Development of an Internet Jukebox System

By

Keith Mitchell

Submitted to
the Faculty of the Information Engineering Technology Program
in partial fulfillment of the requirements for
the Degree of Bachelor of Science
in Information Engineering Technology

University of Cincinnati
College of Applied Science

June 2003

Development of an Internet Jukebox System

By

Keith Mitchell

Submitted to
the Faculty of the Information Engineering Technology Program
in partial fulfillment of the requirements for
the Degree of Bachelor of Science
in Information Engineering Technology

Keith Mitchell — Date: 6-11-03

Russel McMahon, Faculty Advisor — Date: 6/10/03

James F. Sullivan, Department Head — Date: 11 June, 2003

Acknowledgements

First I would like to thank my wife Jamie for her patience and understanding for the last 6 months. This project could not have happened without her help. Next I would like to thank my advisor Russ McMahon for his guidance and help. It was not easy dealing with technology that no one was familiar with and Russ did a great job with helping me handle deadlines and goals. Finally, I would like to thank everyone who spent time researching, answering my questions, and pointing me in directions when I was completely lost, specifically; John Nyland, Matt Huffman, Brian Harwell, Mike Abrenica, Chris Butler, and anyone else that traveled down this path with me.

Table of Contents

List of Figures

Abstract

NetRoxs is a remake of the classic Jukebox with an infusion of the latest technology. Current jukebox systems use compact disks for music and still rely on mechanical means for getting and playing a particular song. *NetRoxs* uses some of the most current technology to achieve a more "advanced" technological experience. This system was created using Microsoft's .NET architecture: ASP.NET, C#, and SQL Server 2000. This system offers fast retrieval of selections and a much more dynamic way of searching for a particular artist or song.

NetRoxs: A Digital Internet Jukebox

1. Statement of the Problem

After having spent many years working in restaurants, bars, and clubs I have found that there has always been one common factor that links all of these diverse atmospheres: the jukebox. Often the jukebox can be found nestled away in a corner by the pool tables or sitting against the wall near the bar. Purchasing one of these behemoths can be relatively prohibitive, costing anywhere in the range of $6,000.00 to $10,000.00, or more for a sophisticated model (7). In addition to the initial cost of the jukebox, an establishment then needs to purchase the music that is to be played on the jukebox, pay the cost of the individual that is to install the equipment and for any repairs that may be necessary. Assuming everything works out perfectly, a typical owner may not see a return on their investment for at least five years (7). This can lead to an amazingly complex and expensive process given the wide variety of music genres requested by a general audience.

A more cost effective and efficient method of playing music in these establishments would be a "Digital Jukebox". In order for this to be practical it must meet the following specifications:

- It must be intelligible. For this application to be successful a simple interface is required.
- Speed. With an estimated access to over 1000 songs, it needs to be able to access its source quickly.
- Dynamic Storage. Since the musical landscape is constantly changing it is critical that this system be able to acquire new music or selections that may not be frequently played.

- Compactness. This is a critical component for any establishment. It must not take up valuable floor space. Ideally, it should be able to be mounted on a wall.

- Minimal Resources. The only piece of networking necessary would be a Network Interface Card (NIC) that is capable of handling both standard phone line and ADSL or cable modem.

2. Description of the Solution

This system is utilizing the potential of the MP3 format for digital music. The advantages of using MP3's are the music occupies $1/10^{th}$ the disk space taken by a conventional CD jukebox, as well as a digital sound. Currently there are companies offering these systems (7), however the current design scheme is still embedded in the idea of the big nickelodeon style of the 50's and 60's.

After considering the long term implications for a project such as this, Microsoft's SQL Server 2000 was determined to be the most efficient database system. This has worked well since Microsoft's .NET architecture was designed to work with SQL Server 2000 seamlessly.

The programming language I used for this application was Microsoft's® C#, part of the new .NET architecture. This was due to C#'s rich ADO and XML capabilities. The interface was developed to employ a familiar web page style. HTML, XML, and JavaScript were used to develop a friendly, easy to use graphical interface for the user. It has always been essential that the graphical interface work seamlessly with the database in order to provide efficient service.

Another aspect of this project worth noting is the murky area known as copyright laws. The majority of all music is protected by three major license companies: The American Society of Composers, Authors, and Publishers (ASCAP), Broadcast Music

Incorporated (BMI), and The Society of European Stage Authors and Composers (SESAC). Each of these performance rights licensing agencies use an organization called the Jukebox Licensing Office (6). Using the Jukebox License Agreement would enable customers to have access to all the music protected by any of the three major license companies (6).

3. User Profile

This application was made available to users with a wide range of skills relating to computers. Since this will attract individuals from extremely different backgrounds it was necessary to design an interface that is easy to use for the novice. Anyone with minimal web browsing experience can navigate this application.

4. Design Protocols

In order to create an easy to use yet powerful user interface, I decided that ASP.NET with C# was the best tool for this task. ASP.NET provides easy to develop web controls that interact with SQL Server 2000 optimizing C#'s rich ADO and XML capabilities.

4.1 Multimedia

The multimedia aspect of this project will involve the overall design of the user interface. Maintaining a simple user interface along with some type of flash animations will be key to making this product user friendly.

