Abstract

In this project my primary goal was to apply and extend my knowledge of computer science into the realm of mobile software development. I wanted to get a feel of the technicalities and challenges faced by mobile developers, as well as acquire some understanding of the frameworks and tools employed in creating a mobile application. To that end, I implemented a simple mobile application. This allowed me to scope out and narrow down the things that I needed learn, in addition to giving me a way to tangibly monitor my progress.

The most popular mobile development platforms, iOS and Android, seemed equivalent in terms of introducing me to mobile development. I chose iOS as the platform I would explore, primarily because I own an iPhone. I decided to learn Swift as the programming language I would use over Objective-C. Swift seemed the more forward-looking choice while still being accessible, given that Apple has taken a lot of care in creating the new language, and that the online community of developers has had a few years to grow.

In deciding what type of application to implement, I tried to define what I thought was a ‘basic’ understanding of mobile development. I created a trip-mapping application that would enable users to add photos and descriptions, tagging them with different locations that can be displayed on a map. This gave me the opportunity to work with different types of data, implement persistent storage, control navigation through various views, and understand how to modify UI elements, as well as use a third-party SDK and API (Google Maps). This report outlines my approach and describes the design and implementation of the application.

Motivation

For many people today, smartphones are their first and/or only computing device. They have become increasingly significant in our everyday lives, with applications spanning a myriad interests, industries, and objectives. We use them to connect with our friends, get from place to place, and even transfer money. Given the present pervasiveness of smartphones and their likeliness to become even more integrated into our lives and with our other devices, I found it worthwhile for me as an aspiring computer science major to spend some time gaining an understanding of how mobile application development works.
Method

Apple’s developer page provided a number of resources that helped me get started. Reading through Apple’s “Learn the Essentials of Swift”, I got a feel of what the language looked like syntactically and had copied chunks of example code into a file that I could use as a syntactical reference (for-loops, array initialization, switch/case, class declarations, etc.). I also did Google searches for other tutorials, articles, and documentation that would help me learn iOS development through Swift. The article that I found gave me the best big-picture overview of what is going on in terms of how to use XCode and an overarching view of the iOS framework was this one. I worked through Apple’s Getting Started tutorial, which led me through creating a FoodTracker app. I found other tutorials where I would work through the parts that I thought might be relevant and skimmed through the rest. This helped me solidify the things I read, in addition to exposing me to the possibilities of what could be done.

I reached this tutorial showing how to implement a Master-Detail View, and realized that this would be what I wanted to use as the foundation of the application that I would build. I planned out what I wanted in the application and mapped out the process that I would use to get there in terms of what I would do once I started from the Master-Detail View tutorial. Following that, I built the program, relying heavily on StackOverflow and other articles and tutorials to get me through the features I wanted to implement and the bugs I needed to fix. I did not worry too much about the visual appeal of the UI, so long as the functionality was there and I knew how to move things around. Lastly, I learned to write and run tests, which are an important part of real-world application development, even if this application is fairly simple.

iOS

Apple’s iOS is the world’s second most popular mobile operating system after Alphabet’s Android (Edwards). iOS frameworks employ the Model-View-Controller design pattern, which is shown in the figure below. The Model represents how the data is stored and abstracted. The View is responsible for presenting information. The Controller contains most of the logic and is responsible for translating user input to changes in the data and updating data to be displayed in the view.

![Source: iOS Developer Library documentation](image-url)
The UIApplication is responsible for the Event Loop, which allows the application to respond to input from the user such as touch, shake, accelerometer, etc. The Application Delegate is where custom application code tends to begin, with handlers for high-level application events such as when the application is launched, when it is going to be closed, etc. View Controllers manage the presentation of View Objects and UI Objects and determine how the application respond to events on those objects.

**Swift**

Swift was built to be cleaner, faster, and more accessible than Objective-C, while working well with Apple’s existing development tools, including Objective-C. It was released in 2014, and has achieved incredible growth in popularity, becoming one of the 20 most popular languages on Github and StackOverflow within a year (O’Grady). Based on this information, I was confident that the online resources for learning and getting help in Swift were developed enough that I should face no significant barriers to using it in the project, despite it’s relatively new status. This proved true, and I only used free online resources throughout the semester as I implemented my application.

