One Sentence A Day: Application of iOS Mobile Development

Abstract

For my CPSC 490 project, I hoped to apply my experiences in the classroom to a practical platform. Much of my CS education at Yale has been theoretical. My goal was to bring this background in Computer Science into mobile software development. I hoped to learn about the app development cycle, the various frameworks, tools, and languages available, and ultimately practice implementation. I had no experience in any mobile app development, so to shape my progress in this realm my goal and final product was a straightforward mobile application. As a Computer Science major, understanding the app development cycle would provide me with a new set of tools with which I can apply my CS foundation.

Most mobile apps are built on Android or iOS. For this project, I built an iOS app. iOS is an extremely popular platform, and it also is the platform that will be easiest for me to work on given my current resources (I personally have an iphone and a MacBook.). Applications on iOS are primarily built with Swift or Objective-C, neither of which I had used before. I decided to write my project in Swift, because I was excited about the potential of this newer, well documented language.

For the content of my application, I was inspired by the app “One Second Every Day”, in which users add one second of video to their application each day, easily tracking their year. I decided to create “One Sentence a Day”, that would encourage users to write one sentence describing their day each day, with the opportunity to add photos. Users can create, edit, and delete these entries, and view them sentence forward or photo forward. They can also choose to be reminded (or not) to write a photo each day.

Through my project, I was able to learn a new language (Swift), a new environment (XCode), and the Model-View-Controller design pattern. I explored the principles of UI, testing, and data storage. I also had the opportunity to work with third party frameworks to build upon pre existing projects.

Background

Today, many people’s daily lives could not be completed without smartphones. We use these smartphones for basic activities, like transportation (Uber, Google Maps) and food (Grubhub), to more recreational activities, like music (Spotify) and social media (Instagram).
Smartphones can connect people to their surroundings and to others through mobile applications.

Apps that allow users to track their own lives, privately or publicly are becoming increasingly popular. This is the entire point of social media platforms such as Instagram and Facebook, as well as the One Second Every Day app. People want an easy way to remember their lives. Twitter, Instagram, and many other platforms offer public facing versions of this, but I was interested in creating a private version.

I have always been interested but struggled to keep up with a diary. I find remembering to write in the diary each day difficult. Many of my friends have had the same problem. My photos and texts provide a detailed log of my history, but I wanted a lightweight way to catalogue the passing of time— simply a sentence and perhaps a photo. This would allow users to capture the passing of time with little enough effort that they could do this consistently.

**Process**

To begin learning how to create an App, I researched Swift and Objective C, two languages I had never touched before.

My initial learning came from two Lynda courses, Swift 4 Essential Training and iOS App Development Essential Training. iOS App Development Essential Training taught me the basics of how to download and setup XCode and the simulator and use the various tools XCode, the Swift Integrated Development Environment, provides. Both of these guides taught me some of the basics of Swift as well.

After these, I began on Apple’s “Learn the Essentials of Swift Playground”, which can be found here. The Swift Playground app of XCode was very helpful, as it allowed me to get familiar with the language with real-time results from the code I was writing. I was also able to assemble a sort of reference guide with this playground, on topics such as basic types, classes and initializers, control flow, structs, functions and methods.

Next, I wanted to put what I had learned into practice, so I followed Apple’s tutorial on making a meal application. Through this, I learned how to build basic UI, though the storyboard, by connecting it to code, and later programmatically. I also learned about Autolayout.

Next, I learned about table views, navigation and segues, and editing/deleting behavior for my data. This was an incredibly useful start to my Swift experience. The tutorial was not always entirely clear, so I also found Stack Overflow very helpful. I learned from this tutorial and others that I found online how to implement Unit Testing. This is an important coding practice, especially as projects scale larger than this application.

I also researched and practiced implementing Core Data, Apple’s persistence framework which organizes data in a relational entity-attribute model.
In order to be successful in building an application, I had to gain a strong understanding of the different elements involved. The Food App tutorial, as well as the Apple guides linked to this tutorial, were both very helpful in this.

The approach I followed for building my application was the “model-view-controller” design pattern. The model manages data and abstraction. The view is the representation that the user sees. The controller responds to user input and creates commands to send to the model and the view.

One important player is the App Delegate. Managed through the source file, it creates the window upon which the application projects the content, and allows state transitions to occur. It begins entry point or launch of the app, and the run loop which processes events to schedule and coordinate them. The UIApplicationMain attribute does the work of these events, allowing the app to react to the user. View Controllers manage views (UI/content display) and coordinate the flow of information between different parts of the app (ie the data model and the views).¹

Throughout the process I also reviewed previous 490 Projects, particularly Nathaya Darakanand’s project Mobile Software Development through iOS and Swift

Specifications, Data Models, and Frameworks

In this app, I wanted the following functionality:
- Users can add entries
- Users can view the entries, either from an editable table or a non-editable photo viewer.
- Users can open up entries and edit details
- Users can delete entries

Entries consist of:
- The sentence/text
- Optional photos (which users could take in the app itself or load from the camera roll)
- The date

In addition, users should be able to turn on and off daily notifications that would be triggered at a time of the users choosing if they do not yet have a sentence for that day.

I used a variety of different methods to store this data. The entries were stored in Core Data, so that they could persist between sessions. I stored an attribute here that would flag whether or not a photo exists, as well as a potential file path to retrieve that photo. The photos

within the entries, then, were converted to PNGs and written to the disk. They were then retrieved when needed.

In building my tables, I needed to be able to sort my entries into sections marking months and years. I created my own structs for this, where I sorted the entries into by their section, and hashed the Core Data item by the date of the entry for fast retrieval.