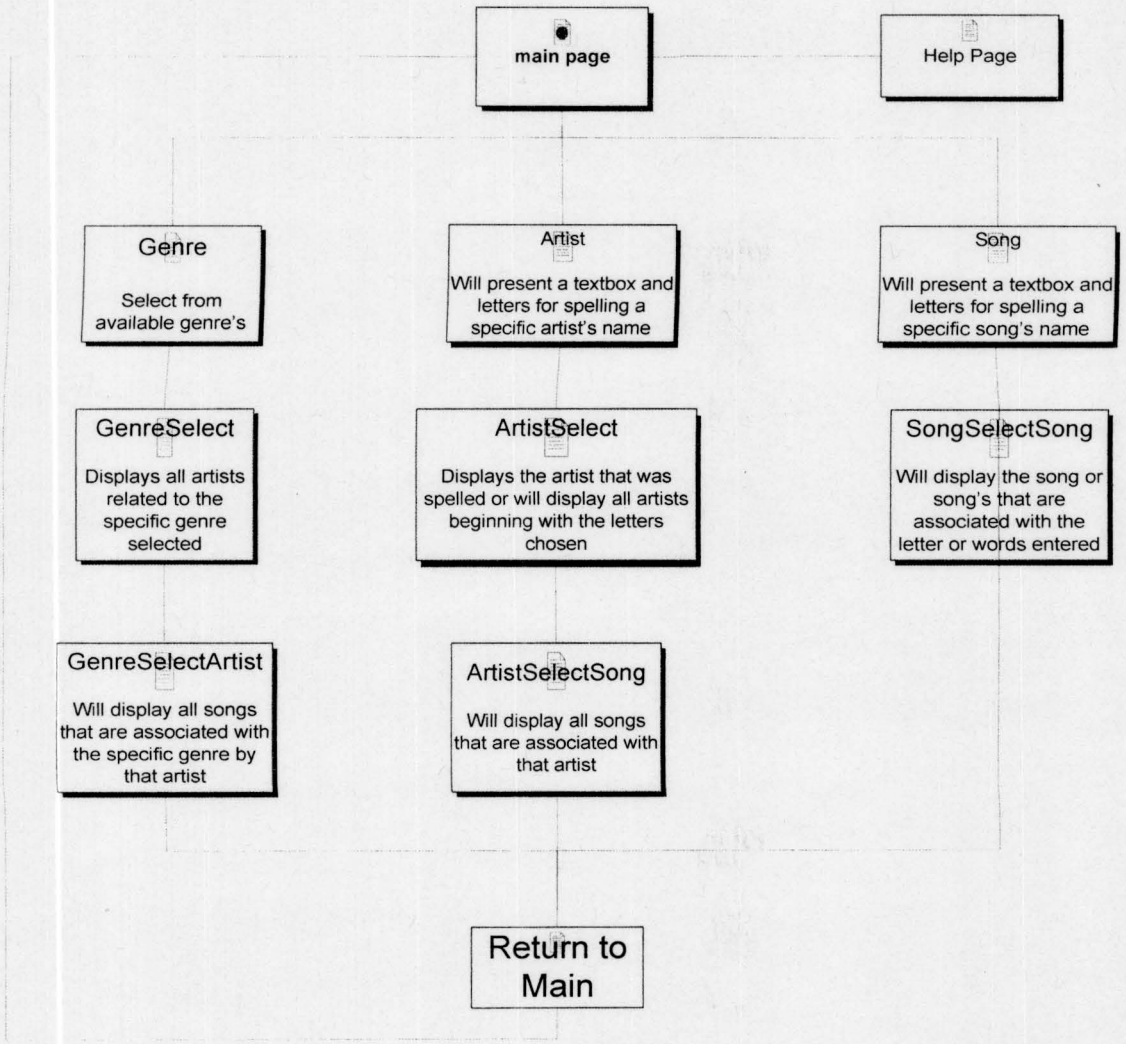
4.2 Networking

Since I am using ASP.NET, a familiarity with Microsoft's Internet Information Server (IIS) software was essential. It was also necessary as my project progressed to move it towards a more logical Client/Server networking configuration. However, due to

various problems discussed later, I was unable to actually move this project to that style of architecture.

