CS194 Project Proposal

James Capps <jcapps@stanford.edu>
Alex De Baets <debaetsa@stanford.edu>
Max Radermacher <maxrader@stanford.edu>
Matt Volk <mvolk@stanford.edu>

Project Description

We will build an iOS app that automates playing music for a group. The main purpose of the app is to allow a host to broadcast a library of songs that listeners can request from using their own devices. The app then queues songs based on the number of requests and creates a mix of songs for the party in real time.

User Interface

The app has two different views, one for the host and one for the listeners. The host can browse through all songs in their personal iTunes library and check off which ones the listeners can choose from. In a separate view, they will see the queue—a list of all upcoming songs and the number of requests for each. The app will automatically keep the queue full, but the host can modify it as much as they'd like.

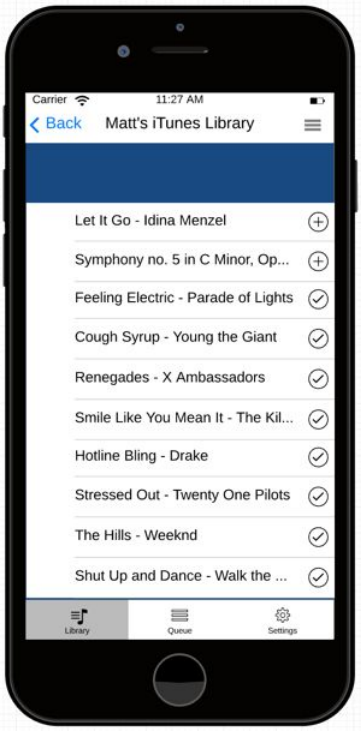

Figure 1(a): Library

Figure 1(b): Queue

Listeners will see just the songs that the host has made available for the party, as well as the queue. They can upvote and downvote individual songs, but cannot directly change the queue. All listeners will see a live version of the queue, which changes in real time. Because the queue uses the host's iTunes library and nothing else, no login is required.

Major UI Elements

- **Library**
  On the host's device, the library is sourced entirely from their own iTunes library. It lists all of their songs, playlists, and album information. All listeners see the subset of the host's library that they have made available for the party. A view of what this page might look like for the host is in Figure 1(a).

- **Queue**
  The queue is the main hub of the app and is a list of which songs will be played next. This list is generated automatically based on how many votes individual songs have. The algorithm will also take into account factors such as how recently songs have been played. The host has full control to change it as they desire. A view of what this page might look like for the listeners is in Figure 1(b).

Stretch Goal Features

To the right is a list of features we would like to implement as stretch goals, listed in order of implementation.

Stretch Goal Features
Bluetooth-based Communication
iPad / AirPlay View
Spotify / SoundCloud Integration
Multi-Device Playback

- **Bluetooth-based Communication**
  In its current form, the app relies on all listeners being on the same WiFi network. Bluetooth-based communication could remove this constraint.

- **iPad / AirPlay View**
  To the best of our ability, we will design our UI such that it will interact with an iPad, but we would like to develop a separate design that takes advantage of larger screens.

- **Spotify / SoundCloud Integration**
  Because using third-party music apps can be difficult, we will start using the built-in iTunes library. But, if time permits, we'd like to tackle allowing multiple music services to contribute to the host's library.

- **Multi-Device Playback**
  For users who are not in the same physical location but would like to make a playlist together, multi-device playback will allow users to sync what they are listening to over the internet.

Need for Product

Our product completely automates an essential task for organizing a party: handling the music. The host, once constantly interrupted with song requests, is now free to sit back and enjoy their own music. Our app also democratizes the process of picking music, allowing all listeners to have a say in what's playing. The playlist is free to adapt to the mood as people's musical interests change. Moreover, no hardware is needed, and the host doesn't have to entrust their own precious device to any strangers, because everyone can vote using their own phone.

We envision our app becoming a staple at parties, in the car, with radio stations, at fancy dinners, or even in your dorm room.

Potential Audience

The audience for our application is anyone who is trying to create a collaborative playlist where songs are queued in real time based on the listener's preferences. This application is perfect for those trying to host gatherings, from small groups to large parties. This application does not require a large audience. For example, even if only several individuals in a larger group have the application, a playlist can still be generated based on those listener's preferences. Finally, no technical knowledge is needed—anyone who can use Apple Music can use our application!

