SoundIT

Your Music, Anywhere

APSC 486/COMM 466

April 5, 2013

Nick Adams **Sonal Haria**

Samuel Chan **Eric Seto**

Anuj Mehta **Douglas Cheung**

TABLE OF CONTENTS

LIST OF FIGURES

INTRODUCTION

SoundIT is a new venture formed by six UBC students through the new venture design course at UBC. The goal of the project was to build not just a technical project, but a business with a sustainable business model. During the course of this term, the three engineers design the architecture as well as the many user interfaces for the product, and made significant progress implementing the product.

From a technical perspective, the project can be broken down into three major components- the Android and iOS applications that enable users to modify the playlist, the web based venue application that lets venues stream music, and the backend that provides the supporting APIs and infrastructure to all the applications. During this term, Nick built the Android app, Sam built the iOS app and Anuj built the venue application. All three of us worked on the backend database and the APIs throughout the term.

This report discusses SoundIT's technical architecture, the tools and processes used by the engineering team, the progress that has been made, and features yet to be implemented. The report will start by briefly explaining the architecture of the product and the tools and processes that were used by the team. It will then describe the progress made in implementing the product as well as the challenges it faced along the way. This will then lead into a short discussion about the features that have yet to be implemented. The report will conclude with a discussion about the lessons learnt during this project, and some of my key takeaways.

DESIGN

Backend

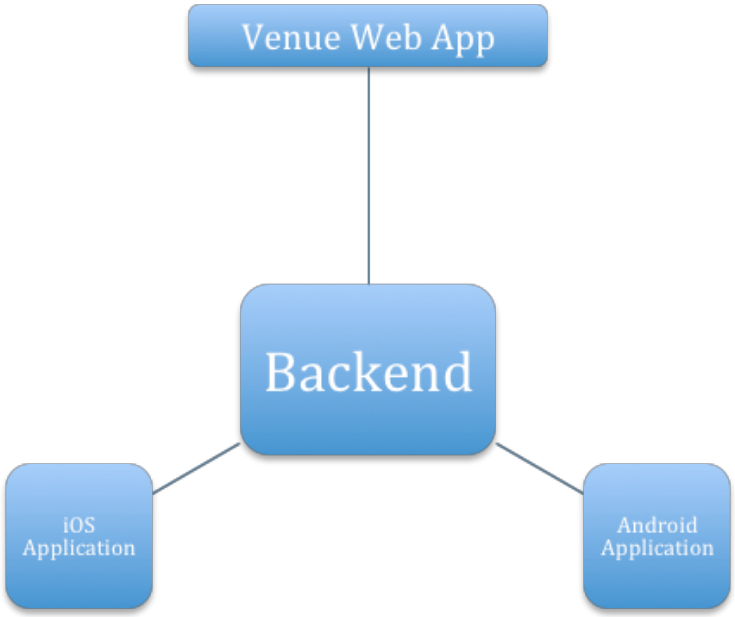

Figure 1: Overall System Architecture

As seen from the figure above, SoundIT consists of four major components- the Android and iOS mobile applications that consumers use to vote on a venue's playlist, the web based venue application that is used to create the playlist and stream the music, and the backend APIs and database to support the applications.

A major architectural concern we had was the high dependence of our system on a large number of mobile users in a small and condensed area, all of which were to communicate with the same backend. Scalability might become a concern depending on the number of users at the location. In ordinary cases where users are geographically dispersed, scalability can achieved by geographic database sharding and server redundancy, however this may not work for SoundIT because of the high concentration of users in any given location.

Fortunately, we did not achieve a large enough user base so that this would prove to be a problem. In beta tests, our users were limited to 60-100 per venue, and our servers were more than capable of handling this. While server scalability was always a part of the discussion while designing our data model and our APIs, we believed that it was best to avoid spending too much time optimizing for scalability when we did not have a significant user base. Instead, we focused on optimizing the experience and interface the product offered to its users.

We hosted our backend on Amazon Web Services Elastic Cloud Computing virtual machines. Amazon's EC2 offered the most economical, scalable and robust Cloud hosting service. While we did consider alternatives such as Windows Azure and Google App Engine our prior familiarity with AWS EC2 led us to go with the technology we were most comfortable with. We also considered hosting it on our own computers, but to better facilitate team coordination we had to have a shared environment that was always accessible. To save costs, we chose to use the free tier of Amazon EC2. We knew that this might result in scalability issues, but since it was mostly used as a development and staging box, it ended up working fine.