4.3 Screen Design

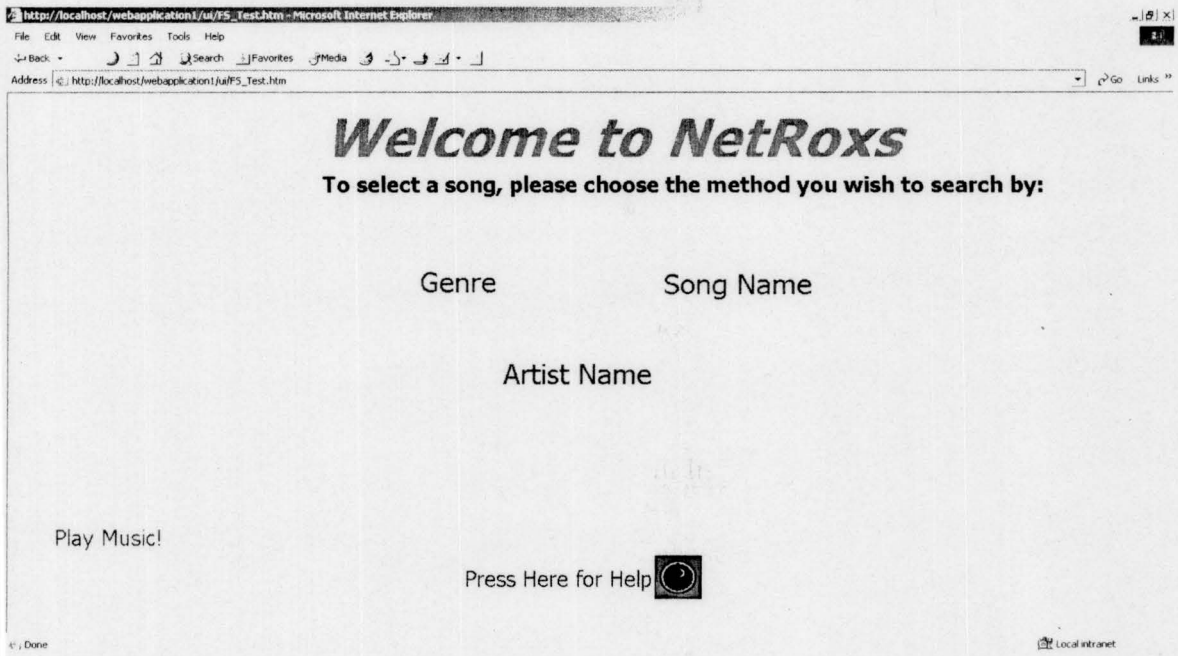
This site has a main page that the user will use to choose the method they wish to use in order to select their song. Based on their selection the user will then be taken to the Genre, Artist Name, or Song Name page. An outline of the website is below (see Figure1.)



(Figure 1)

4.3.1 Main

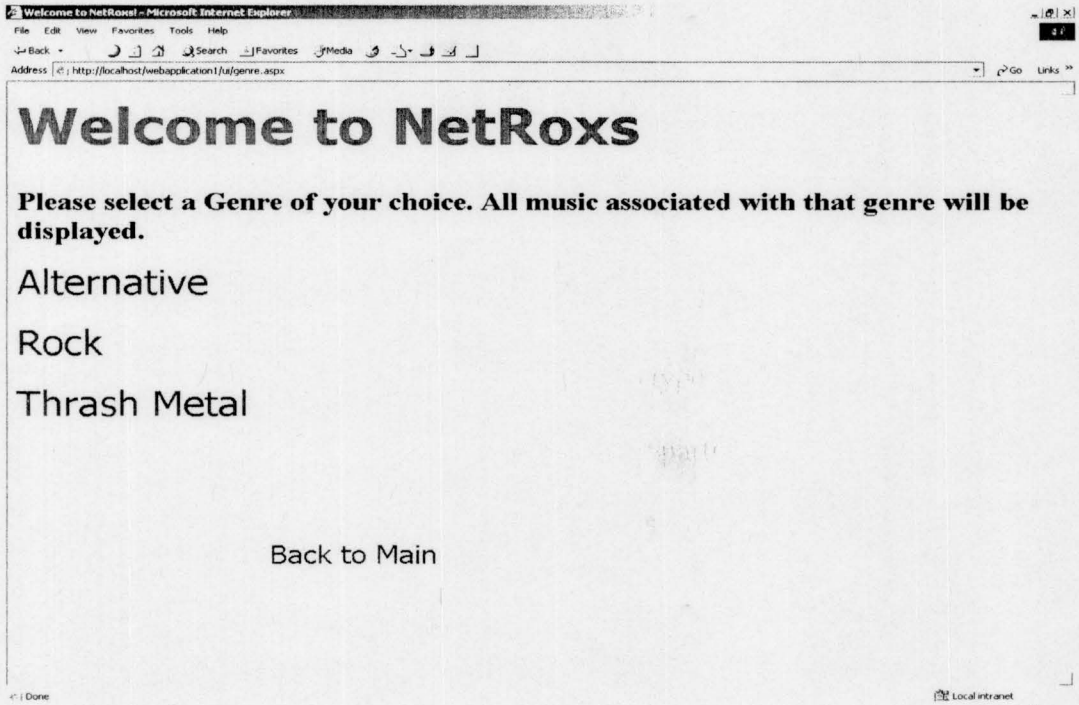
This is the first page that the user comes to. On this page is a banner indicating the name of the application, a set of brief yet clear instructions, and buttons for one of each of the options available: search by genre, search by artist name, or search by song name (see Figure 2.).



