Video from a User’s account</td>
</tr>
<tr>
<td>getExpiryDate</td>
<td>Get the Expiry Date of a Purchased Video</td>
</tr>
<tr>
<td>getVideoAge</td>
<td>Get the age rating of a Video</td>
</tr>
<tr>
<td>getVideoCost</td>
<td>Get the Cost of a Video</td>
</tr>
<tr>
<td>getVideoDescription</td>
<td>Get the Description of a Video</td>
</tr>
<tr>
<td>getVideoID</td>
<td>Get the ID of a Video</td>
</tr>
<tr>
<td>getVideoRating</td>
<td>Get the rating (1-5 stars) of a Video</td>
</tr>
<tr>
<td>getVideoTitle</td>
<td>Get the title of a Video</td>
</tr>
<tr>
<td>getVideoURL</td>
<td>Get the URL of a Video</td>
</tr>
<tr>
<td>getVideoURLByName</td>
<td>Get the URL of a Video by its Name</td>
</tr>
<tr>
<td>isPPV</td>
<td>Is a Movie a PPV movie</td>
</tr>
<tr>
<td>isPurchased</td>
<td>Has the Video been purchased</td>
</tr>
<tr>
<td>listAllMovies</td>
<td>List all movies, both Free and PPV, of a User</td>
</tr>
<tr>
<td>listFreeMovies</td>
<td>List all the Free Movies in a User’s account</td>
</tr>
<tr>
<td>listPPVMovies</td>
<td>List all The PPV Movies a User can watch</td>
</tr>
<tr>
<td>listPurchasedMovies</td>
<td>List all the Videos a User has bought</td>
</tr>
<tr>
<td>maxFreeVideos</td>
<td>The maximum ID of a Free Video</td>
</tr>
<tr>
<td>maxPPVMovies</td>
<td>The maximum ID of a PPV Video</td>
</tr>
<tr>
<td>maxPurchasedMovies</td>
<td>The maximum ID of a Purchased Video</td>
</tr>
<tr>
<td>NumFreeMovies</td>
<td>How many Free Movies can a User watch</td>
</tr>
<tr>
<td>NumMovies</td>
<td>How many Movies can a User watch</td>
</tr>
<tr>
<td>NumPPVMovies</td>
<td>How many PPV Movies can a User watch</td>
</tr>
<tr>
<td>NumPurchasedMovies</td>
<td>How many Purchased Movies can a User watch</td>
</tr>
<tr>
<td>PurchaseMovie</td>
<td>Purchase a PPV movie for a User</td>
</tr>
<tr>
<td>updateVideo</td>
<td>Update a Video’s entry</td>
</tr>
</tbody>
</table>
Bibliography

[1] Biometrics: An Overview,


[3] Developing ASP.Net Server Controls,
http://www.w3.org/TR/ws-gloss.


[5] Microsoft Media Format,

[6] Microsoft Media Player,

[7] Microsoft Media Services 9 Series,

[8] Microsoft Speech,


