Version 1.0

August 13, 2015

Presented by: Fazlay Rabbi & saad makhdumi



music portfolio

csci 499 – hunter college, cuny

## overview

It provides a platform for the upcoming musicians to display their work in a single profile. This application will not only allow them to upload their work to this application but will use APIs to bring their profiles from other social media forums into one single profile.

## Target Market

This product is mainly for upcoming musicians who still have to make a name for themselves and need a tool to market their talent.

## Main features

This product is available both on the Web and Mobile\* platforms. Main features of this application are:

**Search & Explore:** This is the home page for this application, it lists the top musicians in the system based on number of views of their profiles.

**Biography:** Musicians who create an account on this site, can edit their profile and update their biography, they also have the option to add contact info for their agents (if any) and on the same page and they can upload their music audio and video files into the system.

**Social Media Integration:** Musicians have the option to link their Music Portfolio profile with other social media sites listed below. Upon link request, users are prompted to login to these social media sites via APIs and they have to authorize ‘Music Portfolio’ to access their data. All access to social media sites is driven by APIs and user credentials for social media sites are never stored in our system. At the moment following social media sites are supported by our system.

* SoundCloud
* Facebook
* Twitter
* Instagram

\* Currently mobile application is available only for Android.

## Technologies

|  |  |  |  |
| --- | --- | --- | --- |
| Area | Component | Technology | Version |
| Backend | Database | Microsoft SQL Server | 2012 |
| Backend | Server Side Code | C# |  |
| Backend | Web Server Operating System | Windows Server | 2008 |
| Frontend | Client Side Page Rendering | HTML5 / ASP.net |  |
| Frontend | Client Side Scripting | JavaScript |  |
| Mobile | Android | Java | 8 |
| Development | ASP.net Coding | Visual Studio | 2013 |
| Development | Android | Android Studio |  |
| Development | Platform | Microsoft .Net | 4.5 |
| Hosting | Web Server | IIS | 7.5 |

## Team

|  |  |  |
| --- | --- | --- |
| Name | Email | Phone Number |
| Fazlay Rabbi | [Fazlay.rabbi35@myhunter.cuny.edu](mailto:Fazlay.rabbi35@myhunter.cuny.edu) | 347-656-0127 |
| Michael Cardenas | [Michael.cardenas389@gmail.com](mailto:Michael.cardenas389@gmail.com) | 917-254-2582 |
| Saad Makhdumi | [Saad.makhdumi38@myhunter.cuny.edu](mailto:Saad.makhdumi38@myhunter.cuny.edu) | 646-823-5175 |

## responsibilities

|  |  |  |
| --- | --- | --- |
| Team | Primary Tasks | Secondary Tasks |
| Saad Makhdumi | Server Side Coding, C#, Web API for Android | Database, Frontend ASPx |
| Fazlay Rabbi | Frontend ASP.net, JavaScript, Style Sheets, Android | Database, Server Side Coding |
| Michael Cardenas | Database Design, MSSQL | C#, ASP.net |

## Weekly progress report

|  |  |  |  |
| --- | --- | --- | --- |
| Week | Milestone | Details | Status |
| 1 | Complete initial project draft | Define project scope, build a team, agree on features and technologies and define tasks and responsibilities for each team member. | Completed on time |
| 2 | Database and Site’s skeleton | Finish initial database design, create a basic website to allow user logins and signups. | Completed on time |
| 3 | Facebook integration / Audio & Video uploads | Allow user to bring his Facebook timeline into our site. Also user will be able to upload audio or video files. | Completed on time |
| 4 | Twitter / Sound Cloud integration / Asp.net API for android app | All user to bring his twitter and sound cloud profiles into our site. Get asp.net backend api ready, mobile app will use this api to interact with our database. | Completed on time |
| 5 | Android App | Initial layout design and basic communication framework. | Completed on time |
| 6 | Android App | Add functions to backend API for Instagram, Soundcloud, Facebook and twitter.  Add functionality to android app to display data for these social media sites. | Completed on time |
| 7 | Finish Coding and Start QA | Almost finished with coding, some minor style sheet enhancements are pending. | Completed on time |
| 8 | QA / Fix Bugs | QA is in progress | Completed on time |
| 9 | QA / Fix Bugs | QA is in progress | Completed on time |
| 10 | Documentation & Presentation |  |  |

## website Technical details

Website is using HTML5 and ASP.net on the frontend. It uses standard .net System.Data.SqlClient library to connect to Microsoft SQL server backend.