(Figure 2)

4.3.2 Genre

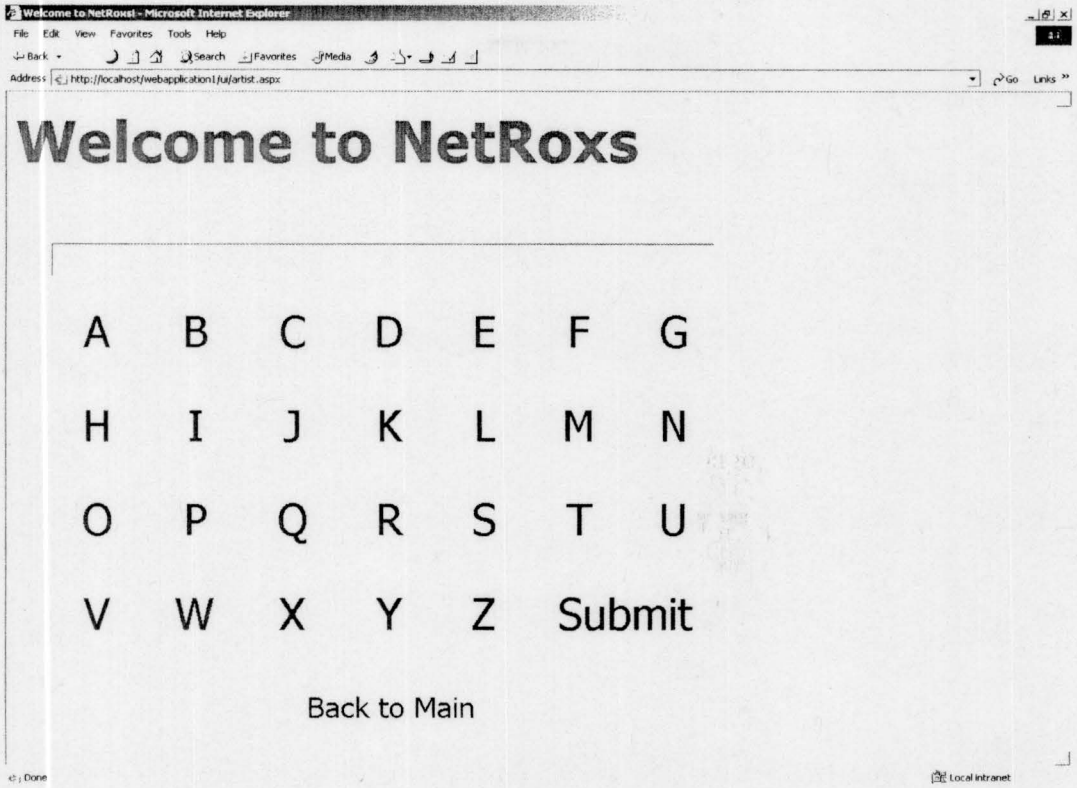
If the user presses the *genre* button the genre page will appear. This page displays all of the genres available within the database. Each genre is represented by a button that, when pressed, makes a call to the database and runs a stored procedure that will then display a new form that will list all of the song's associated by that genre (see Figure 3.).



(Figure 3)

4.3.3 Artist Name

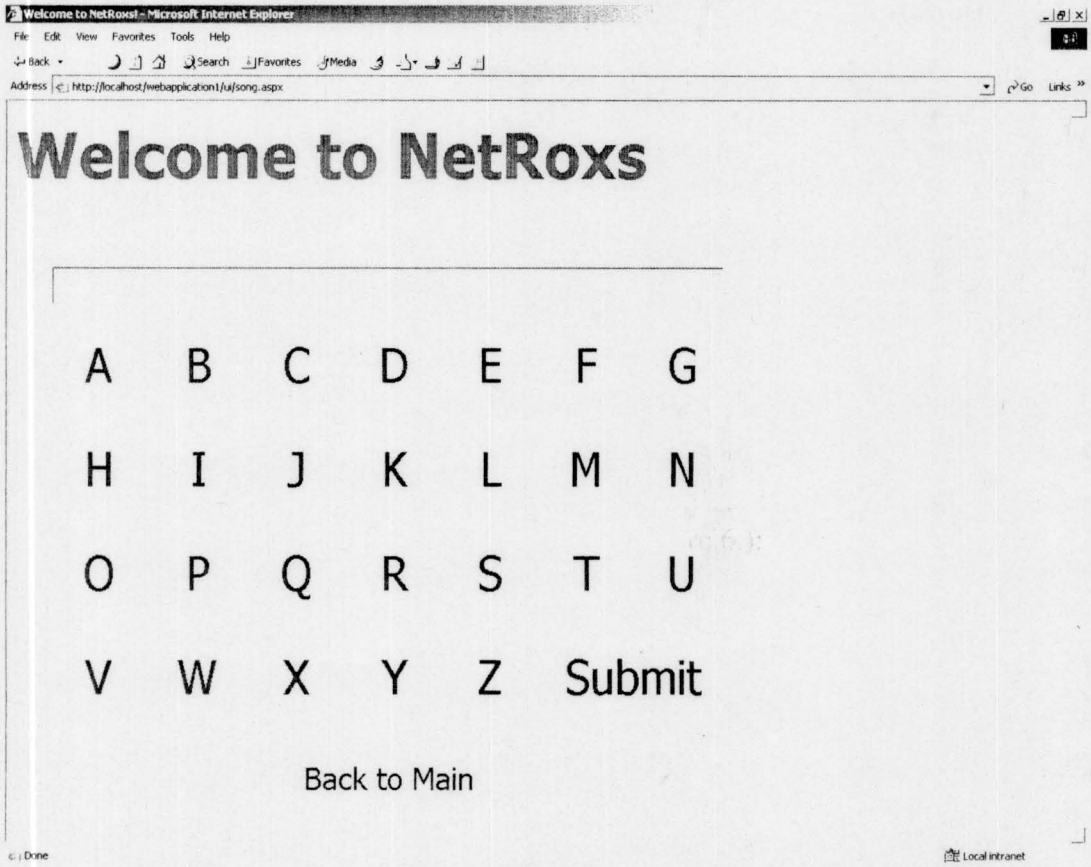
If the user presses the *artist name* button the artist page will appear. This page consists of a text box that allows the user to type in the artist's name or group that the user is looking for. This allows for partial word searches as well. Once the user has typed in their selection and press the submit button, all artists with names that include that spelling will be selected via a stored procedure and displayed on a separate page (see Figure 4.).



