Mike Caterino
Dr. Jackson
COSC 480W
April 25, 2014

Mobile Fast Facts—Optimization of Products for Mobile

Abstract

With the growing number of users moving from the standard personal computer to mobile operating systems, optimizing your product for the mobile landscape is necessary for continual or sustained growth. A major challenge is for the medical industry to have a quick reference to medical articles on their mobile device that can be easily accessible while meeting with a patient. Expanding upon a simple iOS application, this application allows for medical articles to be viewed on the fly—without the need for an Internet connection. This project shows how someone with a general object-oriented computer programming background can create an iOS application with a small learning curve.

Background

With current technology trends, mobile applications are here to stay, and it is key for programmers to adapt to the trend. One of the reasons that mobile applications are becoming such a monumental trend is that many fields are trying to get their brand out to everyone. One field that is starting to move to mobile applications is the medical field.

The starting point for my work was a simple iPhone application that was created in the Fall 2013 offering of Systems Analysis and Software Design for Dr. Mark Zhang. The application, which was Version 0.3.0, had a Web 1.0 style to it—with its home screen as simple text buttons with a small graphic, two table views (which look like simple scrollable lists) for finding articles, and a simple web viewer with no extended capabilities to view the articles.

Dr. Zhang is the Internal Medicine Chief Resident at Allegheny General Hospital. The primary field that he works in is end of life and palliative care. Palliative care tries to provide patients with relief from the pain and stress of illness and improve their quality of life—especially for those with grave illnesses. The current source of information for this field is the

End-of-Life/Palliative Education Resource Center's (EPERC). On their website they have a "Fast Facts" section that has over 270 articles on this topic. The biggest downfall is that there was not a mobile friendly version of the website nor a mobile application. This was Dr. Zhang's goal—to have a mobile friendly version of the articles that would work offline. He also wanted to include other features available on the desktop website like browsing the articles by their subject and the ability to search the body text of the articles. Moreover, Dr. Zhang wanted to have more features added such as a "Toolbox" section for conversions of dosage between medicines and a "Case of the Month" section featuring the University of Pittsburgh's Medical Center's (UPMC's) Palliative Care Case of the Month, which highlights articles published in the field of palliative care that are not featured on EPERC's website.

My goal was to extend the initial application to be like other mobile medical applications, such as MedScape, providing a complete solution for those in the Palliative Care field.

Version 0.3.X

The goal for Version 0.3.4 was to fix bugs left in the application that was inherited from the Fall class. This included things such as removing nonfunctional buttons, adding functionality to nonfunctional buttons, removing placeholder text, removing a disclaimer splash screen, creating an "About" section, and doing some minor layout work. Moreover, without a dedicated iPad version, there were issues with scaling the iPhone application to the iPad. More revisions were done to help scale out the app until an iPad version could be created and the app became universal. The overall goal for this version was to clean up what already existed. Figure 0.3-1A contains after screenshots of the application and Figure 0.3-1B contains before screenshots.