Swift uses type inference, type safety, automatic memory management, generic programming, simple string manipulation, in addition to a host of other features designed to be intuitive, minimize errors, and allow for compiler optimization. This web page contains a good overview of some of Swift’s more distinctive features. Furthermore, the Playgrounds feature that is integrated with Swift in XCode made it even easier to pick up the language. While Swift is a compiled language, Playgrounds allow developers to see the effects of code changes in real-time, making the learning process very interactive and minimally frustrating. Below is a screenshot of the Playground from Apple’s tutorial, “Learn the Essentials of Swift”.

```
let sampleString: String = "Hello"
let sampleArray: Array = [1, 2, 3, 4, 5]
```

**Swift and Cocoa Touch**

Swift is designed to provide seamless interoperability with Cocoa Touch, the set of Apple frameworks you use to develop apps for iOS. As you walk through the rest of the lessons, it helps to have a basic understanding of how Swift interacts with Cocoa Touch.

So far, you’ve been working exclusively with data types from the Swift standard library. The Swift standard library is a set of data types and capabilities designed for Swift and baked into the language. Types like String and Array are examples of data types you see in the standard library.

**Experiment:** Read about types in the standard library by Option-clicking the type in Xcode. Option-click on String and Array in the code above while looking at this playground in Xcode.

When writing iOS apps, you’ll be using more than the Swift standard library. One of the most frequently used frameworks in iOS app development is UIKit. UIKit contains useful classes for working with the UI (User Interface) layer of your app.

To get access to UIKit, simply import it as a module into any Swift file or playground.

```
import UIKit
```

After importing UIKit, you can use Swift syntax with UIKit types and with their methods, properties, and so on.

```
let rectShape = UIView.frame: CGRect(x: 0, y: 0, width: 44, height: 44)
colorSquare.backgroundColor = UIColor.kiwiGreen
```
Specifications

The primary functionalities I wanted in the application were:
- Adding entries
- Displaying entries on a map
- Displaying entries in a list
- Viewing entry details
- Editing entry details
- Deleting entries

An entry should contain the following attributes:
- Name
- Description (optional)
- Photo (optional)
- Location (optional)

Data Model

I started by using simple structs for my data models. Once some basic functionalities were implemented, I learned CoreData and updated the application to use that instead, in order to make my data persistent. Core Data is Apple’s persistence framework which organizes data in a relational entity-attribute model. Data is modeled as Entities, which contain Attributes and relationships to other Entities. I divided what I considered an entry into two different Entities:

Pin:
- Name (String)
- Description (String)
- Picture (Binary Data)
- Related Location (Entity)

Location:
- Place ID (String from Google’s Place Picker)
- Name (String)
- Longitude (Double)
- Latitude (Double)
- Address (String)
- Related Pin (Entity)

While the split between Pins and Locations is hidden from the user, it helps to simplify and better organize the application logic.
Views

I started with a Master-Detail View, where the Master view contains a list of Pin names, that when tapped would update or segue into the Detail view of the information associated with that Pin. The Master-Detail View comes with logic that collapses the views on top of each other in portrait orientation and displays them side-by-side in the landscape orientation or bigger screens like the iPad. I configured the Master View to show the names of the Pins and the Detail View to show the information associated with it.

Figure 1: Master-Detail View in horizontal orientation

Figure 2: Master-Detail View in vertical orientation
I then created the view that would appear when users want to add new Pins. The name and descriptions were simple text fields which were easy to implement. However, images and locations needed additional views for selection. Thanks to this image-picker tutorial and the Google Places API demo and sample code, it was surprisingly uncomplicated to implement both image- and location-picking as extensions onto the AddPinViewController. It was nice to use the Google API and learn to work with Cocoa Pods, which is a dependency manager for Swift and Objective-C projects.

![Figure 3: Adding a Pin, selecting a location (left to right)](image1)

![Figure 4: Picking an image](image2)
After implementing the view to add Pins, it was fairly simple to create a view to edit an existing Pin. Within the edit view, I added a button to delete the Pin being edited. Delete was surprisingly challenging to implement due to gaps in my understanding of how to navigate between controllers- I had a hard time figuring out how to get out of the deleted Pin’s Edit view and get back to the Master view when the Pin is deleted. After I filled those knowledge gaps and solved the problem, I took the opportunity to learn how to create pop up windows. I created a pop up window that would confirm that the user wants to delete the Pin, and one that would show up when the user tries to save a Pin that is not allowed (no Pin name).