Backend Authentication

Website to database connection is using Windows Integrated Authentication.

Each of the social media site required its own authentication and presented data in a different format, so for these we had to use different libraries.

### Facebook

First step for Facebook integration is to register our application with Facebook and request permission to pull end-user’s profile posts. Upon registration with Facebook, an application ID and a secret keys are provided. This is essentially our application’s user/pass pair to talk to Facebook API.

Once user (musician) initiates the Facebook integration process, a request is send to Facebook using their API to get the login prompt for user. This login prompt request contains several parameters such as application ID (which we retrieved during our application signup process), a redirect URI (this must be a publicly accessible site), type of access required etc.

Upon successful login by the user into Facebook login prompt, API redirects the user to the redirect URI we provided in our request. Embedded in the redirect URI is a query string parameter that contains the “access code” for that particular user.

The page that serves as the redirect destination captures this query string parameter and saves this access code along with user’s profile. This access code is used in future to retrieve user data without needing them to re-login. User’s Facebook credentials are never saved in our application.

Once user authentication is complete, next step is data retrieval. A request to get user posts is made using the API and in response we get user’s profile and posts. User posts are returned in a multi-dimensional array that contains each posts type, data, likes and comments.

Upon receipt of this data we loop through all entries and render them using html5.

|  |  |  |
| --- | --- | --- |
| Facebook API Name | Version | Details |
| Facebook C# SDK | 6.0.10.0 | Platform C# |

### SoundCloud

First step for SoundCloud integration is to register our application with SoundCloud. Upon registration with Facebook, an application ID and a secret keys are provided. This is essentially our application’s user/pass pair to talk to SoundCloud API.

Once the user initiates the integration process, a JavaScript is invoked. This JavaScript initiates a connection to SoundCloud using the client ID which we retrieved during signup process and requests user’s profile data. A redirect URI is provided with the request. Upon successful login by the user, API redirects the user to the redirect URI we provided in our request. Embedded in the redirect URI is a query string parameter that contains the “access code” for that particular user.

Similar to Facebook this query string parameter is captured and saved in database for future data retrievals without asking for user to login.

Retrieving the data and rendering to HTML5 is much easier compare to other social media APIs. SoundCloud provides a widget that we can embed in our HTML page and that widget only needs the client’s access code we retrieved.

|  |  |  |
| --- | --- | --- |
| SoundCloud API Name | Version | Details |
| http://connect.soundcloud.com/sdk.js | NA | JavaScript |

### twitter

First step for Twitter integration is to register our application with Twitter. Upon registration with Twitter, an application ID and a secret key is provided. This is essentially our application’s user/pass pair to talk to Twitter API.

Once the user initiates the integration process, an http request is invoked to initiate twitter login prompt. A redirect URI is provided with the request. Upon successful login by the user, API redirects the user to the redirect URI we provided in our request. Embedded in the redirect URI are four query string parameters i.e. Token, Secret Key, Screen Name and User ID. Our redirect page captures and saves all these parameters in database for future data retrievals without asking for user to login.

Next we retrieve twitter data using the four parameters we saved, tweets are returned in a multi-dimensional array and contain tweets along with other information such as profile image, Author screen name, date and time, tweet text etc.

We loop through the array and render data to HTML5 page.

|  |  |  |
| --- | --- | --- |
| Twitter API Name | Version | Details |
| Custom C# OAuth Class | NA | C# |
| TweetSharp | 2.0.0.0 | C# |