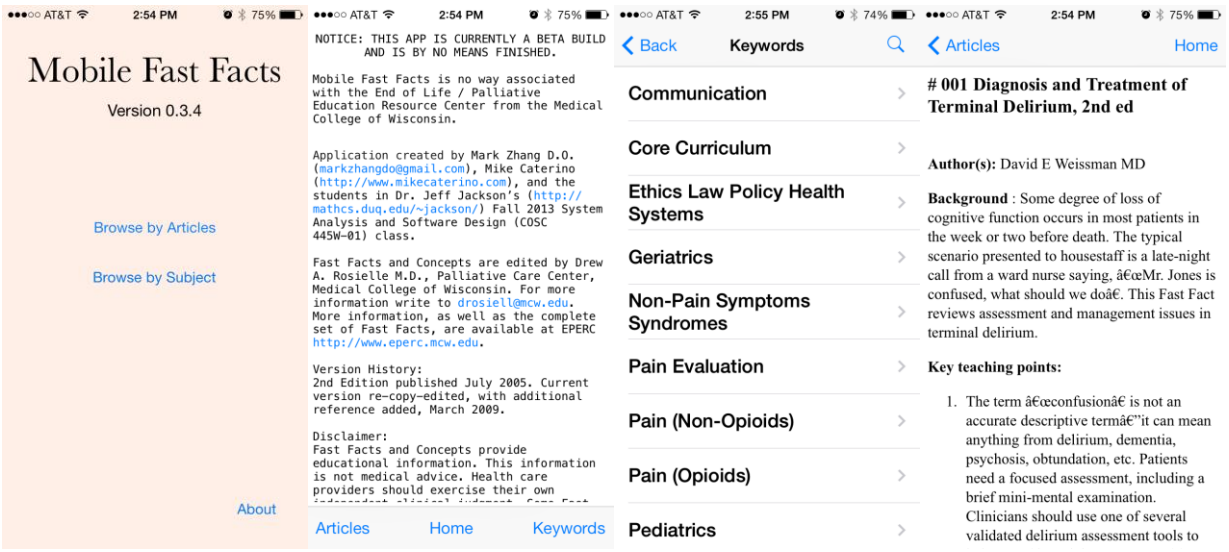

Figure 0.3-1A: Screenshots of the Application as of Version 0.3.4

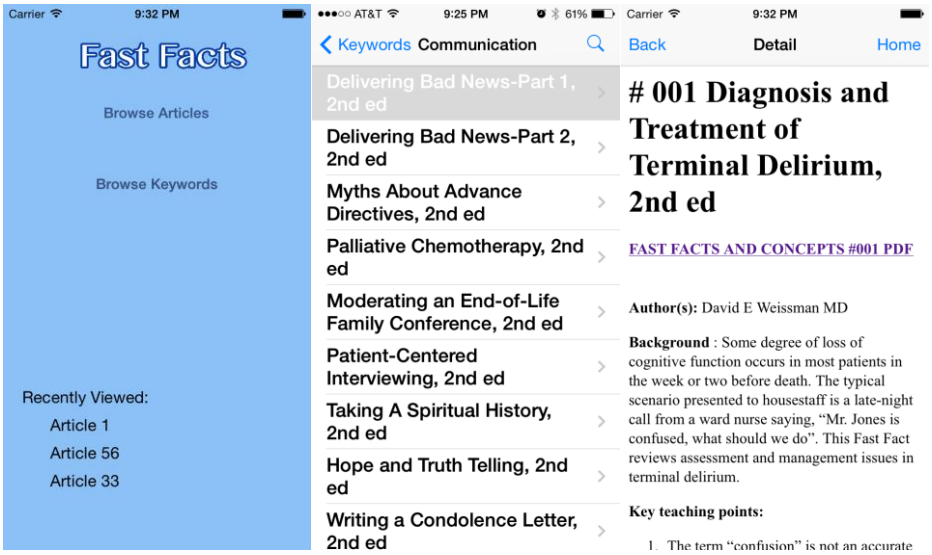

Figure 0.3-1B: Screenshots of the Application as of Version 0.3.0

The leftmost photo of Version 0.3.4 highlights the main page, with includes the version number at the top. The information was included because Dr. Zhang wanted to send the application to others in the Palliative care to spread the word and get feature and application feedback. This version number automatically updates based on the version number given to the application in its info file. This was to highlight the fact that this was not the final look of the design. Moreover, with the added user base, there needed to be a way to distribute it to others without putting it on Apple's App Store. This was done by using a third party API called TestFlight, which allows me

to wirelessly distribute the app to allowed users, track app usage and get crash reports automatically sent to me for further work in bug fixes.

Another goal was to create a new application icon that looked more like a logo and less like a plain text icon. The before and after comparison can be seen in Figure 0.3-2

Figure 0.3-2: Old application icon on the left with the new on the right.