![Figure 5: Alert when saving a Pin with no name, confirmation for delete (left to right)](image)

The next thing was to add a toggle feature to the Master View that would allow users to switch between seeing the Pins in a list and seeing the ones with associated locations displayed on a map. After some research, I created a new ViewController for the “Master” scene with a toggle button on top and embedded into it both the original Master View (list of Pin names) and a new Map View. Again, Google’s documentation on its SDK made it quite simple to create the Map View and add Markers to it. The toggle button allows the user to alternate which of the two embedded views is shown/hidden. Lastly, I made the Map view be more functionally like the Master view in that users can tap on the markers to see the name of the Pin, and tap again on the name of the Pin to see the Detail view.
Testing

The final part of my project was to implement tests. Technically, this is something that should be done throughout the application development process. However, as this is a very basic application whose internal structure was changing all the time as I tried to figure things out, I held out on this until I was more confident of what I was doing. In the end, I implemented unit tests primarily so I would know how they work. I checked that adding, updating, and deleting pins work as expected. I learned also that Cocoa pods, which worked very well in the rest of the application, needed a little bit of configuration to get working with the tests. This article was very helpful for that, but I expect that Cocoa pods developers will probably get rid of this problem in future versions.

Discussion

This project was a great introduction to the realm of mobile application development. I achieved my goal of expanding my knowledge in this area, gaining an understanding of the basic tools and frameworks, as well as learning how my existing skills can be applied. I felt that the application I implemented gave me exposure to the different objects, functionalities, and frameworks that would exist in many modern applications. Despite the newness of the language, I was impressed by the online community working in Swift that I could rely on for information and answers to questions and bugs that I found along the way. At some points I still had to try to translate sample code or tutorials in Objective-C to what I would do in Swift, but I never ran into problems that I could not find a solution to (even though some problems may have taken a very long time to solve). I really enjoyed using the language and given its incredible growth so far, it is exciting to
think of the platforms and applications that will run on Swift in the future, especially as it recently became open-source.

Future Work

There are obviously very many possible improvements to the application that I made. I could implement back-ups to iCloud, retrieve geolocation data from the user’s images, add features for sorting and organizing pins, etc. However, I think it served its purpose as a learning tool and it might be more interesting to focus on the things that could further my growth as a mobile application developer.

Different types of tests would be important for developing more advanced applications. Knowing how to create performance and UI tests in addition to unit tests would be beneficial. There is also a lot of room for me to learn what makes a good test in the first place- how to structure them, what types of cases to test, etc.

In terms of technical knowledge, knowing how to back up CoreData objects to iCloud could actually be quite useful. An advanced extension of that would be to learn to build a server-side application that is presented to users through a mobile application. On a similar note, I could learn things that would make an application less of a silo, such as integrating with other apps (i.e. through deep-linking), integrating with hardware (i.e. fitbit or smart devices), or structuring data models to be shared in some sort of social component.

Overall, I found that making views look good was a very tedious and confusing process that I will probably never enjoy. My application is far from glamorous, but I am quite happy I managed to figure out how to move objects around to approximately where I wanted them to be. I realize that this is a major area for improvement, in both deciding how things should be arranged and knowing how to arrange them. With the latter, it would be particularly important to know how to ensure that things would look in multiple platforms and screen sizes. I only worked with the iPhone 6 screen, yet many of my meticulous arrangements disintegrated the moment I changed the screen orientation.

I could definitely benefit from more knowledge about design and how to create an application with intuitive flow between views and an appealing interface, especially for multiple platforms and screen sizes. Not only that, but to design applications that will fit easily into people’s lives as well as take full advantage of the capabilities of the smart phone, such as being with the user almost all the time, or having the user’s location, speed, etc. at a given point in time.

I did not pay much attention to memory or bandwidth constraints in this application, which are key considerations for developing applications that are really efficient. Understanding how to structure the application to use the least amount of power, memory, and bandwidth should be a vital skill for a mobile developer.
Another realm of mobile software development is games, which would be an entirely different approach than the one I took here with the trip-mapping application. That would require a different set of skills, knowledge, and design sensibility that might be interesting to develop. Lastly, even if I prefer iOS, I could benefit from having more exposure to Android development. I could conduct a similar project to the one I did here on Android, and get some insight into how the two platforms compare.

Acknowledgements

I would like to thank my advisor, Professor Ruzica Piskac, for her help and guidance throughout the semester.

Other Links I Found Quite Useful

Application Development Life Cycle:
- iOS Developer Library documentation

Swift:
- Cheat Sheet and Quick Reference on RayWenterlich.com

Core Data:
- Getting Started with Core Data Tutorial on RayWenderlich.com
- Core Data and Swift Tutorial (page on relationships) on envantotuts+
- Apple’s Core Data Programming Guide on relationships and their delete rules

Navigation Control (Segues):
- iOS Developer Library documentation on Segues
- StackOverflow post on how to pass data from one ViewController to the next

Alerts:
- The StackOverflow answer that is basically a tutorial

Tests:
- Apple documentation on testing with XCode
- Better Unit Testing with Swift on masilotti.com

References
