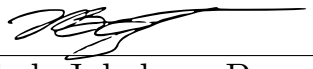
iOS programming, technical report

Live Music

May 21, 2013



Jakob Jakobsen Boysen
s112344

Contents

1 Introduction

This report will focus on the software engineering part of making an iOS app, namely the Live Music app. The choices made are documented and alternatives, sometimes better, are listed as well. The original description of the app from Podio, Appendix A, has been changed slightly to only describe the features actually implemented.

2 Description and features

This iOS app is targeted iOS >= 6 and iPhone only. The main purpose of the app is to easily list nearby concerts for today and the coming days. The initial screen of the app consists of two parts: The top part shows the current location and the bottom part is a list of events added to favorites. The next screen is the Events screen: Here events are listed based on the location. The events are grouped by day and ordered by time. At the bottom of each day-group a "Show on map..." entry can be pressed and a map with events as pins can be seen. At the top it is also possible to filter the results loaded so far.

Each event can be clicked and additional details about each event can be seen: The main artist information and picture, supporting artists, the venue, distance from current location to venue and a link to an automatic youtube.com search of the artist. On this screen there is also three buttons: Share, remind and favorite.

2.1 iOS APIs

The app makes use of the following iOS APIs:

CoreLocation and MapKit Used to get the current location and show concerts on a map.

EventKit and EventKitUI Used to create entries (with reminders) in the calendar.

MessageUI Used to the share functionality.

2.2 Web APIs

Besides the traditional iOS APIs the app also makes use of RESTful APIs from the following services:

Songkick.com Is a provider of live music events. All events are linked to a so-called metro-id, i.e. metropolitan identifier, and it is possible to search and list all of these events and metro-ids.

Last.fm Is *"a music discovery service that gives you personalised recommendations based on the music you listen to."*[1] But it is also a library of thousands of artists, albums, tracks, artwork and events, that can be searched and listed as well.

[1] `http://www.last.fm`

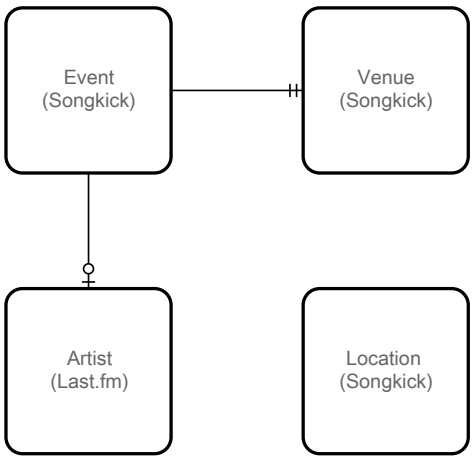


Figure 1: Data structure

3 Data structure

Figure 1 illustrates how the data is organised on the back-end and indicates which service is the source for that particular data. The concrete data classes are subclasses of the more general classes `Event`, `Venue`, `Artist` and `Location`. Each of these classes are initialised with a `NSDictionary` object describing the data entity in question.

It should be clear that this data structure allows for easy switching to or implementation of more data sources, e.g. to support use of Last.fm's event calendar as well.

3.1 Data fetchers

The data classes mentioned before are created from the data from the data sources Songkick and Last.fm. The `Fetcher` class has functionality for querying a RESTful API and parses the (only JSON) response to a `NSDictionary`. The concrete implementations, `SongkickFetcher` and `LastFMFetcher` of the `Fetcher` class, have numerous extra methods added.

4 Component overview and design

The different tab bar relationships, navigation bar relationships and segues between views can be seen in Figure 2. Not all views will be explained in detail, but we will focus on a few interesting views.

4.1 Changing location

The current location can be changed either by choosing another location nearby based on GPS coordinates or by entering a search query. The general location changer consists of a navigation bar with a cancel button, a table view listing the locations and a tab bar at the bottom. The entries in the table view are instances of the `SongkickLocation` as only Songkick is used as location provider for now.

As one might notice the two views look almost the same, but there has not been taken advantage of this fact in the views. Instead we have made one location view controller