One of the major bugs in the application was that the application did not load the articles when connected to a Wi-Fi network that did not have an active Internet connection. When this case occurred, the article page would show a blank page. I discovered that the reason for this bug was a line of code in the article file. To take a quick step back, the way that the application works is that every article is its own hypertext markup language (HTML) file. These webpages were scraped from EPERC'c website as-is. Their website uses JavaScript on their pages and therefore the mobile application was trying to run the JavaScript code before displaying the website. Normally, this does not cause a problem; however, there was one line of code that was hosted by a remote server, thereby causing the application to try to load the external code. The simple, one line of code is shown in Code Snippet 0.3-1.

```
<script src="http://www.google.com/recaptcha/api/js/recaptcha_ajax.js" type="text/javascript"></script>
```

Code Snippet 0.3-1: Troublesome JavaScript Code

The code snippet is for a spam deterrent system in form submissions; however, there are no forms to be submitted in the application, therefore the code was superfluous.

Version 0.4

The goal for Version 0.4 was implementing the "Case of the Month" section along with various bug fixes and issues with the auto layout feature when using the app on iOS 6.0+ or on the iPhone with the 3.5 inch screen. Bug fixes included creating actual full sized icons instead of

using buttons and correcting a resizing issue when going to landscape, as seen in Figure 0.4-1. Once these issues were fixed, along with layout issues, support was brought back to iOS 6.0+ devices. Another issue with the layout of the articles was simply a missing line of code in the HTML files explicitly defining the character encoding as UTF-8, as illustrated in code snippet 0.4-1, so that it does not use the UIWebView's default of ISO-8859-1.

```
<meta charset="UTF-8" />
```

Code Snippet 0.4-1: Necessary Character Encoding Code inserted into the <head> of all HTML documents.

Another addition was the Next and Previous buttons which shows the respective article based on article number. So if a user is reading article 7, then the previous button will take them to article 6 and next will take the user to article 8. Also, the Next and Previous buttons will disable if there is not an article to go to, as shown in Figure 0.4-1. In addition, when a user would click on a link inside of an article, the next and previous button were referencing the old view. This bug was also fixed. Therefore, if the user is reading article 7 clicks a link to article 43, the previous button will reference article 42 instead of 6 and next will reference 44 instead of 8.

Moreover, the "Case of the Month" section was added to the application. This section simple displays the PDF file for each of the cases and then incorporates multi-touch features such as pinch-to-zoom and the double tap to zoom in or out of a section. This section, in landscape mode, can be seen in Figure 0.4-1.

