A Web-based Approach to Music Library Management

Jason Young
California Polytechnic State University, San Luis Obispo
June 3, 2012

Abstract

This application utilizes modern standards developing in web browsers to create a desktop-like application for managing a music library online. The server side application is written in a model-view-controller style using Python backed by a MySQL database. The client side JavaScript was designed around a modular concept interfacing several third-party frameworks and libraries. By taking advantage of developing browser features such as persistent local storage, this web application was designed to handle a large library of music.

Introduction

Over the last decade, the Internet has transformed our use of operating systems drastically. A large majority of time is spent using a web browser instead of native applications. Since a website can be accessed from any computer connected to the Internet, the number of applicable uses for the web has grown exponentially. Gone are the days where you needed a Windows machine to run a word processor or Mac OS to design and edit videos. With new technological standards of HTML and ECMAScript being developed for use in modern web browsers, websites can now look and function just like a standard desktop application.

Applications for managing and playing music have traditionally been desktop-based. This limits the ability to share the library or listen to it when away from the computer. My web application, called CloudMusic, addresses these issues by moving everything to the web. The application is written in Python using the Django framework for the server-side, while the client-side is composed of a modularized JavaScript framework. Python was chosen because it is a very expressive, concise and clean language, and Django's model-view-controller structure organizes the code to create a manageable and intuitive codebase. Python also has the advantage of having large range of libraries easily imported into the code, with Django adding even more. A modularized JavaScript framework was chosen for the client-side to support scaling.

The system works by first loading the application's dynamic front-end. This is done by querying the server for a list of tracks in the database, list of artists, and list of playlists. While the user interacts with the site, the display information is kept up-to-date, such as the current album art and track listings. The user is also capable of creating and modifying playlists, searching and filtering songs, downloading songs, posting to the community chat box, and playing songs.

Similar Concepts

There are two concepts for an online music management system: a system where users do the setup and maintenance of their server, and a system where a third party handles everything for the user. Both have benefits, but some people would not feel comfortable with someone else having access to their media. My preference has always been to self-management of my own music.

An example of a third-party system is Google Music, which allows users to sync their songs to Google servers and play them via a web-based interface, as well as mobile devices [1]. However, due to piracy laws, certain desirable features, such as sharing music with friends is not allowed. Google also then has complete control and knowledge of your library, as well as saving your usage statistics to suggest new music. Users that feel this is too much of an invasion of privacy have the option of a self-hosted solution, such as this CloudMusic application.

With the explosion of web-based applications, it is not surprising that there are many other music management websites already in existence. The concept for CloudMusic came from my attempt at moving my large music collection online so I could use it anywhere. While researching existing options, I discovered kPlaylist, "a music database that you manage via the web" [2]. After installing and running it, I

ran intro problems. While kPlaylist initially succeeded, my collection grew extremely large and kPlaylist could not properly scan and display my large list of songs in a timely fashion. Since kPlaylist was written in PHP and no longer maintained, I searched for other solutions. I found similar applications such as Ampache [3], but its user interface was not efficient and not fit for larger collections. I have always favored the iTunes interface for browsing music and could not find anything similar.

These failed attempts at finding a suitable system that could handle a significantly large music collection lead me to design my own. The first version of my application was written using Kohana, a PHP model-view-controller framework for the server-side, and JavaScript for the client-side [4]. The site successfully operated for about a year, gathering nearly fifty of my friends as users. However, as the application's data storage and features grew, so did the codebase, and thus I realized I needed new technologies to properly implement this idea.

Before starting from scratch on this new application, I again researched new alternatives. A second look at kPlaylist revealed that the project has stopped development indefinitely. Ampache now has changed their application strategy and is no longer a solely web-based interface for music management. Another application discovered is the Voodoo Music Box. Their website states "Voodoo Music Box is a web based MP3 library management application [5]. The program is written in PHP using the Zend Framework and a MySQL back-end. Voodoo Music Box has many features other web based MP3 software application do not have" [5]. Nowhere does it mention efficient support for large music collections. While this initially sounded appetizing, I had several concerns: it is written in PHP, utilizes the Zend framework, and is in the alpha development status [6]. Also, while analyzing its codebase, I found the organization of the project to be all too similar to my first attempt: due to the PHP Zend framework, it contains an enormous number of folders and class files which makes development rather challenging, as well as containing too many separate libraries that have to be externally included, unlike Python. Also, due to the constant updates to the PHP language, compatibility with different servers creates unnecessary challenges. Python's development, on the other hand, is rather stable and updating a server to a newer version rarely breaks existing code. Aside from the backend issues, the interface was nothing like the iTunes format I desired. Lastly, from my several years in PHP development, I dislike the rather messy code formatting and style, so I did not want to work with such a language anymore. This led me to research other languages and frameworks that could accomplish my task in an organized and efficient manner.