(Figure 4)

4.3.4 Song Name

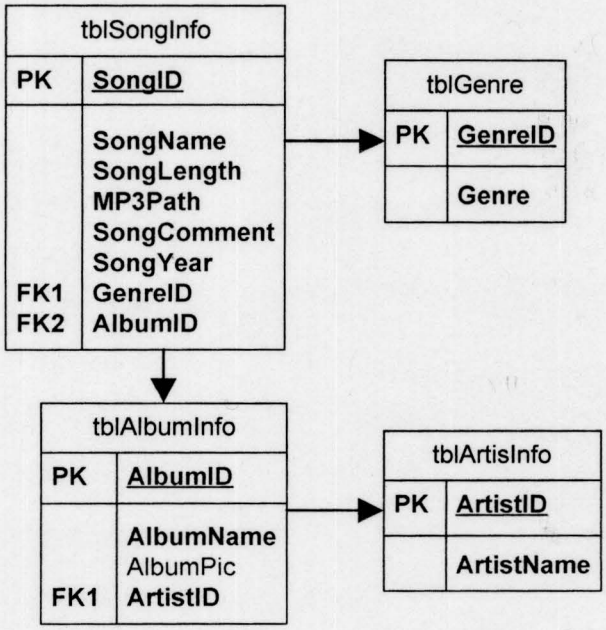
If the user presses the *song name* button the song page will appear. This page consists of a text box, similar to the artist name page, which will allow the user to type in the song name that the user is looking for. This will also allow for partial searches. Once the user has entered their selection and press submit, all songs with names that include that spelling are selected via a stored procedure and displayed on separate page (see Figure 5.).



(Figure 5)

4.4 Database Design

The database contains the following tables (see Figure 6.):



(Figure 6)

5. Deliverables

In order to display a project that was worthy of being considered for Senior Design III specific milestones were agreed upon by myself and my advisor. During the design phase of this project the following deliverables were defined:

- The user interface is written in HTML, XHTML, and C# which will provide for simple user navigation.
- Uses ASP.NET for accessing the database server and providing robust controls.
- Create a play list so the user can see what music has already been selected.
- Have a "Now Playing" field which will display the artist, song name, song length, and album information if available.
- Develop a way to track songs played and how many times each song has been played.
- Integrate useful help systems that will aide the user in the basic use of this system.

6. Design and Development

This section will describe the project's timeline and overall budget including hardware, software, and books.

6.1 Timeline

While working on this project it has been necessary to re-evaluate my timeline often. Due to various technical problems and a lack of resources for trouble shooting, it has been necessary to be as flexible as possible with deadlines and milestones.

6.1.1 Senior Design I

During Senior Design I the following tasks were completed:

- Researched the typical cost of most current jukebox systems.
- Analyzed which technology would offer the best chance for a successful product as well as provide a great learning experience.
- Began basic development design protocols.
- Developed proposal and oral presentation.

6.1.2 Senior Design II

During Senior Design II the following tasks were completed:

- Created a relational database and imported the necessary information using SQL Server 2000's Data Transformation Service.
- Began work on the user interface.
- Performed research on tools needed to construct user interface.
- Prepared Design Freeze and oral presentations

6.1.3 Senior Design III

During Senior Design III the following tasks were completed:

- Updated and modified database stored procedures.
- Finished user interface.
- Completed documentation for the project.
- Tested the design and functionality of the project.
- Prepared final paper and oral presentations.

7. Budget

The following shows the budget for all aspects of this project.