### instagram

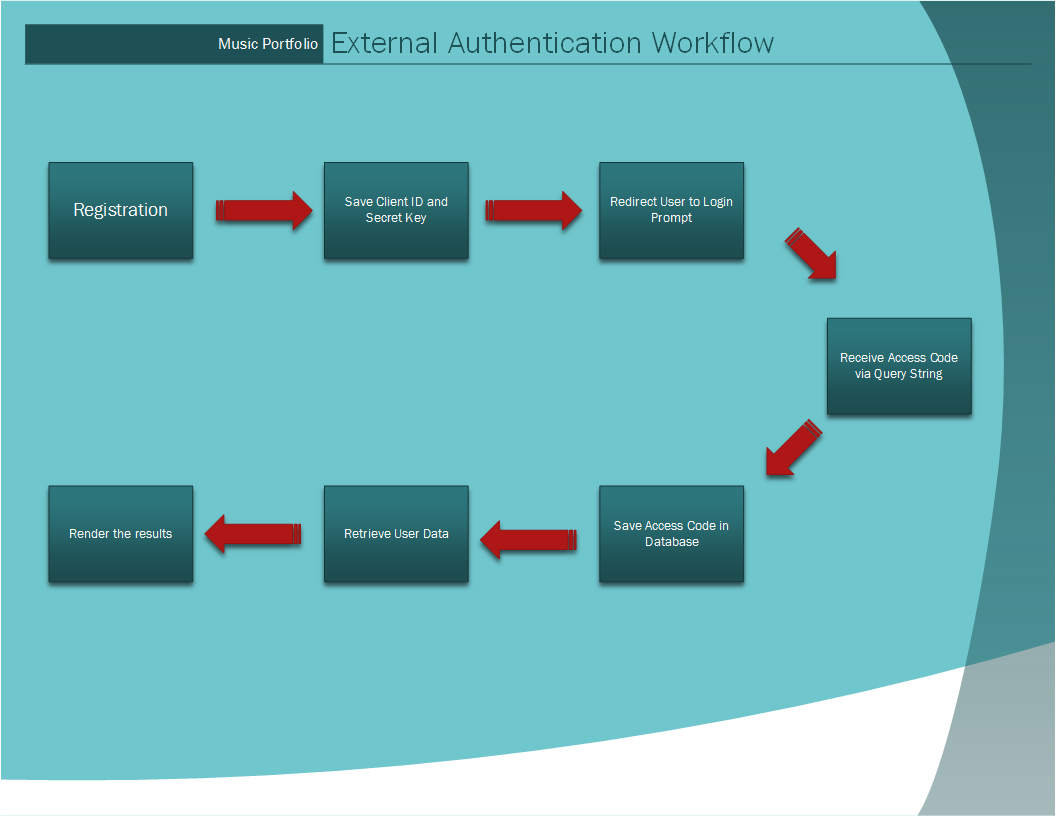
First step for Instagram integration is to register our application with Instagram. Upon registration with Instagram, an application ID and a secret key is provided. This is essentially our application’s user/pass pair to talk to Instagram.

Once the user initiates the integration process, an http request is invoked to initiate Instagram login prompt. We simply redirect the user to Instagram web server and we submit few parameters in query string, such as client id and redirect URI (to come back to your site). Upon successful login by the user, their API redirects the user back to our site using the redirect URI parameter. Embedded in the redirect URI is a **‘temporary code’** for this user. Then we make another call to Instagram API using this temporary code and this call brings back an **“access token”**. This is the permanent code for this user and our application saves it in the database for future data retrievals without asking for user to login.

Next we retrieve Instagram data using the access token we saved. We loop through the results and render data to HTML5 page.

|  |  |  |
| --- | --- | --- |
| Instagram API Name | Version | Details |
| Skybrud.Social | 0.9.3.0 | C# |

### external auth workflow



### LOCAL media

User’s local Music Portfolio profile and media uploads use HTML5, native ASP.net controls and C#. No 3rd party APIs are used.

## WEB API technical details

Our objective was to display all of musician’s data on mobile devices. Since none of the data from social media sites is stored in our databases, we make new API calls to these sites upon every visit. While this approach brings the latest updates from these sites, it presents a challenge for mobile apps.

None of these sites allow using same application ID and secret for websites and mobile apps and the APIs for mobile apps are different in many cases. Which means that if we wanted to talk to these social media sites, we would have to get new IDs and re-authenticate users for mobile platform.

To solve this problem, we decided to create our own server side API for our mobile application. This API receives requests from our mobile app, pulls the same data that a web user otherwise would request and sends it back to mobile app in JSON format. Creating this API greatly simplified the mobile implementation, user authentication and data retrieval were taken out of the picture. It only had to be concerned with the presentation layer.

We used .net ASMX web API and following functions were published. All functions are available for access via SOAP 1.1, SOAP 1.2 and HTTP GET. All code for this API is written in C#.