Application Features

The main features in the initial release of CloudMusic are based off my previous application, as well as important design ideologies used in iTunes. The following features are divided into two categories: the features that affect the user experience and the features that are part of the code design.

User Experience Features

- Single Page Layout

Unlike traditional websites, modern browsers and web applications are migrating to dynamic content. By utilizing AJAX, or asynchronous JavaScript and XML, a single page can request and load data from the server without ever leaving the current page. The benefit of this is reducing server load. Without dynamic content, each time a link is clicked, the server has to render the HTML for the new page which is usually wasteful since much of the code in the pages of a website, such as the document's head, is identical. When the site is loading new content, a loading animation appears to notify the user.

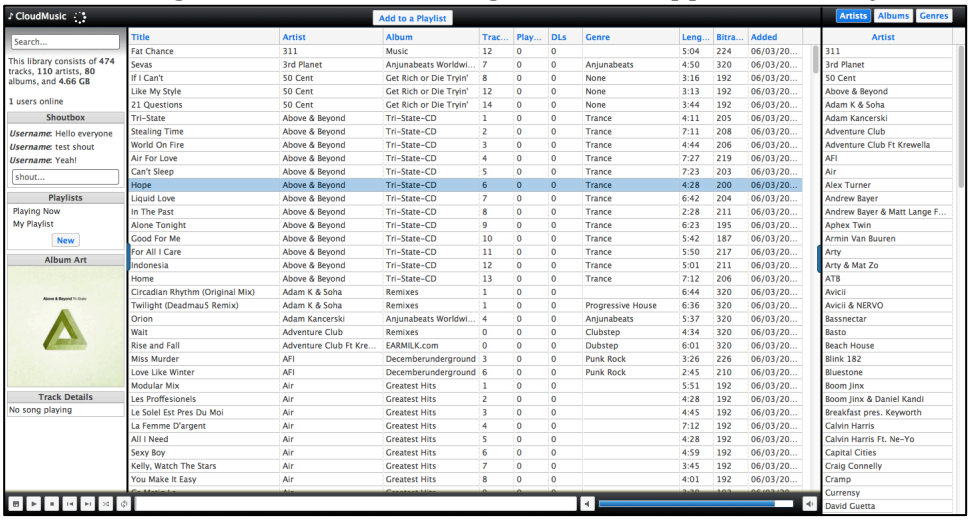

Figure 1: Single Page Layout

- Artist, Album, and Genre Filtering
  Inspired by many existing music library management programs, the interface contains a list for every artist, album, and genre that exists in the database. Upon selecting items from these lists, the main grid of tracks is filtered to match the selection.

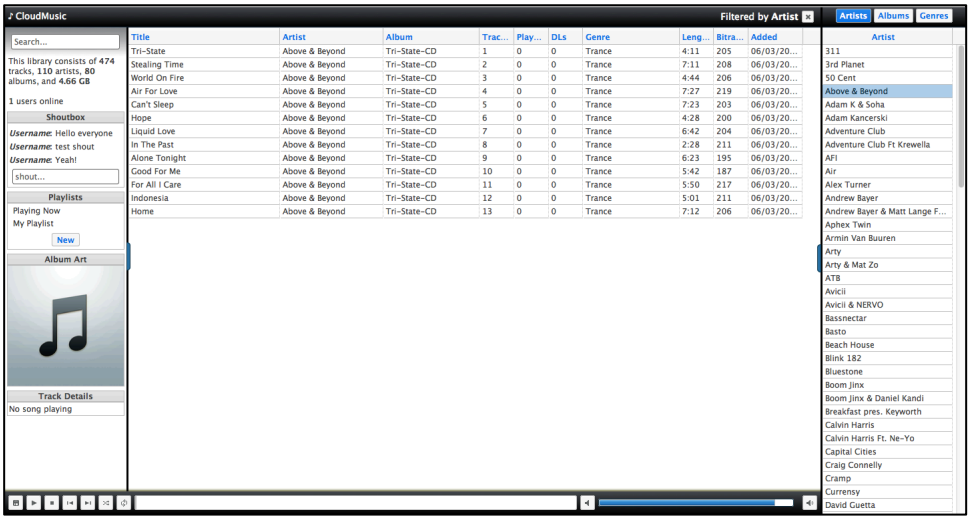

Figure 2: Filtered by Artist

- Sorting
  Each grid has the ability to be sorted by columns.