Software: Server side-

Microsoft's® SQL Server 2000 Enterprise Edition	$ 19,999.00
Microsoft's® Windows 2000	$ 349.00
Microsoft's® Visual Studio .NET	$ 1,799.00
Subtotal	$ 22,147.00

Hardware: Server side-

ATX PC Case w/300W ps	$ 43.00
AMD K7 T-Bird Socket A 266Mhz bus	$ 91.75
Deskstar 120GXP 120Gb ATA100/7200RPM	$ 329.00
ASUS A7V266 Socket A motherboard	$ 149.00
512Mb DIMM/DDR PC2100 266MHz	$ 146.00
Gainward GeForce2 MX400 SDR 64Mb (AGP)	$ 76.00
Subtotal	$ 834.75

Software: Client side-

Microsoft's® SQL Server 2000 Standard Edition	$ 4,999.00
Microsoft's® Windows 2000	$ 349.00
Subtotal	$ 5,348.00

Hardware: Client side-

Microsoft's® Windows 2000	$ 349.00
ATX Desktop Case	$ 76.00
Deskstar 120GXP 40Gb ATA100/7200RPM	$ 91.99
Netgear FA312 10/100 Pci NIC RJ-45	$ 18.99
15IN Capacitive Touch Screen LCD	$ 864.06
MSR210 Credit Card Reader	$ 87.00
PC Charge CC Software	$ 395.00
Subtotal	$ 1,882.04

TOTAL (software/hardware) $ 30,211.79

Books-

Professional C#	1-861004-99-0	$60.00
ASP.NET with C#	0-7821-2989-7	$49.99
Programming C#	0-596-00309-9	$39.95
Programming ASP.NET	0-596-00171-1	$49.95
Sams Teach Yourself Microsoft SQL Server 2000	0-672-31969-1	$39.99
Sub Total		$239.88

Grand Total $30,451.67

8. Proof of Design

This section shows how the deliverables were fulfilled and the types of challenges that were encountered.

8.1 User Interface

The main concept behind this project was a jukebox which would be placed in areas where individuals of all types of computer skills will have access. Also, this was not intended to be entertaining in and of itself but to simply provide an easy to use interface to access the entertainment. This was achieved

utilizing HTML, HTML Frame Sets, JavaScript, and C#. Also, the style of all the buttons and links consisted of a Tahoma font with blue coloring and a font-size no less than 14pt. The blue color was chosen since it is familiar with most users as being a "clickable" object on most web pages as well as its contrasting affect against the dark gray background.

8.2 Uses ASP.NET for Accessing Database

A main reason for deciding to ASP.NET for this project was due to its' integration of SQL libraries in coordination with ADO.NET's rich data access capabilities. This provided for an efficient and fast connection to the database. In addition, using Visual Studio.NET to create database connections was a breeze. Below is an example of Visual Studios connection string for accessing a SQL Server 2000 database:

data source=HOME2;initial catalog=Mitchell_SD;integrated security=SSPI;persist security info=True;workstation id=HOME2;packet size=4096

Visual Studio's IDE (integrated development environment) makes processes such as this quick and easy. However, using Visual Studio to create such links makes the task of moving a project to another computer quite complicated. As a rule of thumb it is best to create SQL connections programmatically.

8.3 Creating Play List

The third item of the deliverables consisted of creating a working playlist that would show users 1) what song was playing currently and 2) what songs have been queued to be played so that other users are not likely to waste money on a

song that they are going to hear anyway. This task proved to be quite complicated and eventually required a complete overhaul of the original design specifications.

My intent was to create a method that would produce a playlist based upon a user's particular song selection. Ideally, this playlist would then be fed to an embedded Windows Media Player (WMP) which would then begin playing. However, I was unable to find (through research and inquiries) a useful way for my server-side to communicate with my client-side. In addition, a large amount of time was used trying to find a way to dynamically create buttons. The need for creating buttons dynamically was driven by the usefulness of having an "onclick" method for which it would be possibly to call methods and manipulate a playlist. This is not to imply that a solution was not found. Through long hours and help from faculty and friends a way was developed that would create a playlist and have Windows Media Player recognize and play the list.

The solution was found partly through the WMP SDK and various online resources as well as the realization that ASP.NET's DataGrid offers some of the "onclick" functionality I needed. Basically, by creating a file with the appropriate header I could, programmatically, wrap the song name, artist name, and mp3 path in xml tags and insert them into an .asx file which WMP recognizes as a Windows Media metafile. This provided a means of creating a playlist as well as having WMP recognize and play the selections. However, this method severely limited the original functionality of my design. For example, WMP can only read one .asx file at a time. This, obviously, prevents multiple playlists from being used. Also, WMP stores the playlist directly into the browsers cache which makes

manipulating it near impossible. Currently, the only method that I have found is by manually deleting the .asx file as well as the browsers cache. This is definitely not part of my original design. See the section *Conclusions and Recommendations* for thoughts on how to improve this process.

8.4 Track Songs Played

The reason for this particular deliverable is mainly for billing purposes. Currently, the traditional jukebox is covered by the Jukebox License Agency which, in turn, is controlled jointly by ASCAP, BMI, and SESAC (see section 2 for more information). The fees generated by these licenses are a flat fee since there is not an easy way to determine which songs get played the most. By tracking which songs get selected it is possible to generate reports that show which artists are selected more frequently than others, which songs are selected more frequently than others, as well as which artist is selected the most. This can then be used to devise a better system for paying the artists that get played the most. Another plus side to tracking song selection would be to ensure that the basic song selection locally is actually what the patrons of that establishment want to hear.

To write this information to the database required a new table being created. I named this table "Selected" and chose to insert only three columns of data.