The communication between the backend server and the mobile applications was done via web APIs. Each mobile client would make an API request whenever required. Anuj designed the initial versions of the APIs with the consultation of Sam and Nick as per the needs of the mobile applications. As we iterated, Nick and Sam took on more backend and API responsibilities to ensure knowledge sharing as well as shared understanding of the entire stack. Each API was designed for a specific function to ensure that the APIs were reusable and easily maintainable. The APIs were designed such that both the Android as well as the iOS applications could use the same APIs. This prevented duplication of logic and enabled the reuse of code.

Android

The Android app was built in Java using the Android SDK developed by Google for Android developers. We chose to use Eclipse as our development environment because we were also using it for the backend and because it is the most popular and fully featured development environment.

The main goals with the Android application were to create a smooth, consistent user experience for users that would allow them to use the SoundIT service quickly without becoming frustrated and then move back into their previous activity at the bar. The application is not designed to be in your face all the time. We want to continue to promote a social atmosphere at our venues and we believe a simple easy to understand interface will help us achieve that. For these reasons the interface we have build for the Android app is very minimalistic and easy to use.

Nick was our main Android developer because of his experience with Android development in the past. Much like iOS there are very little viable options for developing apps so almost everyone uses similar tools. We did make use of an open source project called ActionBarSherlock [1]. ActionBarSherlock (ABS) is a project that backports the Android action bar feature from Android 4.0+ to earlier devices. We are using this because it allows us to support devices running older versions of Android while still creating a modern UI for users.

No data is persistently stored on the Android client, all the data is retrieved from the backend server using REST API calls. The REST library we wrote uses a separate thread for the request so it does not block the main thread. Once the data is retrieved, the returned JSON is parsed and stored in a client data model. A call is then made on the main thread to update the UI with the new data. This allows us to dynamically update the data without ever blocking the main thread so the user can operate functions of our app or close our app and move on to another app.

The album art images that are displayed are dynamically loaded using a thread pool model and cached in a least recently used cache (LRU cache). The thread pool allows us to download the images from the web without creating too many threads and web requests and crashing the phone,

even if there are hundreds of images to download because they will be queued. The LRU cache will cache the images in memory and if the cache becomes full, it will remove them from the cache in the order of least recently used. We also implemented a disk cache in case the memory cache becomes full. The disk cache is far slower than the memory cache for loading images, so all the image loading and saving will also need to be done in a background thread.

At this point, we have all of our data downloaded, but have to be extra careful in how we have done everything because it is very possible that if we create too many threads the mobile phone will slow down or the application will crash. Using thread pools in our REST library and our image caching class we are able to limit the amount of simultaneous threads and network requests that could cause adverse performance issues for the device.

The UI is composed of an activity class which contains the UI layout for the screen and controls the main functions of the application. Our activities contain a model called a Fragment, which represents a behavior or portion of a user interface in an activity. The reason for using fragments is that they allow for multiple fragments in an activity, and the layout of the fragment can change depending on the device size and orientation. Fragments are also a more reusable structure than an activity which is somewhat limited.

The main layouts in the Android application is a ListView. A ListView is a structure that displays a list of elements on the screen, each element has its own unique layout and and items. The views in the layout are reused as the user scrolls the device and the data is replaced as necessary. Because replacing text and images from memory is very fast, and loading images or text from disk or the web is done in a background thread we are able to maintain a consistent 30 frames per second required for a smooth user interface.

iOS

The iPhone app was created with xCode, the native development environment shipped by Apple for developing iOS and Mac software apps. We chose xCode for several reasons (many similar to Android ratifications).

One, Sam's familiarity with the development environment, the Apple SDK having developed two apps (one published) to the Apple App Store. Second, Objective-C, the programming language in which xCode primarily supports iOS development for, is extremely fast and efficient. Three, xCode is integrated in every sense of the word. Finally, much like Android being mainly developed on Java with Eclipse+Google SDK, almost all iOS applications are developed using xCode.