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Parameters | Response | Purpose |
| Filter | Name, Genre | Matching Musician’s profiles (name, agent, biography, contact info, profile pic) in JSON | Allow android application to filter results without having the need to keep local cache. |
| getBio | UserID | One Musician’s complete Biography in JSON | Once mobile user requests a musician’s details, this function is called. |
| getBioAll | None | Returns Biography of top 20 musicians in JSON | This is used when mobile app loads. |
| getFBPosts | UserID | Returns Facebook posts data in JSON | When mobile user clicks on Facebook tab, this call is made. |
| getFBProfile | UserID | Returns Facebook profile data in JSON | When mobile user clicks on Facebook tab, this call is made. |
| getInstagram | UserID | Returns Instagram data in JSON | When mobile user clicks on Instagram tab, this call is made. |
| getLocalMedia | UserID | Returns locally saved audio/video data in JSON | When mobile user clicks on Local tab, this call is made. |
| getSoundCloud | UserID | Returns SoundCloud ID | When mobile user clicks on SoundCloud tab, this call is made. |
| getTwitter | UserID | Returns Twitter data in JSON | When mobile user clicks on Twitter tab, this call is made. |

## DATABASE technical details

We used Microsoft SQL Server 2012 as our database platform. Since we do not save any social media content, database storage is minimal and limited to some metadata, user’s profile credentials and access codes.

Tables

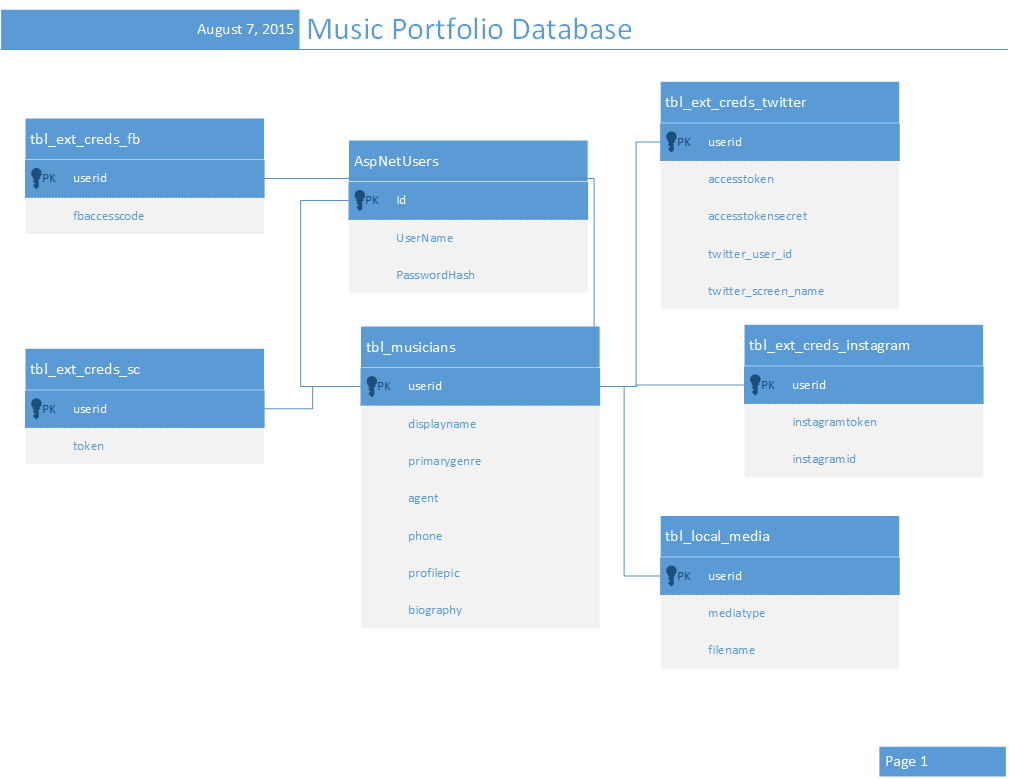
Custom tables include tables for site login, one table for each of social media sites, one table for local media pointers and one table for musician profiles.

Views

In addition to tables, we created a view to return most viewed musicians. This view joins the musicians table with a view count table.

Stored Procedures

Instead of embedding SQL into our C# code, we wrote stored procedures on MSSQL server for all inserts and updates.