Selected	
PK	SongSelectID
FK1	DateSelected SongSelected

(Figure 7)

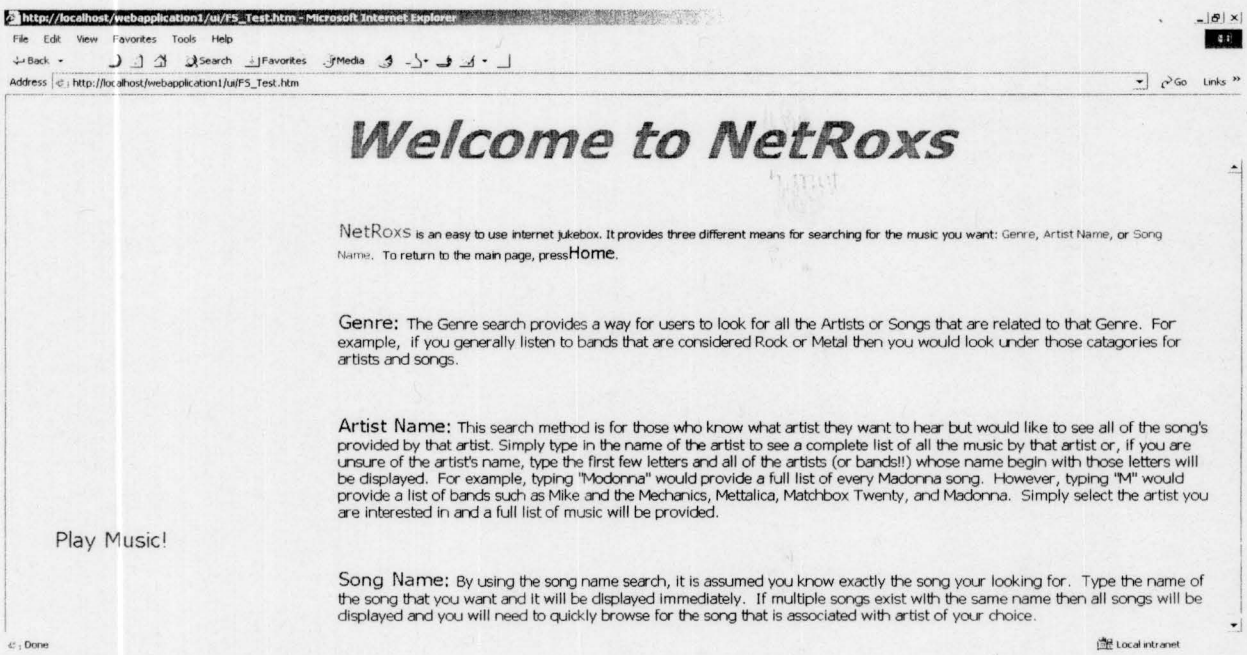
An example of the code necessary to make this process happen is below:

```
string scUpdate = ("Insert into Selected (SongSelected) values (" +_
songid + ")");

SqlCommand cmd;
cmd = sqlConn.CreateCommand();
cmd.CommandText = scUpdate;
cmd.ExecuteNonQuery();
```

8.5 Help System

The last deliverable required was a useful help system that provides the user with a basic description of how to use the NetRoxs jukebox system. This was achieved using HTML exclusively (see Figure 8).



(Figure 8)

The page offers a simple description of how each method for searching is used and provides links for getting to the section the user desires as well as back to the home page.

9. Testing

Testing for this project was done primarily on win2000 with 1GHz AMD processor, 256Mb of RAM, integrated AC97 sound on a MIS K7S5A mainboard and a 32Mb GeForce video card. Minor testing was accomplished over the internet from users dialing in to the local boxes IP and accessing the system.

10. Conclusions and Recommendations

10.1 Conclusions

This project proved to be more advantageous than initially thought. I did not have any working knowledge of ASP.NET and only minor experience with C#. This led to many issues regarding the actually intent of the project versus the model for how ASP.NET is designed. I was looking for a client side interface that would have access to a given number of mp3's while having access, through the internet, to thousands of other mp3's. I chose to use ASP.NET because it seemed to me that ASP.NET was, basically, windows programming for the internet. This turned out to only be partially true. After months of hard work and research I came to the conclusion that this was not the right technology to use for this project.

However, all was not lost with this project. Through much trial and error I can now say that I have a strong working knowledge of ASP.NET and how it is used to bring rich content over the internet. I have also become pretty knowledgeable concerning SQL Server 2000 and C#. All of these technologies were fairly new to me when this project began.

10.2 Recommendations

For anyone else wishing to create a project such as this I would recommend developing using windows forms instead on internet based technologies. Windows' forms allow more control over client side issues and still have strong controls for accessing servers across the internet. I think this project would have been best as a Windows' distributed application than as an internet based application. The biggest limitation of ASP.NET as it relates to this project is the inability to control client-side information with server-side code. This was the single biggest reason for the issues concerning this project. Huge amounts of time and effort were wasted in the pursuit of this single obstacle which, ultimately, proved to be the deciding factor in the limiting functionality of this project.

Given all of the many issues that I encountered in developing this project I would first like to mention that it was an experience that taught me a great deal in the way of software development and working with bleeding edge technologies. However, there is a reason why they call them bleeding edge; you often get cut and have no means to stop the bleeding but, you also get great experience and knowledge of products and technologies that few others have seen. To me, this is what learning is all about.