- Searching
  Searching is a crucial aspect to dealing with any sort of data, especially a music collection. The interface contains a search box that will filter the music grid for the search terms provided.

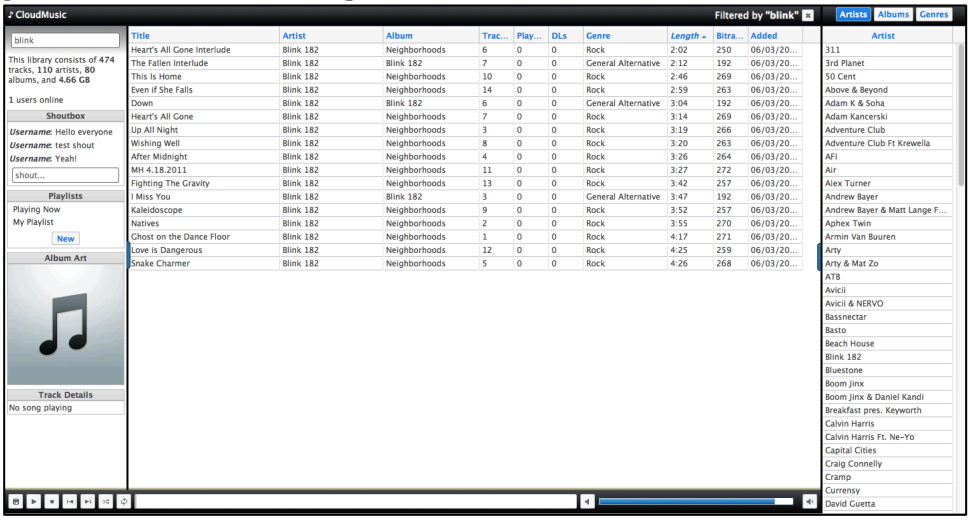

Figure 3: Searching "blink"

- Play controls
  As with any music player, controlling the currently playing song is a necessity. The interface implements a player bar that has buttons for stopping, playing or pausing, increasing the volume for, and muting the current song. It also has buttons for playing the previously played track and the next track. The bar also contains buttons to turn on playback shuffling, as well as playback repeating. The player bar also has a slider for showing the current song's progress and for seeking in the song. While the track is buffering, the background of the bar slides to show the progress of the loading. There is also a volume slider for changing the volume. Finally, this play bar also displays the name and time left for the playing song.

Figure 4: Player controls

- Downloading
  Since Internet access is not everywhere and some people like to have a copy of the songs on their own hard drive, the interface contains a button that creates a zip file of the selected songs and allows the user to download it.

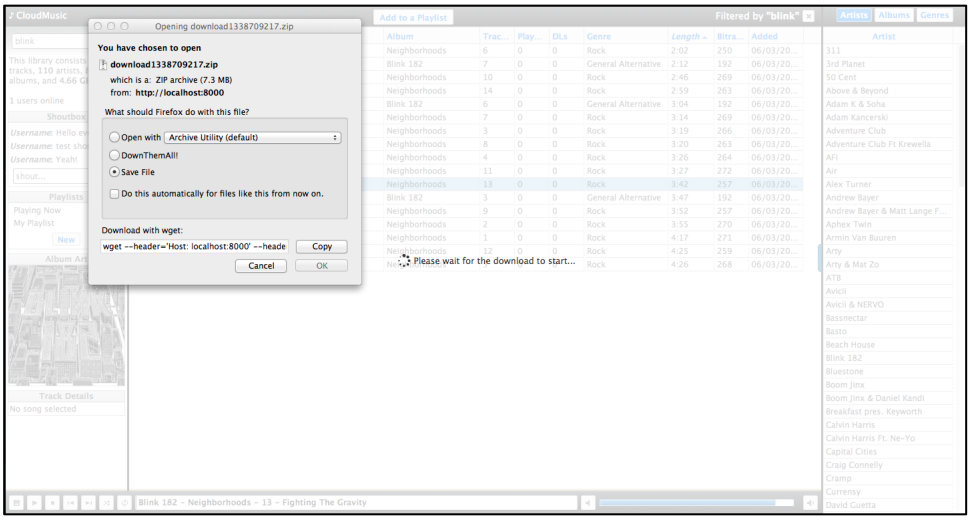

Figure 5: Downloading a track

- View site statistics
  The sidebar of the interface shows a brief overview of library's statistics, including the number of songs, artists, albums, genres, total users, total users online, and the amount of disk space used by the library.

- View ID3 tags
  Since the database does not need to store extended information found in mp3 ID3 tags, the interface's sidebar shows all the tags present in the playing song. This is useful for discovering the URL for the song's source, for example.