The iOS app was developed using the Model-View-Controller (MVC) programming pattern. The app was split into three major screens. One, a SplashScreen view and controller class that initiates connection to the web server and does sanity checks (Internet connection, web server response etc). Once a valid connection is setup the app automatically goes to the second screen, a CurrentPlaylist screen that allows the user to vote up their favorite song as well as browse the current playlist. This screen also has a view and controller class file. Finally, a button on the CurrentPlaylist screen takes the user to an AddSong screen which lets user browse a library of songs and add their favorite songs to the current playlist. The AddSong screen is also implemented in MVC with a controller and view class.

The web server handles the organization of the data into a model abstraction. I simply interact with it through the REST Network API that I built.

The diagram on the next page details the all my major implementation and class files.

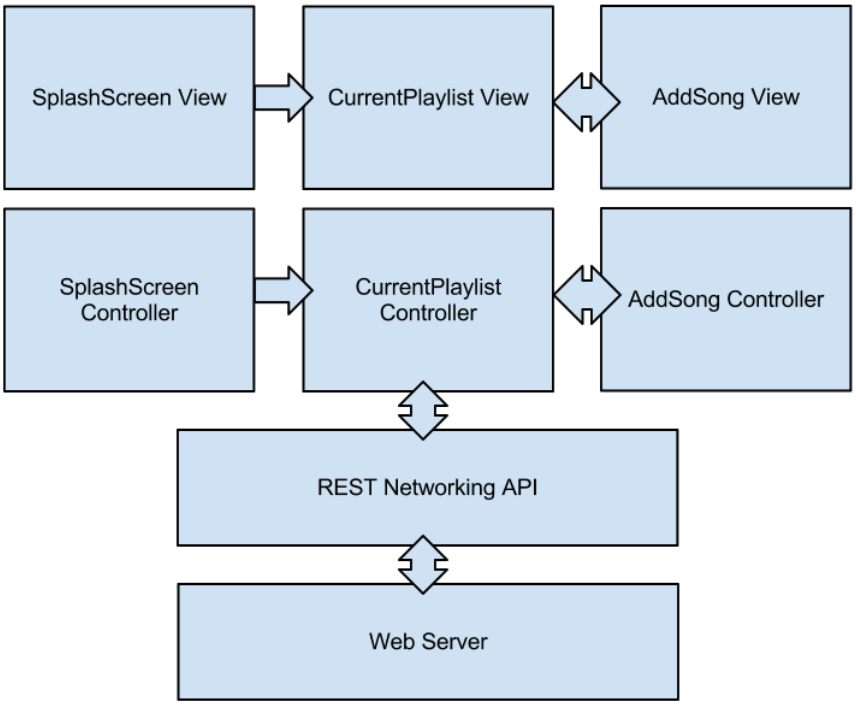

Figure 2: iOS class diagram

Venue Application

The venue application was created to allow venue owners to be able to play music from the SoundIT system. We had the choice of creating native applications for Windows, Mac, iOS and Android, or creating a web application that could be used on any of these platforms. Creating native applications had the inherent advantage of an easier to use interface as well as a better developer access to client resources. However, we decided to start with a web based application since it would be quicker to prototype with and was a solution that would work at more venues since both mobile devices and personal computers should have web browsers available. Anuj's familiarity with web development also contributed to this decision, as he was the primary venue application developer.

To play music from the web application, we used HTML5's stock audio player. We used an innovative combination of front-end JavaScript and backend Python APIs to loop through the songs automatically. This was done to ensure that venue owners and administrators do not have to monitor the music system.

We took several steps to optimize both the user experience provided to venue owners, as well as the performance of the front-end. The venue application was designed to be a single page application since the initial functionality was very limited. The user interface was designed to be aesthetically simple, with clear indication of what the user is able to do at any given time. A screenshot has been pasted below to show readers what the venue application looked like.

Knowing that the responsive of the website is affected to a significant extent by the number of HTTP requests per page load, we minimized the number of icons on the page. We enabled server side caching as well. The caching was hard coded to two weeks from the date of first download. While we knew that there are better techniques to cache such as basing it on name and date changed, we decided to focus more on functionality than optimization.

We put our JavaScript and HTML within external files as much as possible to ensure maintainability, but we ensured that these were limited to one file each. This was one of the steps we took to reduce the number of HTTP requests. We also put the reference to our CSS at the top

of the HTML file. This ensures that each page is already formatted correctly during initial rendering by the browser, and prevents any resulting white flashes post-rendering. Since some browsers make their JavaScript requests serially, we put our JavaScript references at the bottom of the HTML file. This ensured that the page is loaded and rendered quickly, without having to wait for the supporting JavaScript files to render.