Appendix A

Code Snippets

A1. Accessing the Database

The primary thing that each page must do is pass whatever parameter the user has chosen to the database so that the next group of choices can then be selected. This is done through each page's Page_Load event. All of the code within this designation is ran the moment the page is called. Below is an example:

```
private void Page_Load(object sender, System.EventArgs e)
{
      if (!IsPostBack)
      {
            string aselection;
            aselection = Request.QueryString.Get("aselection");

            sqlConn.Open();//Opens SQL connection
            SqlCommand cm = new SqlCommand("EXEC GetArtistSelection '"
+ aselection+"'", sqlConn);//Passes in parameter and stores command for
later use

            DataSet ds = new DataSet();//Creates new DataSet
            SqlDataAdapter da = new SqlDataAdapter();//Creates new
DataAdapter

            da.SelectCommand = cm;//DataAdapter calls command
            da.Fill(ds, "Artist_Selection");//and fills the DataSet
with the results

            dgAS.DataSource =
ds.Tables["Artist_Selection"].DefaultView;//Points the DataGrid to the
Dataset
            dgAS.DataBind();//Binds the data grid with the dataset

            da.Dispose();
            sqlConn.Close();


            Response.Write("<h2>Please select the Artist or Group of
your choice.</h2>");
```

```
            //The foreach statement runs through the Dataset and prints
a hyperlink for each row in the dataset.  This is also where it passes
on the information for that link.
            foreach(DataRow row in ds.Tables["Artist_Selection"].Rows)
            {
                  Response.Write("<a style='text-decoration:
none;color: blue; font: 18pt Tahoma;'
href='./artistsongselection.aspx?aname=" + row[0] + "'>" + row[1] +
"</a><br>");
            }
      }
}
```

A2. Creating the .asx File

Once the idea was discovered that the .asx file would be recognized by Windows Media Player it became necessary to develop a way to "wrap" what the user selected into a format that the player would understand. The format of an .asx file is XML. Using C#'s ability to create XML data I was able (with help) to devise the method below that achieved the proper results:

```
XmlDocument pl = new XmlDocument();
XmlElement root;

public void load(string filename)
{
      //LoadXML
      pl.Load(filename);
      root = pl.DocumentElement;

}

public void savePlaylist(string filename)
{
      pl.Save(filename);
}

public void addEntryToPlaylist(string path, string song, string artist)
{
      root.AppendChild(createEntry(path,song,artist));
}

public XmlElement createEntry(string path, string song, string artist)
{
      XmlElement entry = pl.CreateElement("entry");

      entry.AppendChild(pl.CreateElement("title"));
      entry["title"].AppendChild(pl.CreateTextNode(song));

      entry.AppendChild(pl.CreateElement("author"));
```

```
        entry["author"].AppendChild(pl.CreateTextNode(artist));

        entry.AppendChild(pl.CreateElement("ref"));
        entry["ref"].SetAttribute("href",path);

        return entry;

}
```

A3. Writing to the Playlist

This example comes after many long hours attempting to solve the actual playlist problem. Once the .asx file solution presented itself it was only a short time to figure out this code. C# comes with an extensive library of XML classes and objects which make creating and reading XML a breeze. Below is the code snippet that actually reads through the .asx file and prints to a listbox all of the song selections within. This code was placed behind the "Play Music" button on the NowPlayingPage page.

```
private void Button1_Click(object sender, System.EventArgs e)
{
        doc.Load("C:\\User Interface\\WebApplication1\\ui\\playlst.asx");
        XmlNodeList nl = doc.GetElementsByTagName("entry");

        foreach(XmlNode node in nl) lbPlaylist.Items.Add(node.InnerText);
}
```

Bibliography

1. "C# Corner". http://www.c-sharpcorner.com/

2. "C# Help". http://www.csharphelp.com/

3. "Microsoft's MSDN Home Page". http://msdn.microsoft.com

4. "ROWE International Home Page". http://www.roweami.com

5. "SQL Interpreter & Tutorial". http://sqlcourse2.com/

6. "Jukebox Licensing Office". http://www.jukeboxlicense.com/

7. Ciresi, Sam. Sales Representative, Pioneer Vending Company. Personal Interview. February 27, 2002.

8. Hoffman, Matt. Teachers Aid, Information Engineering Technology Dept., Ohio College of Applied Science. Personal Interviews.

9. Hurwitz, Dan and Jesse Liberty. *Programming ASP.NET.* O'Reilly Publishing, 2002.

10. Ladd, Eric, et al. *Platinum Edition: Using XHTML, XML, AND JAVA 2.* QUE Publishing, 2001.

11. Liberty, Jesse. *Programming C#.* O'Reilly Publishing, 2002.

12. Robinson, Simon, et al. *Professional C#.* Wrox Press Ltd., 2001.

13. Stroh, Jason. Computer Lab Head, Ohio College of Applied Science. Personal Interviews.

14. Waymire, Richard and Rick Sawtell. *SAMS Teach Yourself: Microsoft SQL Server 2000 in 21 Days.* Indianapolis, IN: Sams Publishing, 2001.