- Chatting ability
  Due to the immense increase in social features appearing on websites, the interface includes a small "shoutbox" that allows users to post messages for everyone to see.

- Playlist management
  Crucial to any music management system, playlists allow users to organize their music based upon their own categories. The interface has controls to create and delete playlists, as well as to add and remove tracks from them.

- Local playlist management
  Like many other music management programs, this interface allows users to create a non-persistent playlist that is good for quickly creating a list of songs to play, for instance. This playlist is labeled "Playing Now," and functions like any other playlist.

- View album art
  When a song is selected, the "Album Art" sidebar widget updates to reflect the album art for the particular song, if it has one.

Design Features

- Dynamically-loading grids
  For large music collections online, loading the list of songs can take a considerable amount of time and memory. To reduce the load time and memory usage, the grids load the data in sections with each section only being loaded when it is scrolled into view.

- Dynamic page layout
  A consistent issue when creating page layouts is the difficulties in ensuring the page will display on different sized monitors, as well as retaining the layout when the browser is resized. To handle this, a JavaScript layout library combined with an auto-sizing function ensures the application will keep its proportions.

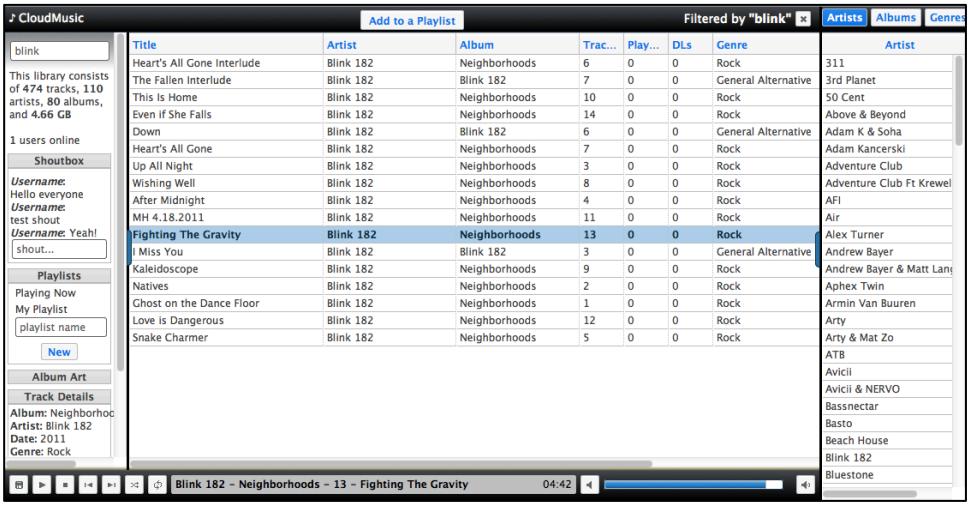

Figure 6: Resizing window keeps layout

- Persistent data storage
  The constant loading of rows of songs takes time and server processing time. With the new standards evolving in HTML, my application takes advantage of the offline data storage capabilities found in the HTML5 standard. By storing each request in the browser's local storage, the grid can reload without waiting on the server, dramatically increasing rendering time. Since the data is persistent, it exists even if the browser is closed.

Design Information

After analyzing the similar applications, I decided on using Django and MySQL for the server side. Django is a framework written in Python and follows the Model-View-Controller design paradigm. In short, this means that components of the framework are organized into models, views, and controllers. The concept being that models should contain all the data manipulation functionality and should notify when data has change. Each model should only represent a single piece of knowledge [10]. For example, my application contains a model called Track that

manages only the data manipulation for tracks. Models are usually the component that interfaces with the database. For storing data in the database, Django contains an advanced ORM, or object relational mapping, for abstracting database access. This makes code easily readable and allows for any database management system to be used without changing code. My implementation uses MySQL for ease of use, but future versions might swap it out for a faster system. The view component abstracts the display aspect of the application. In this case, the views render the output to the browser. The controller component takes in the request parameters and determines which views to render. In Django, however, the controller is simply a mapping of URLs to views. When a URL is requested, the framework checks each regular expression mapping in the controller file to find a match. When it finds a match, it passes the request on to the corresponding view, which might utilize models before rendering the output.