Discussion of Competing Products

Our application is roughly based on an old iTunes feature called iTunes DJ. Unfortunately, iTunes DJ was discontinued several years ago in iTunes 11. However, there are other options which we will discuss below.

- **Track.tl** is a similar collaborative music player. However, Track.tl requires the host to sign in via a social network account. In addition, Track.tl forces the listeners to have a unique URL to the "party" in order to join instead of using location or a wireless network to join. Furthermore, Track.tl pulls its music from Deezer, YouTube and SoundCloud.
- Both **CrowdSound** and **QCast** are very similar to our application. Both Applications require login via social media or email account. However, CrowdSound and QCast both populate their music libraries using Spotify and other web streaming services. Neither application allows the host to use his or her music library.

Major Technologies Used

At a very high level, we are planning to implement this project within the realm of an iOS application (theoretically supporting iPhones, iPods, and iPads with the same binary). As such,

many of the choices for technologies will be guided by the requirements imposed by Apple. When we do have a choice of a few technologies, we tend towards those recommended or endorsed by Apple since they have the best support within the platform. We have tried to stay within a single platform (i.e., no web service) to try and reduce the complexity of the project.

As for an IDE, much of the development will take place within Xcode on account of its close integration with the platform. In addition, the supplemental tools provided as part of the package (Interface Builder, iOS Simulator, Instruments, etc.) will be used to aid in the development of the application.

Swift

There are two officially-supported languages when developing a native application for iOS devices: Swift and Objective-C. While it is possible to write an application in another tool and convert it to native code, this approach is not recommended. Due to the generally lower quality that these tools produce, and due to the fact that members of the team wish to learn iOS programming, we have elected to use Swift, one of the languages supported by Apple.

While no members of the team have much experience with Swift, it is the only language needed by the project. (We don't have a web component, so we don't need to worry about server-side languages.) There are many resources available online for free to aid in the learning process for Swift. (Plus, it is very syntactically similar to current popular languages, so it won't look nearly as foreign as something like Objective-C.)

Music API

A major reason for choosing to build an iOS app is the predictable, closed nature of the platform. It makes it much more reasonable to develop an application in the allotted time because the environment is more controlled. Along with this, because the devices are predictable and consistent, Apple is able to provide easy-to-use APIs that are "guaranteed" to work. (This means that users can't inadvertently break them.)

One of these APIs is the iPod Music Library API. There are several aspects to this framework, but there are two that we will utilize most heavily in our application. The first is the ability to read the metadata for the songs contained in the library. We will use this information to populate our own user interface. Apple provides everything from the name of a song to the album artwork. The second feature is the ability to pass one of these song objects back to the framework in order to play it. The focus of our app will fall in between these two steps: how we choose from among the available songs which one should be played.

Bonjour / CFSocket / NSStream

A critical component of what we are building involves communicating among multiple devices. We need to be able to send the list of songs to client devices, and we need those devices to be

able to send song requests back to the main device. We'll do this using some pretty standard networking technologies (sockets, streams, TCP, etc.).

For iOS development, we'll probably use Bonjour for the discovery of devices on the local network, the CoreFoundation layer to create the socket, and the NSStream subclasses for the actual reading and writing of data. This will make it easy to discover devices, and it will let us use one of the higher-level classes for actual communication. Using a higher-level class like NSStream should be more than sufficient for our purposes.

Core Data

For the internal data model representation within the app, we will most likely use Core Data. While this framework is generally used for persistent storage of data, some of its tools will be helpful in developing and maintaining the data model. Plus, it provides the team with more experience in another important part of the iOS APIs.

Core Data provides a visual modeling tool to establish the relationships between various objects. It can then automatically generate and implement the corresponding class files. Not only does this avoid mistakes made when writing source code by hand, but it keeps the implementations much more consistent. As a final benefit, it makes it much easier to maintain the model layer as changes are made.

Git & Github

With respect to source control, we are going to use git. Everyone is familiar with git, and so we should be able to successfully use it without too much overhead. We have discussed some of the common usage patterns for git, and we have made some decisions about how we intend to make the best use of this tool. We have also created a public Github repository for the project.