## ANDROID technical details

Minimum SDK required to run the Android app is SDK 14 which is Android 4.0 IceCream Sandwich. Contrary to the website, the Android app is view only as it is complicated to setup Authentication for the Android app and we did not have sufficient time to do so. As mentioned earlier, the app makes web API calls to get JSON data. Each web API call is done on a separate thread to take load off the main thread. Since kernel allocates only a certain amount of memory to the app, each bitmap image downloaded gets recycled after the corresponding fragments gets destroyed.

The app has four fragments, **Most Viewed Artists**, **Search by Name**, **Search by Genre** and **Artist Profile**.

**Most Viewed Artists** fragment, just like the home page in the website, displays the artist in order of the page views that they have had in both Android app and the website. It makes getBioAll web API call to get data. For each artist, this fragment displays name, genre, profile picture and 150 characters of biography.

**Search by Name** fragment lets user search for an artist by name using the searchview. One of the features of the searchview is that it lists recently submitted queries so that the user does not have to repeat their search and simply click on one of the previous queries. For each search query submission, it stores the submission query in native SQLite database and when the searchview is expanded, it lists those queries. The user also has the option to clear search history by clicking the “Clear History” button. It makes Filter web API call to get JSON data and has similar feature to Most Viewed Artists when it comes to displaying the list of artists.

**Search by Genre** fragment is very similar to Search by Name fragment except instead of a searchview user can filter artists by selecting a genre from the dropdown spinner. Once the user selects a genre, Filter web API call is made and displays all artists of the genre selected.

On each of the three earlier fragments discussed, user can view the artist’s profile page by clicking on the profile image which in turn starts **Artist Profile** fragment. This fragment displays user’s full biography, social media profiles and features a custom media player for the local music database. Each tabs are loaded asynchronously on separate threads to improve performance as they make calls to their corresponding web APIs. All social media profiles are displayed in webview since webview requires less memory to load and display data. On destroy, the fragment destroys the media player and all webviews so that media playbacks do not overlap.

## appendix i

### Weekly Agenda and Team Notes

06/02 – 06/09

1. Initial DB schema is complete, we have defined our initial entities and their attributes. Separate document with details is attached.
2. We have created a basic site, it can viewed at <http://mp.makhdumi.net> . This site is using bootstrap library and will conform to RWD (Responsive Web Design).
3. Since there are multiple developers working on this site, all of the features that we have coded are not available in this demo site. We are simultaneously working on Facebook API, Instagram API, a/v uploads, musician signups and login pages.
4. Out of these Instagram API is complete, we have correct code to sign in a user and retrieve an authentication token from Instagram.
5. At the moment, demo page is hard coded to pull one user’s profile, once we finish our signup and login pages, we will use signed in user’s ID instead of hardcoded one.

06/09 – 06/16

1. Finished login and signup pages and integrate all the code from all team members into one project so all changes are visible in our demo site.
2. Made Instagram API work with logged in user’s credentials.
3. Finished Facebook API login and implemented the layout for Facebook widget (still under construction).
4. Added Youtube widget.

06/19 – 06/23

1. Finished implementation of Twitter API and widget.
2. Finished Facebook widget.

06/23 – 06/30

1. Finished SoundCloud Authentication.
2. Started Android application design.
3. Created SoundCloud webview within Android.
4. Created a web service online that returns user info data for Android app.

06/30 – 07/07

1. Finished local audio/video uploads.
2. Added functions to our API for android app. Add Instagram, Facebook, twitter and SoundCloud data.
3. Added features to android app to process and present all data.
4. Formated JSON data in Android app for all social media.

07/09 – 07/14

**Priority 1**

1. Fix the bug which disabled SoundCloud and Instagram in website.
2. Incorporate local music player in the mobile app.
3. Make web service for home page.
4. Setup the homepage for website and Android app.

**Priority 2**

1. Modify CSS to make website neater.
2. Incorporate navigation bar for Android.

07/21 – 07/28

1. Add search functionality to Android App.
2. Finish modifying CSS for the website.
3. Start adding more test users.
4. Start QA Testing (we expect this to go for one more week).

07/29 – 08/04

1. Fix bugs in website and mobile app.
2. Add more test users.

08/05 – 08/11

1. Add more test users.
2. Complete project documentation and presentation.