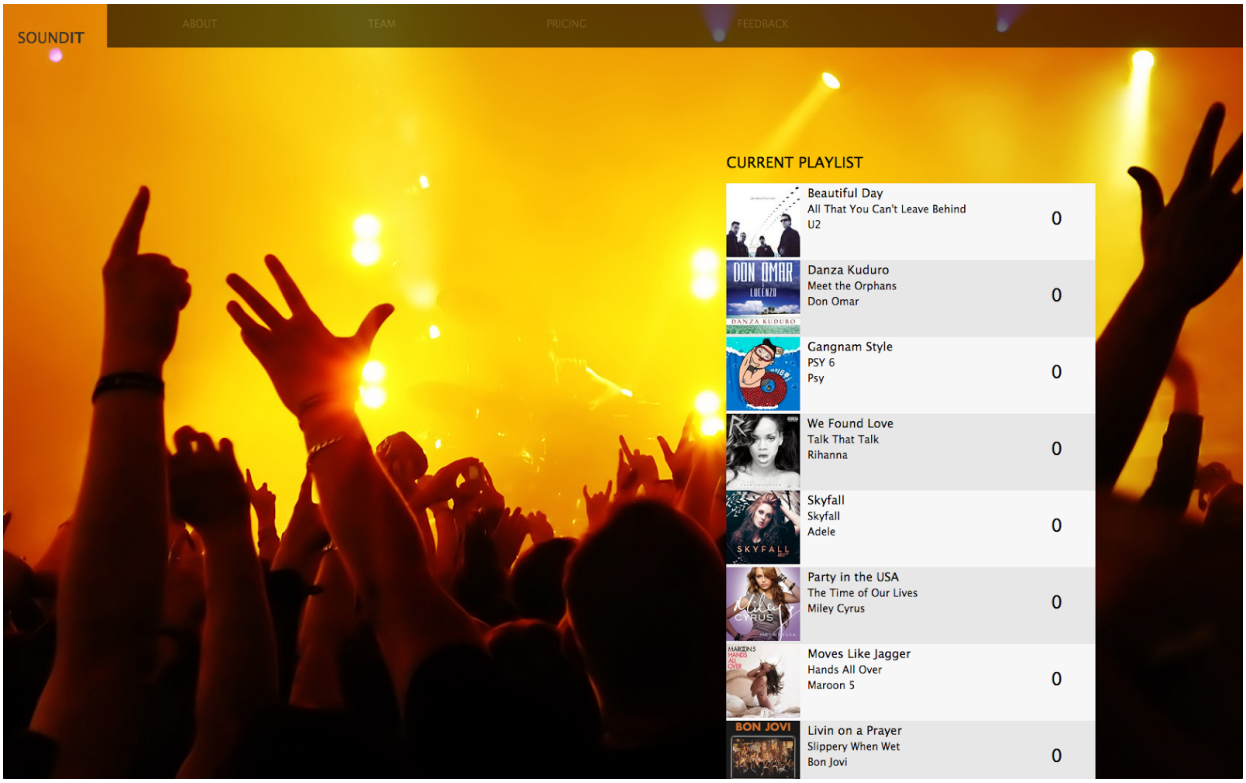

Figure 3: Screenshot of venue application

TESTING

Beta Test at the Pit Pub, UBC

Our first big milestone was the unveiling and deployment if the SoundIT system at the Pit Pub UBC in a private beta test for invite-only guests. The goal of the test was to stress test our web server, gather valuable feedback regarding our mobile apps, and test the system as an integrated whole for obvious flaws, drawbacks to improve upon.

Figure 4: SoundIT Beta Test

The test was very successful. We got over 70+ unique users and over 500+ API calls over the short three-hour event. Many of the testers gave us positive feedback as well as valuable critique and problems with the current system, which we took to heart moving forward.

Enterprize Business Plan Competition

Our second big milestone was SoundIT qualifying for the final round of the Enterprize Business Plan Competition. The event is a nation-wide business plan competition where start-ups pitch and compete with their various products. The final round held over ten thousand dollars worth of cash prize and many industry experts and venture capitalists were on hand to critique, give feedback and generally provide us with insight on the venture. We came in 3rd place and netted a one thousand dollar cash prize. More importantly, we got valuable feedback from industry experts and made some potential mentors in the process.