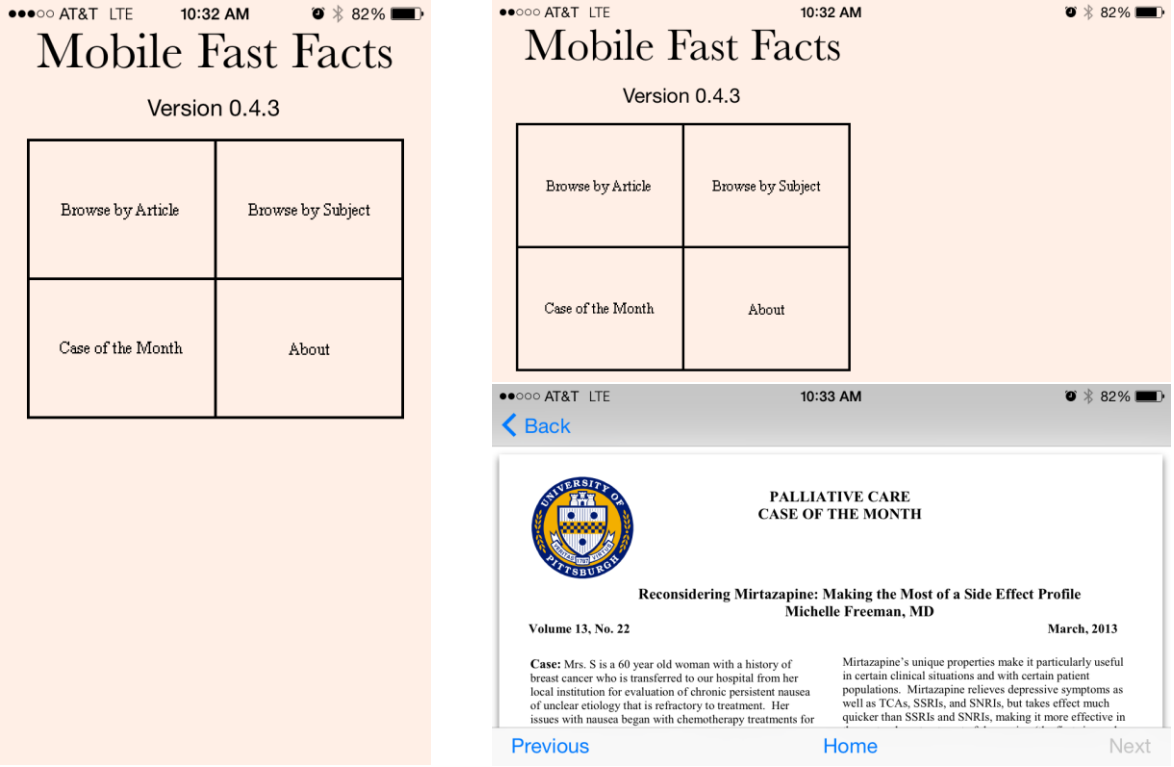

Figure 0.4-1: New buttons on home screen (left), landscape view of home screen (top), landscape view of the Case of the Month (bottom)

The user can also double tap inside of the Case of the Month and the article's display controller to show or hide the navigation bar, toolbar, and status bar through the use of code snippet 0.4-2.

```
[[self navigationController] setNavigationBarHidden:YES animated:YES];
                                [self.navBar setHidden:YES];
[UIView animateWithDuration:0.3 animations:^{ self.webView.frame = CGRectMake(self.webView.frame.origin.x,
  self.webView.frame.origin.y, self.webView.frame.size.width, self.webView.frame.size.height + 88); }];
```

Code Snippet 0.4-2: Hiding the Navigation Bars

This code tells the application to hide the top navigation bar and the bottom navigation bar navBar, that was created in the storyboard. The last line of code modifies the UIWebView, which is the view in which the articles are displayed, to expand to the full size of the view over the course of 0.3 seconds. This is the full size of the view because the bottom toolbar has a height of 88 and the top is automatically resized with the first line of code.

Version 0.5

The primary goal for Version 0.5 was to create a "full body search" for the application. The "full body search" returns a list of all articles containing a user-specified search term, sorted in descending order of number of occurrences of the term in each article. The original approach was to load the full text of each article into the database and have the search results delivered by an SQL query. After testing the efficiency of an SQL search versus storing the contents of the database in memory, I went the route of storing the contents in memory since it saved anywhere from one to three megabytes of memory which is equivalent to a three to ten percent savings. With this method, I can use the NSPredicate method, which is like the SQL WHERE LIKE, allowing the data to be searched nearly instantly. This method returns a list of articles containing the word, which are quickly scanned to obtain the number of times the word, occurs within each article. Then a quicksort is used to sort the data by number of occurrences and the search returns the results. This all happens near instantly with a low spike in memory (from around 20 megabytes to around 27 megabytes). The search query also highlights the text in the article's view using JavaScript code that modifies the display of the webpage using the Document Object Model (DOM). The Javascript code identifies the text in the HTML file and highlights it.

Another addition to the application was a tutorial that displays on the user's first run of the application. To implement this, a new controller was added, FirstViewController, which is the first class that is called after the initialization of the application every time. The controller checks for the key "firstRun" inside of the "standardUserDefaults" object of NSUserDefaults. [NSUserDefaults standardUserDefaults] allows the application to store data on the phone's disk—therefore, once the application is run the first time, the key is changed to notify the application to no longer display the tutorial. They way this happens is through code snippet 0.5-1.

```
[self performSegueWithIdentifier:@"toHomeViewController" sender:self];
```

Code Snippet 0.5-1: Instantly Push to Next View

The method "viewDidLoad" in the FirstViewController checks to see if the key "firstKey" has a value. If it does, then it calls the line of code above, which pushes to the segue "toHomeViewController" which was defined the in storyboard to take the FirstViewController to the HomeViewController. This code stops the FirstViewController from being displayed and it

goes directly to the HomeViewController. Furthermore, to prevent any excess use of memory, nothing else gets loaded in the FirstViewController if that key is found.