For my notification framework, I found that userDefaults was the best option. I needed the information about whether or not notifications were on, and what time they were set to, to be persistent. Because they were lightweight (a Boolean and a few Ints), it didn’t make sense to put them into core data either.

I built the UI and debugged as I went, making great use of print statements and Stack Overflow.

I also implemented different frameworks, which was a very helpful and practical tool.

- **Chameleon Framework**: This was a powerful color framework, though sometimes too powerful. I used it in my storyboard and programmatically.
- **DLLocalNotifications**: This framework wrapped the new UserNotifications framework of Swift 4. It made these local notifications easier to handle, easy to configure repeating notifications, the basis of my notification specification.
- **ImagePicker**: I used this framework to let users select or take photos for their entries. I had created my own image picker, but preferred the UI of this framework.
- **SwiftyPickerPopover**: This framework allowed a customizable pop up to appear at the click of a button. I used this to allow users to select the time of day they wanted their notifications to ping.

**Application Walkthrough**
The user opens the app to the following homepage:
The user is guided to the above screens. If no sentence has been added, they are prompted to add one. They can go directly to the entry editing screen with the edit button, to create a new entry if there is no entry for that day, or to edit the entry if one has been written. The user can also see how many days in total have been added, and the first date that was added. They can toggle on/off notifications and by clicking the red time button can adjust the time of the daily local notification timing (permission for which is requested when the app is first opened).

Add Sentence
Write your sentence!
When entries are added, they look like the notification above.

Users have the option to go to two other views: the text view and the photo view. We will examine the text view first.

![Fig 2a (left) and 2b (right). The text table view populated with entries.](image)

The table view shows the most recent entries on top. It uses the data structures I created to create and populate the correct sections. Users can edit cells with the edit button to change the content. They can slide to delete and entry. They can also create a new entry with the plus symbol in the top right.
In the editing page, the user can enter text into the sentence area by clicking it. They can also use the datepicker to change the date, and confirm that change with the button “change date”. The picture/icon in the top right corner opens the image picker. Users can cancel and abandon changes with the cancel button. They can also select “save” which either updates the entry or creates a new one. The viewcontroller loads the dates of all pre-existing entries to stop a user from accidentally overwriting a previous entry.

If an entry is added for a day, the notification that would have gone off that day is deactivated. Notifications will restart the next day at the user chosen time.

Now we will go to the Photo View, which a user would get to by returning to the main menu.
In this view only entries with images are shown. Otherwise they are organized the same way as the text table, with the same sections and newest entries first. Entries are not editable from this page, because it primarily is for more easily viewing the added photos.

**Fig 4, Photo View**

In this view only entries with images are shown. Otherwise they are organized the same way as the text table, with the same sections and newest entries first. Entries are not editable from this page, because it primarily is for more easily viewing the added photos.

**Layout**

Autolayout was very difficult to work with and often behaved unexpectedly. Ultimately however all of my views work on different devices and in landscape. I handed some of the bugs with a combination of using the storyboard and editing constraints programmatically. See figures below:
Today's sentence:

Example sentence! This is an example. I am making it longer to show how the text boxes change. They won't cut off the text. Instead they will grow with the text.

Notifications off
Total Days added: 6
First Date: 05/03/15

Main Menu

May 2018

3  Example sentence! This is an example. I am making it longer to show how the text boxes change. They won’t cut off the text. Instead they will grow with the text.
   Edit

2  You can write sentences in entries with photos to record more!
   Edit

April 2018

3  Short example sentence!
   Edit
Testing and User Testing

Because I was learning as I built, I did not start with unit tests and added them in later. In the future I would implement tests as I go. For more efficient unit testing, and better code, I refactored my code to create a class, AllEntries, that handled adding, deleting, fetching, and editing entries, as well as writing images to disk. This allowed me to test these methods in my unit tests. I specifically tested that entries were added with the correct information, were properly added, and were properly deleted.

I also updated my app in response to user feedback. From this feedback, I created a main menu and re-ordered some of the layout of the text table boxes. I also created sections in the main menu for better readability. I let this user feedback guide my feature decisions. For example, my users said they were more interested in flexible notifications than a screen that would lock the app from outside users, which I had previously been considering.

Discussion

I enjoyed my time creating this project, though at times I found it frustrating. Small UI elements would take hours to debug, and errors would sometimes happen inconsistently. Not infrequently, XCode would self quit with no clear bug, at times losing my work.

Ultimately, I really enjoyed putting my experiences to a practical outlet. I know that in the future creating a Swift App would be immensely easier because of my experience.

In the future, I could see taking this app in many directions. I would love to spend more time studying UI to improve the elegance and ease of use of the layout. I would like to incorporate a login, with backed up data, that would allow users to log into their account on a new phone or on the web. This might include another data store system, such as Realm.io or Firebase. I would also love to incorporate a text system. Notifications are often ignored but texts
aren't. It would be great if users were texted when they had not uploaded an entry, and could upload it simply by responding to the text.

In general, users have many options when it comes to applications to track their lives. I think this app could be improved while staying successful by sticking to the simplicity, basing entries off of one sentence and one photo.

Acknowledgements

Thank you very much to Professor Piskac for her guidance and support on this project. Thank you to the people on message boards like Stack Overflow who spend their free time answering insanely specific questions– without their help I would not have made it this far in the application. And thank you to Guayaki Yerba Mate, who provided me with the caffeine I needed, and to everyone who has ever delivered one to me and made me feel very loved.

Other Useful Links
- Segue Guide
- Sectioned UITableView Help
- Core Data Tutorial
- More Core data help
- Editing cell from edit label inside of cell (trickier than it seemed!)