Figure 5: 3rd place at Enterprize Canada

Launch Academy: "The App Pitch"

At this point, our technology was well developed enough and business plan robust enough that we could enter startup pitching competitions in the app space with minimal additional preparation.

One such opportunity arose when we heard of the Launch Academy "App Pitch" event where Launch Academy, an incubator program situated in the Downtown core, and various sponsors (Microsoft being a notable one) was holding one such pitching event.

Anuj and Sam went down and pitched and showed off SoundIT to the 100+ entrepreneurs and nearly 30+ app ventures. We won first prize and received a five hundred dollar cash prize and two tickets to Polygot, a software developer conference held annually in Vancouver, BC.

Pacific Venture Capitalist Competition

Another competition SoundIT was featured at was the Pacific Venture Capitalist Competition (PVCC). The competition featured prominent venture capitalists (VCs) from Vancouver who listened to mock funding efforts from various student startup teams from across the nation.

SoundIT once again made the final round out of nearly 30+ applications nationwide and we pitched our venture to several prominent VCs from professional VC firms and companies. While we did not win a prize at this competition we came away with valuable feedback once again, specifically with the business aspects (such as the our monetization strategy) of the SoundIT venture.

UBC IEEE Project Fair 2013

We also participated in UBC's end of year IEEE project fair. This fair was to showcase technical achievements by student teams in electrical and computer engineering at UBC. While we did not win any monetary prizes at this competition, we got valuable feedback and suggestions from some ECE faculty.

CONCLUSIONS

This report detailed the process, design and technical developments associated with SoundIT; a digital jukebox for mobile and web music service that empowers bars, pubs and lounges to let their patrons control the music. The idea is born from the UBC New Venture Design course, where three business students and three commerce students work over a semester to attempt to bring a company to life.

SoundIT attempts to provide a solution to the problem of bad music at clubs, bars and lounges. On one hand owners find it difficult to find and play music that their given customer demographic, at any given time, wants. On the other customers dislike having to listen to bad music and wants some way of interacting and influencing the music being played at their favorite venues.

Our team presented and tested SoundIT at a variety of test events, business competitions and pitching events. The first was our public beta test we held at the Pit Pub at UBC where we gathered 70+ unique downloads and over 500+ API calls. We competed at business plan competitions such as Enterprize and PVCC, pitching events such as Launch Academy's App Pitch 2013, and at a technical project fair- IEEE Project Fair. We were able to successfully demo our product at each of these events without a single glitch, resulting in very successful technical operations.

In conclusion, the SoundIT venture was a technical success. The technology was thought out, alternatives considered and finally decided on. Our development process was smooth and our final product was feature complete. The engineers have all walked away with more experience in software development for our respective platforms and, more importantly, gained key insight into how the business side of a tech company is organized, deals made, and profits gained.

FUTURE STEPS

The SoundIT system is currently fully functional, but we want to continue to add new features such as location fencing and more analytics to gather data about how our users use our service. We also want to continue testing our service at venues to verify that it is a stable product that could be rolled out to multiple places. We also will focus on polishing our apps so they work even faster and look nicer for consumers to use.

The next step for the SoundIT team is to continue working with local venues to create interest and allow us to test the SoundIT system as it evolves. Because we were having trouble generating enough interest from bars, pubs and lounges without going to each one directly, and we believed that this would continue until we were an established brand, we decided to investigate radio.

The radio industry generates huge amounts of revenue each year, but is beginning to decline due to the rise of easily available Internet radio services such as Pandora or Spotify. These Internet services provide the same service, generate revenue from advertising, just like radio, but also get tons of analytics and user data about their users. Most radio services use primitive estimation techniques to perform analytics. Because radio is beginning to decline, we believe that they will be looking for technologies to help them grow and expand amid the new era of Internet radio. Modifications to our service will be required, such as branding the application for a specific radio station and adding radio specific features before we attempt to launch on radio. We are in talks with radio stations to determine what their technical requirements would be for such a service and then we can begin to modify our system to simultaneously support both radio and our traditional business model.

REFERENCES

[1] Wharton, J. ActionBarSherlock, accessed April 4, 2013. http://actionbarsherlock.com/