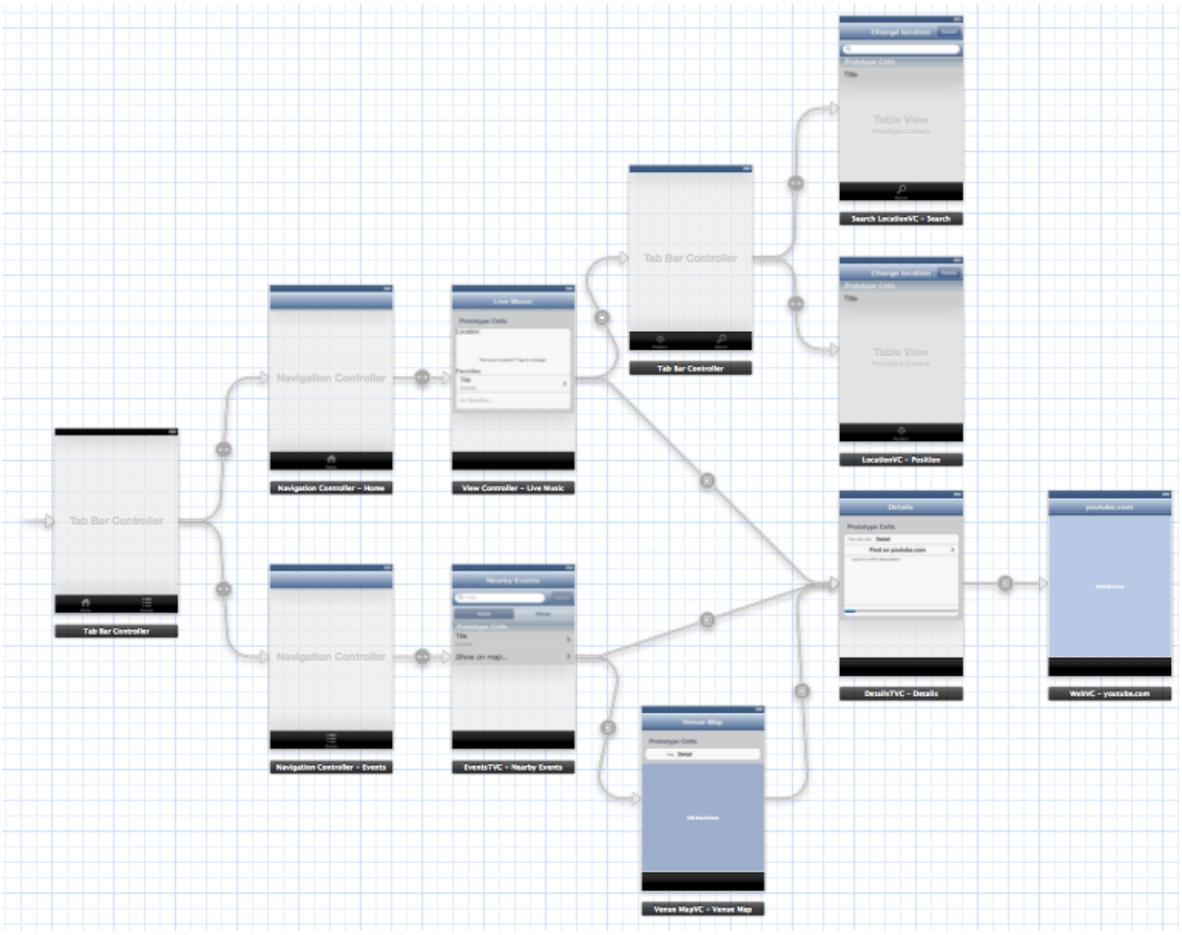


Figure 2: Storyboard

that handles the data in the table view and the cancel button, sub classes of this need just to set the property `NSArray *locations` and the data in the table view will update automatically. When the user presses an entry in one of the table views a message is sent to a view controller implementing the `DLMViewControllerDelegate` protocol. This makes sure the location button on the home screen is updated.

Another approach to creating two almost identical views could have been to create one shared view in a .xib-file, but this would have one major downside: In the storyboard we would not be able to easily see how views are related, as we would have to create the view controllers in the tab bar programmatically, thus loosing the huge benefit of using the storyboard.

4.2 Details Table View Controller

The event details view is the most feature rich view in the entire app. The view controller `DetailsTVC` conforms to 4 protocols and inherits from the `UITableViewController` class. The view itself consists of some dynamic prototype table cells with two custom table view cells, a table header and a table footer initialised from .xib-files.

The table header consists of a label, an image view and three buttons, and the table footer consists of a map view. In this view controller we decided to create these views in individual

files instead of creating the same views as prototype cells just like in the initial view controller of the app that has a prototype cell that contains the labels "Location" and "Favorites" and the current location button, see Figure 3.