Resource Requirements

Our project requires very few resources. Our entire team needs to have a computer which runs OS X. While this is problematic for James, who uses a PC, he has access to many cluster computers which all run OS X. In addition, we need to be able to test our application on all screen sizes for both the iPhone and iPad. Our group can provide almost all the screen sizes for the iPhone except for the 5.5 inch (iPhone 6 Plus) screen size. Furthermore, we'll need to be able to test our application on all the screen sizes for the iPad. We will rent these devices from Lathrop.

Potential Approaches

We discussed a few different ways to implement our idea of "let people request songs," and we settled on our current approach after considering some of the weaknesses or potential roadblocks

with the others. The main discussion points involved the source of the music and the destination(s) of the music.

With respect to the source of the music, we were divided about whether or not we should source it from the local music on the device or an app like Spotify. The team was divided about 50-50 on this issue, but we chose to use the local music given that (a) Apple provides an easy way to do so and (b) we could not find any officially-supported way to accomplish something similar with Spotify. It would have involved more "hackish" workarounds, and that was not something we wanted to pursue.

In addition, in testing some of the other apps that fall into this category, the need to log into an online service before playing songs was a rather intimidating barrier. By using the Music API, there is no need to log in to any account, and there is no need to register. That will make it very easy for people to get the app and quickly request songs.

For the destination of the audio, there was discussion about whether or not it should come out of only the host device, or whether it should be possible to simultaneously play the same audio from multiple devices. The use case in the latter would be people who are physically distant (and perhaps communicating via some video chatting service) who all want to listen to the same songs. Once again, we decided to not pursue this for complexity reasons (it's more difficult when working with a local library) and copyright reasons (it's probably illegal to "share" songs in this way, even if they aren't stored on other devices). Therefore, this has been left as a topic to reconsider, time permitting.

Another potential approach would be to design the host application to run on a computer instead of a phone. This would give us some more flexibility (and would address some of the issues with the closed nature of the iOS platform). In addition, the original inspiration for this project—iTunes DJ—was a part of the Mac incarnation of iTunes. There are two reasons why we chose to design this application as a mobile application rather than a desktop application. First, we are constantly transferring more and more usage and functionality from traditional desktops to mobile devices. It is much more common now to see someone plug a phone into speakers instead of a computer into speakers. Second, the team was more interested in learning iOS development than Mac development.

Assessment of Risks

While building this app, we will need to be mindful of a few possible risks and obstacles that we might face. We have hopefully anticipated the greatest risks and have listed them here.

- **WiFi**

  The use of this app as we have designed it assumes that both the host and the listeners are accessing the same WiFi network. This puts a significant restriction on how host and listeners use the app.

- **Compatibility**
  We are targeting our app to be compatible with as many iPhone and iPad devices as is feasible. Accounting for screen sizes and different iOS releases is an obstacle we will need to overcome.

- **Copyright**
  We need to be particularly careful about how listeners should be able to access songs. The app requires song information to be shared between the host and the listeners, and a risk of copyright infraction could become an issue during this sharing process.

- **Design**
  Another foreseeable risk lies in the graphic design of the app. To achieve a pleasant experience for the user, the UI needs to be simple enough for easy and intuitive use, yet simultaneously it needs to be unique and non-generic for a strong appeal. In designing the app we will have to successfully balance these two requirements.

- **Trend**
  In building this app, we recognize that one of the greatest risks includes dissipating appeal to users over time. As technologies and forms of entertainment shift and change, our app would be in danger of losing its relevance and appeal among our audience. This is evidenced particularly by the fact that users are beginning to move away from iTunes as a platform for listening to music, preferring alternative applications like Spotify, etc.

Next Steps

The next stage of planning involves putting together a rough model of how we want the application to look and operate as well as dividing up the work among team members.

We see three main branches of this model: User Interface (including layout and graphic design of the app), Networking (handling WiFi requests and managing the queue of songs), Importing Data (retrieving song data from iTunes), and Storing Data (implementing how the retrieved data is represented).

We will as a group work together to lay the groundwork for the basic design and backend implementation of the app. Once the basic framework is completed each team member will work on separate individual tasks within the branches.