Pollster iOS App
Chris Anderson
CSCI 4237

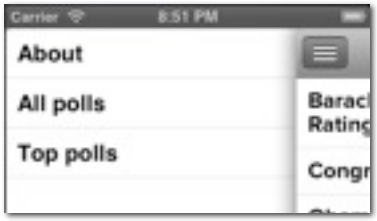
Introduction
Pollster.app is an iOS application that consumes the Pollster.com API to display political poll results and aggregate data about issues such as presidential favorability ratings and issue handling. The app has lists of the most popular polls and all polls; both lists are sorted and searchable. Individual polls also show all of the historical data associated with them.

Pollster.com provides their poll aggregates via a public REST-style API that provides JSON data in response to RESTful style queries. For example, one could query to get all polls around a certain topic, or polls in a certain location.

Background
Having worked in the political polling field in a prior job, I had always wanted to make a political polling app. Many such apps exist, but most of them are fairly slow, with cluttered UIs and a glut of unnecessary features. My goal was and is to make an app that loads and displays data fast. And, especially in politics around election season, people want polls provided as fast as possible.

I also wanted to explore aspects of the iOS SDK that were previously unknown to me, so I made an effort to use Apple-provided APIs whenever possible. Initially, I was going to use RubyMotion, a Ruby-backed iOS development framework, but that framework proved to be too immature for my tastes.

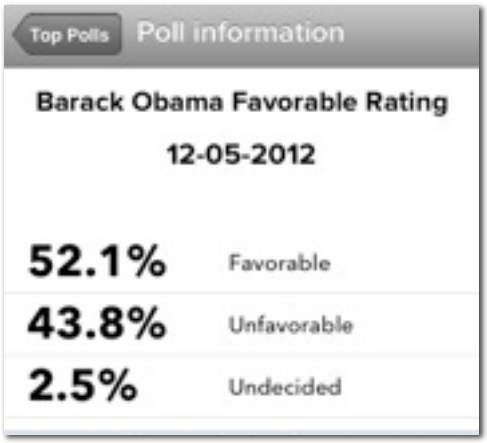
UI
The app is fairly straightforward, consisting of three main parts: the UI, network management, and data storage. I built against iOS 6, the latest version of the iOS SDK as of this writing, to make use of iPhone 5 size screens and native pull-to-refresh controls. While the app looks fairly simple, I spent a lot of time making sure the pieces that were there worked well. For the UI, I made efforts to customize the UI using Apple-provided methods instead of using custom images for buttons and rows. I provided a custom font, and used that font for all of my text rendering. I applied tint colors to the navigation bars, providing a somewhat unique app flavor while still being comfortable to iOS users.



One 3rd party framework I used was called JASidePanels.[1] It let me have a sliding-panel style navigation menu. I like this approach as I wanted to keep my navigation options open in the future as I add more features to the app. It also increases the viewable space in the main app window, as it does not need to be taken up by a large toolbar. And, it is a navigation style being adopted by major apps such as Facebook and

[1] https://github.com/gotosleep/JASidePanels

Path, so it seemed useful to become familiar with such an approach.
For the UI, I also implemented pull-to-refresh, a increasingly common iOS design pattern that provides an easy way for the user to fetch more data. While in the future, I'd like the app to update based on push notifications, I wanted to see how the pull-to-refresh methods worked in iOS. I also implemented a search bar to filter polls auto-hides, so it does not clutter up the screen needlessly. I sorted polls alphabetically, but could also easily sort them based on their last updated date, which is something I will do in the future.



For the individual poll view, I created custom table cells in order to display the data clearly. Each response is sorted based on the numerical percentage. I was not pleased with the historical view at the bottom of the window and will work on refining that in the future.

When navigation, because I made use of a third-party view controller library, I had to take some extra steps to make sure views were rendering in the proper position while maintaing the users position, so for example, if you scrolled down in the poll list, tapped a poll, and navigated back, you would not lose your spot in that list.

I also added a custom icon and custom loading screen to the app, with all the proper sizes for various iOS devices. I used iOS Storyboards to do the bulk of my UI work, though created some views and layouts entirely in code.

Architecture
Because my app is almost entirely dependent on a remote API, I knew that managing my JSON requests efficiently was very important. Because Apple's JSON libraries are minimal, I used the 3rd party library RestKit[2], an Objective C framework built to make working with RESTful resources simple. The library was very helpful, but it's a bit 'heavy', so it was challenging when I wanted to diverge from the path the library provided. But, the library's networking code regarding threading and blocks worked very well, which was the point of adopting it in the first place. RestKit was useful in mapping the JSON response to Objective C objects, and provides hooks to update the UI when data has been properly or improperly fetched, so I could load up the table view accordingly. It also provides a simple way of mapping fetched data to Core Data objects, which is something I will adopt for future releases.

Even though the amount of data fetched per request is minimal, I tried to minimize data transfers when possible. The app loads by fetching a summary of just the top polls, and only loads all polls when the user explicitly asks for them. Additionally, when the user taps to view a poll, only then is the historical data for that poll loaded. That data is

---

[2] http://restkit.org

attached to the poll, so if the user taps that poll again, the app does not execute another request.

Testing
I used the Testflight service to distribute my app to some friends to test it out. Testflight makes it convenient to share an installable IPA with minimal fuss. And, it allows me to see when people are using the app, how long they use it for, if the app crashed, etc. Testflight will make future development on the app more convenient.

Challenges
Dealing with RestKit was the hardest part of the app, as if I could not load data, there would be no point of the app. RestKit itself was in the midst of revamping their documentation to support a new version, so it was often tricky to find examples of how to implement it, though I think it worked out well in the end. I had some difficultly dealing with the sliding menu controller as well, but am happy with out that turned out.

The individual poll UI is something that I have been thinking about for months but haven't been able to come up with anything I like. The problems are that 1) polls can have different number of responses and 2) I wanted to show as much data as possible without being too cluttered. I originally wanted to use a 3rd party charting library to display the data. I'm not happy with the simple list at present. However, there aren't many iOS charting libraries and the best one was expensive. If I make a serious commitment to the app, I may either find the money to purchase that charting library or work on developing one myself.

Conclusion
I was pleased with how the app turned out. I want to continue developing it, and will add two main features: the ability to select favorite polls and push notifications when your selected polls are updated. I also want to work on the individual poll UI, and create something that is clean and expressive at the same time.