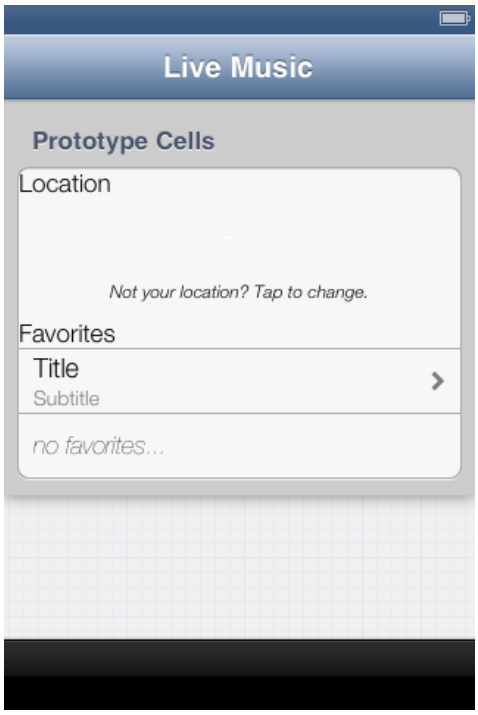


Figure 3: Initial screen

By creating prototype cells it is easier to understand how views are put together but we are not able to easily move specific view code out of the view controller. The `DetailHeader` class sub classes `UIView` and has IB actions for the buttons and IB outlets for the label and image view. The button actions are passed on to the delegate controller, and the label text and the image view content are set when an event has been loaded into the view from the view controller instantiating it. By doing this we can easily keep specific header view code in this class and instantiate it in the `DetailsTVC`.

The `DetailsTVC` conforms to 4 protocols to be able to handle mail sending (share), create entries in the calendar (remind), annotate and receive updates from the map and to be able to make the share, remind and favorite functionality accessible for others (in this case the `DetailsHeader`.

The details view needs to load some data from Last.fm before it is able to show the view. The data is loaded in a chain asynchronously on another thread than the main thread, and each time some data has been loaded the view is updated accordingly - but as long as the property `loadingPercent` < 1 the progress view is shown.

4.3 Graphics

The app icon have been created by finding an image on Google Images and using an online tool to convert the image into appropriate sizes. Apple's guidelines on graphics "Custom Icon

and Image Creation Guidelines"[2] states that the launch image should show all static content of the initial view controller, thus we simply just took a screen shot of the home screen and edited out the non-static elements. This obviously gives a feeling of a faster loading app.

4.4 Network dependency

Right now the app heavily relies on an internet connection. The app has been set as "Application uses Wi-Fi", but this means not that the app cannot work without internet connection, it only means that you get a warning when you enter the app without internet connection. For now the app is not very useful without internet connection, and there is even no error-handling, so a further improvement would be to handle no internet connection correctly and maybe also cache at least the favorite events.

5 Profiling

When the app came closer to a state where all functionality were working, the app was profiled using Instruments. It was quickly clear that the design of the event details view controller caused a memory leak, see Figure 4. By further investigation two circular references was found causing iOS not to be able to determine when to deallocate the memory for the view controller and the views. In this case it was pretty important as the map view in the footer allocates quite a lot memory.

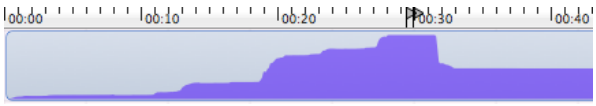


Figure 4: Memory leak: This is a trace of the app loading, tapping the Events tab and going in and out of the first four events. The flags at the time line indicates a memory warning.

A quick fix to this problem was to set the delegate of the header and footer to `nil` in `-(void)viewDidDissappear`, after that and with automatic reference counting in place the memory was now properly freed as seen on Figure 5.

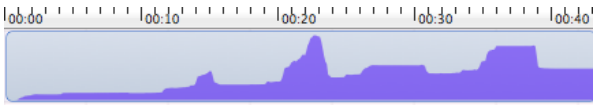


Figure 5: Memory leak fixed for the same trace as Figure 4.

Other than that we did not find any unusual memory allocations.

6 Testing

No test suites have been set up, but preferably there should have been created unit tests for the model classes and user interface testing with Instruments.

[2] `http://developer.apple.com/library/ios/#documentation/userexperience/conceptual/mobilehig/IconsImages/IconsImages.html`

The app has been tested using the simulator and a physical device (iPhone 4S), both running the latest version of iOS, which is 6.1.3 at the time of writing.

7 Conclusion

We have created an app combining standard iOS features with two different data sources. A few challenges regarding loading of data and managing memory were met but also solved to the best of our knowledge.

7.1 Further work

Before any further development is done on this app test suites should be set up first. After that the app could be extended with several features, e.g. searching for artists, getting events from Last.fm as well, showing more details (price, genre, etc.), show events based on Last.fm/Spotify profile, etc.

A Original description from Podio

The main purpose of this app will be to easily show the nearby concerts today and the coming days. Songkick.com is a database of live music events (and other kinds of events) which has an open API that easily can be used. The events should be shown in lists and maps, and it should be possible to get more details about each event by clicking it. The details include date and time of event, venue, specific details about the artist (preferably retrieved from last.fm), maybe ability to listen to tracks by the artist (again through last.fm or maybe Spotify), the distance from the current destination to the venue, ability to create reminders in the native Reminder app directly, ability to share the event through email, Facebook, Twitter, etc.

The API from last.fm also allows to retrieve the recommended artists for a user based on already listened music - Spotify might have this feature as well - use this in a functionality where recommended artists are looked up at Songkick.com.

Songkick.com A database of events.

last.fm A tracker of what artists you listens to and recommends similar artists.

Spotify A music streaming service.