Settings were also added to the application, which allows the user to configure the display of the article view. Inside of this view the user can choose the font size of the articles and a color scheme. The color scheme choices are Black on White, White on Black, and Peach (the color of the application). The user's preferences are stored in the same manner as the "firstRun" token and are loaded as soon as the article view is displayed. If no settings are chosen, then the application defaults to Black on White with a font size of 100% for iPhone 4S and lower, and 120% (which displays as 100%) on iPhone 5 and up. This also applies to iPods with the same screen size.

```
if([[UIScreen mainScreen] bounds].size.height == 568)
```

Code Snippet 0.5-2: Detecting Screen Size

The code detects the user's screen, checks the height of the screen, and if it is the height of devices starting with iPhone 5, it modifies the default font size and updates the equation that displays the current font size. The settings are reflected in the article view controller through methods defined in the SettingsViewController class that return the user's font size and color scheme choice. The article view is then updated through JavaScript to modify the look of the articles. The user can also pinch to zoom on the article view itself to dynamically resize the font size. This also saves the value for the Settings.

Furthermore, an email feature was added to the article view. Now, a user can click an email button at the bottom of an article, shown in Figure 0.5-1, which will extract the HTML file from the article view and mail it to whomever the user chooses. This email also includes text to download the app and the subject "A Fast Facts Article has been Shared with You." The email button, along with the next and previous buttons, are disable automatically if the user navigates to a page on the Internet and not an article on the device.

Moreover, multiple bugs were fixed in Version 0.5. A few such glitches were the navigation bar and toolbar not displaying correctly where they should or displaying when they should not. This was fixed by modifying the root Navigation Controller, which controls all views of the application, so that it only pushes the navigation bar onto the controller's stack and not the toolbar. Then inside of the sub-view, or in the subsequent view controller classes, the navigation

was explicitly defined to either show or not show using code snippet 0.5-3 and the toolbar was explicitly defined in the Storyboard.

```
- (void)viewWillAppear:(BOOL)animated { self.navigationController.navigationBarHidden = YES; }
```

Code Snippet 0.5-3: Hiding the Navigation Bar

Code snippet 0.5-3 gets called just before the sub-view is going to be pushed on the navigation controller's stack. In this case, the line tells the root Navigation Controller that the default navigation bar should not be shown. This code snippet fixed bugs such as views becoming misconfigured when a user would double tap on an article which would show/hide all navigation bars and the status bar at the top or click on the global search on the home screen which would shift the contents of the view down and add a blank navigation bar at the top of the view. This is shown in Figure 0.5-1.

Figure 0.5-1: Display Bugs

Version 0.6

The creation of the iPad application was extremely simple since there was already a fully coded iPhone application. The main difference was that in creating a universal application, Xcode automatically adds code snippet 0.6-1 in AppDelegate.m:

```
if ([[UIDevice currentDevice] userInterfaceIdiom] == UIUserInterfaceIdiomPad) {
    UISplitViewController *splitViewController = (UISplitViewController *)self.window.rootViewController;
    UINavigationController *navigationController = [splitViewController.viewControllers lastObject];
    splitViewController.delegate = (id)navigationController.topViewController;
```

```
}
```

Code Snippet 0.6-1: Automatically Added Code

However, this code is actually very problematic. What this line of code does is tell the root Navigation Controller to create a split view for all views—which is how the iPad creates tables as seen in the default Settings or Mail applications. This allows for the table to be scrollable on the left third of the screen and the detail to be shown on the right two-thirds of the screen. Since this is not the look we were going for, the above line of code needed to be removed so that the Navigation Controller could handle the stack the same way for iPhone and iPads. This allowed me to copy the storyboard for the iPhone and scale items appropriately so that the look would be better optimized for the higher resolution display. Therefore, the app became a Universal app for all devices running iOS 6.1+.

Other small modifications were to the default font sizes for iPads, starting at 140% of standard iPhone, and to the tutorial, scaling it for the iPad and changing the images depending on what device is being used. The rest of the features and functionality are the same as the iPhone.

Another feature added in Version 0.6.4 was the ability to shake to remove highlights from search and the ability to disable this feature and the highlighting feature. Also, a bug where the search query would highlight text in the referenced article was also removed. This occurred when the user pressed the Next or Previous buttons or clicked a link to a referenced article.