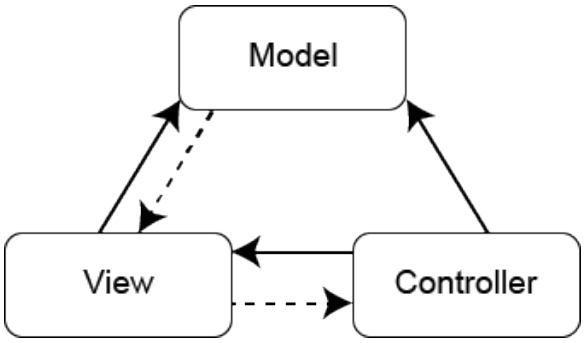

Figure 7: MVC components

On the client side, or in the browser, the design hinges around the object-oriented nature of JavaScript. The technologies used on the client side are jQuery, SoundManager2, SlickGrid, and dynamically loading JavaScript modules. Each of the models is a custom JavaScript object that interacts with the page. jQuery is "is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development" [7]. Using jQuery abstracts browser-specific JavaScript API differences to ensure cross-browser compatibility as well as adding useful functionalities in building websites that are not found in the JavaScript standards. SoundManager2 controls audio playback by using HTML5 and Flash, providing reliable cross-platform audio under a single JavaScript API, which is perfectly suitable for a web-based media server [8]. SlickGrid is "a lightning fast JavaScript grid" that allows loading of data from a remote source [9]. The non-third party models written for this application are:

- An AJAX controller for handling server to client communications
- A grid controller that initializes and interfaces with the SlickGrid objects on the page
- A layout model which initializes and controls the page's layout
- A chat model for handling the chat box operations
- An audio model for interfacing with SoundManager2 and controlling playback
- An interface module for handling element-specific functionality
- A playlist model for managing the site's playlists,

- A storage model for interfacing between the browser's persistent storage capabilities.
- A grid loader model for interfacing between the grids and the server

Together, these JavaScript objects control the functionality of the application in the browser.

Implementation

On the server, Django is organized by projects and applications. Inside each project there can be many applications. There are three applications part of this project as a whole. The main application merely controls the loading of the initial page content. The library application controls everything related to managing the music library. There are four main functionalities part of the library app. The first are view methods for retrieving the track information required by the grids. When the grid on the page requests the list of songs, the library view calls the Track model and filters the track objects by the parameters passed in. The view then renders the track objects as a list that the SlickGrid can display. The second view methods are for streaming a track's data to the SoundManager2 audio player. This method creates a file streamer class that bandwidth-limits the sending of audio data to preserve bandwidth in the case that the user plays a different track right after loading a different one. The third view method of the library application is for retrieving album art for a track. If there is an image file found in the same folder then the view will return this image. If not, the view will open the music file and grab the embedded album art in the ID3 tags and return that. Lastly, the library application also contains the scanner method that traverses the music directory for songs and adds them into the database via the library models. The third application manages the many CSS style sheet files and JavaScript files for the page by combing them into a single source for faster loading by the browser.

On the client side, the functionality of the application is based on the event-driven nature of browser JavaScript implementations. When the application is first requested and the server returns the page, the individual JavaScript components load sequentially, starting with jQuery and jQuery extensions. Next, SlickGrid and SoundManager2 are loaded. During this initialization, each SlickGrid is assigned a model that controls which URL to get data from. Lastly, when the document-loaded event is fired, all the custom models are instantiated. During these instantiations, elements on the page are bound functions calls for different events. For example, the play button on the page is bound to the audio model's play function when its "click" event is fired. Each model has its own methods and properties that work together to allow the application to function.

Conclusion

In conclusion, the CloudMusic web application is an approach to managing music libraries on the web by utilizing the powerful features of modern browsers. As these browsers evolve, they become more capable of supporting websites that function like native programs. The CloudMusic application allows the user to manage their music from anywhere that has Internet access. This allows the user to share music

between friends, as well as to never worry about losing your music if your computer is stolen, for example. Although there are similar web applications that exist, most are either managed by a third party or don't support large libraries.

References

[1] *Google Music*. Retrieved from http://music.google.com/.

[2] *kPlaylist - get your music online today! PHP based media streaming system and music database*. Retrieved from http://www.kplaylist.net/.

[3] *Ampache :: For the love of music.* Retrieved from http://www.ampache.org/.
http://kohanaframework.org/.

[4] *Voodoo Music Box.* Retrieved from http://sourceforge.net/projects/voodoomusicbox/.

[5] *Zend* Framework. Retrieved from http://framework.zend.com/.

[6] *jQuery: The Write Less, Do More, JavaScript Library.* Retrieved from http://jquery.com/.

[7] *SoundManager 2: JavaScript Sound For The Web.* Retrieved from http://www.schillmania.com/projects/soundmanager2/.

[8] *mleibman/SlickGrid.* Retrieved from https://github.com/mleibman/SlickGrid.

[9] *Model-View-Controller*. Retrieved from http://ootips.org/mvc-pattern.html