An additional feature added was "Metrics." When a user clicks on an article, the app logs it using TestFlight's Checkpoint feature, which will anonymously log the article that the user has viewed. This is viewable by Dr. Zhang and whomever he grants access to, as well as myself.

Finally, four bugs pertaining to the article view were fixed. When a user would click through to an external website, the next and previous buttons were available, as well as the email article. Since it was an external website, it thought article 0, which does not exist, was being viewed. It is now set to disable those buttons when an external website was viewed. Moreover, if the user modified the font size or had highlight enabled, it would break the external websites view. Therefore, these features are disabled and default controls are enabled—i.e. when an external website is clicked, the website is handled and has the same features that are available in the default web browser (Mobile Safari).

Version 0.7

The goal of Version 0.7 was to finalize the app for the AppStore. Since we did not receive complete copyright approval for the Case of the Month section, it was removed and in its place is Article of the Day. When a user clicks the Article of the Day button it checks the current date (in dd format). If the user has never clicked the button before, it saves the current date in the same manner as Settings. Then it randomly generates a number between 0 and the latest article number minus one, currently 272-1=271, and then saves that number in an NSMutableArray. If the user has clicked an article before it checks the dates—if they match, it displays the current article; otherwise, it creates a new random article that is not in the array of previously viewed Articles of the Day. When the user has viewed all articles, it resets and starts over again. These articles display in the same controller as the other articles.

The other feature added was a Tips section, which displays all of the tips shown in the first run tutorial. This is accessible in Settings and About.

Closing Remarks

At the end of this project there was one very important thing that I learned—take control. When I say take control, I mean do not rely on the built-in functionality buttons, view controllers, etc. The majority of my frustrations came when trying to switch between default interactions and custom interactions. The best thing to do is take control of the object's delegate and explicitly define the default actions that you want to keep while adding your custom interactions. Moreover, you can always switch the delegate back to the default and take control back on certain interactions. For example, when the user clicks an external link, not only do all of the buttons disable, but the delegate is given back to the system so that the UIWebView acts the same was as Mobile Safari. When the user goes back to an article, the app takes control back of the view to allow for my modifications. Originally I was trying to do both, but this was the easiest way to control functionality without all of the bugs.

Some things that I wish I could have done were create a landscape view, an iPad specific layout, optimized code, Toolbox section, and implement the Recently Viewed code that started to be created in the Fall 2013 Systems Analysis class. The reason there was not

a landscape view was due to the layout of the buttons, since the view would cut off the rest of the buttons and there is no scroll available for the Home Screen. I wanted to create a scrollable home view, or a nicer layout that would also be optimized for landscape; however, time did not allow for it.

Moreover, I wanted to create a iPad specific layout so that it did not look liked a scaled version of the iPhone/iPod app; however, with the lack of time to do constant layout changes and feature additions and removals, there was no time to just focus on the iPad layout.

Another thing that I wanted to do was finish optimizing the code for older iPhones and iPads that have limited memory. Currently the app lags slightly on older iPhones, iPhone 4 and less, and on iPad 2 and the iPad mini. While I was improving as I went, I know that certain features, like the quicksort, could be better optimized, but the rush to get features added and removed took precedence.

Furthermore, the original plan for the app included a Toolbox section, but again due to constant design changes and other feature additions, like detailed settings, and unexpected bugs in full body search, this section was not implemented. Moreover, the section was not considered as important to Dr. Zhang as other features like highlighting search text, and having a "clean design" so that the app could be submitted to the AppStore, which was submitted as of today.

Finally, I wanted to incorporate the code for recently viewed articles that was included in the app received from the COSC 445W class, but it was not fully functional. While the code was improved slightly (some results were generated rather than no results), there was no place for it on the home screen. With the Metrics section, all of the extra code was superfluous and therefore memory costly.

For more information on the app, check out mikecaterino.com/fastfacts or download the "Palliatice Care Fast Facts for iOS" app on the AppStore (currently in review) at https://itunes.apple.com/us/app/palliative-care-fast-facts/id868472172.