Abstract

For my senior project, I aimed to learn the basics of iOS application development through Swift and XCode, as well as supplement this core knowledge with best practices in regards to mobile design patterns and software engineering principles. To this end, I created a habit tracking application that leveraged third-party libraries and databases.

The application I worked on throughout the term, Betterment, allows users to select, edit, and track habits they wish to enforce and visualize on a simple calendar layout. The interface displays dates a user has successfully completed specific habits, and also keeps track of the number of completed days. Altogether, the application leverages key iOS frameworks such as UIKit, Chameleon, JTAppleCalendar, and many others to create a smooth, elegant user experience.

I was able to exercise mobile design patterns and software engineering principles throughout the implementation of Betterment, namely the Model-View-Controller paradigm. Furthermore, I made use of storyboards, navigation, and presenters to create fluid transitions throughout the onboarding and main application phases. Finally, it was essential for me to become proficient in the ViewController lifecycle in order to ensure that functions and UI components were called and displayed appropriately.

To achieve the design goals I set out in my initial proposal, I also spent a considerable amount of time familiarizing myself with XCode, the interface builder, AutoLayout, and manual layout concepts to create a coherent, sensible user interface that adheres to Apple’s design philosophy.

Future work would add additional tracking features to the app while implementing robust UI, integration, and unit tests. Key challenges I encountered were optimizing application performance, fleshing out UI bugs relating to Apple’s UIScrollView object, and working with asynchronous DB requests.

Motivation

In a survey conducted through January 2018, the Pew Research Center found that 77% of adults in the United States, or over 250 million individuals, own a smartphone. This figure was
as high as 94% for those in the 18-29 age group, which underscores the prevalence and importance of smartphones in our society today. The power and convenience of these mobile devices have revolutionized how we carry out our day-to-day activities, allowing us to hail transportation on demand (Uber, Lyft), navigate efficiently to our destination (Google Maps, Waze), book lodging (Airbnb), shop online (Amazon, eBay), as well as countless others.

Leading the way is Apple and iOS, which holds a 56.9% market share in the United States among all mobile operating systems. Given the widespread use of iPhones and Apple devices among my peers, I aim to dedicate my senior project towards learning about the full iOS application development cycle in order to design, implement, and ship an application to the App Store.

**Learning Trajectory**

Beginning in February, I started a Udemy course taught by iOS developer Angela Yu. The course has over 15,000 5-star ratings online, and after scanning the syllabus, felt that it was naturally a good place to start my foray into iOS development. Over the next couple of weeks I received a comprehensive overview of:

- XCode
- Interface Builder
- Working with UI elements
- Gesture handling
- Swift Playground and basics for testing
- View Controller lifecycle
- AVFoundation and AVAudioPlayer
- Error handling
- Debugging in XCode
- Model-View-Controller design patterns
- Animations with UIKit
- Persistent data storage with Core Data and Realm
- CocoaPods for third-party dependencies
- Basic Networking with AlamoFire
- Delegates and Protocols
- Navigation and Segues
- UserDefaults for handling small, data storage
- Version control with Git
- Asynchronous programming

Learning the Swift programming language turned out to be far more straightforward than I anticipated, namely because of its type-safety and similarity to object-oriented languages I’m comfortable with, such as Java and C++. Some key, new concepts that were introduced were
completion blocks for handling events and optionals for handling null types. Out of curiosity, I briefly dabbled with Objective-C to understand different ways to implementing the same functionality, but quickly realized that Swift is a far more modern, readable, and easy-to-use language with copious documentation online.

Perhaps the most critical stage in my iOS learning was using the various Apple view components that manage how data is displayed via cells or other means. I learned about the tradeoffs and intricacies of UITableViews, Stack Views, UICollectionViews, and the requisite delegate methods and protocols necessary to properly implement these various containers. To practice this, I worked on a Book Review project that allowed me to focus on good iOS fundamentals: clear navigation flow, bug-free UI, and managing asynchronous calls to a DB with Grand Central Dispatch:

I supplemented the videos and practice projects provided in the course with the iOS Programming: The Big Nerd Ranch Guide book, as well as tutorials from RayWenderlich.com and various YouTube channels I’ve listed in the references section at the end of this report.

For the design and user experience portion of my learning trajectory, I mainly read through the Apple Human Interface Guidelines and learned about proper layouts, recommended palettes and typographies, as well as best practices for selecting app icons and creating elegant views. While not directly related to iOS development, I also developed a design scheme that adheres to Google’s Material Design requirements.
Altogether, over the course of 6 weeks of guided and self-learning as well as numerous practice projects, I felt well-equipped to begin designing and implementing Betterment, the application I had detailed in my initial project proposal.

**Application Overview and Commentary**

Beginning in mid-March, I began creating mocks for Betterment using the Sketch application of the user stories and screens I would want to create for my application. I knew that I would want a simple, clean onboarding flow, as well as a scrollable calendar displayed on the main app page. Finally I wanted there to be a way for the user to quickly navigate between the calendar view and their list of tracked habits. This led to the development of the following screens:

![Figure 2: Onboarding Screens](image)

The two onboarding screens above are only displayed once when a user launched the application for the first time. Upon completion of this flow, I made use of UserDefaults to ensure that the onboarding flow is not shown again upon relaunching of the app. The two screens were created using a Storyboard and a PageViewController that allows a user to swipe left and right from one
screen to the next. While the image above is static, there are animations added to the first screen that demonstrate a sample of the app’s list-like behavior. These animations were achieved using UIKit and manipulating layout constraints to hide/show views at designated times. To achieve the desired layouts, I had to read up on how Apple’s Auto Layout engine works, as well as methods that can be called to redraw layout cycles such as setNeedsLayout() and layoutIfNeeded().

The main application page was by far the busiest page of the application as it holds Betterment’s key functionality. I made use of Apple’s default recommended JTAppleCalendar library to display the calendar views above. This library allowed me to scroll vertically from month to month while allowing the user to select one at a time. Additionally, it allowed me to fully customize how I wanted my calendar to appear and what behavior would occur if a user taps on a cell. I went through the JTAppleCalendar tutorial at https://patchthecode.com/jtapplecalendar-home/ to familiarize myself with the basics of the library, and then set out implement the calendar view above.

The bottom third of the screen holds a container view displaying a list of habits. This container view was created using a UITableView, which each cell in the TableView holding a habit and its selection state. Upon pressing a habit cell, the cell will light up with the corresponding color and increment the “days completed” counter; reselecting the cell will uncheck the habit and decrement the counter to its original value. If a habit for a specific date is selected, then the
A user can create a habit by pressing the plus button on the bottom right, and they can edit/rearrange the order of habits by pressing the bottom left button.

This popup is presented when the user selects the plus button from the main calendar page. Doing so will allow a user to enter a new habit and the color they wish to associate with it while, importantly, not removing the background view from the navigation stack. This is evident by the calendar being dimmed and vaguely visible; pressing the “CANCEL” button will bring the main calendar view back into focus and normal activity can be resumed. The colors selection menu was created with a stack view with individual “color” cells that are outlined.
upon selection. Upon pressing “SAVE”, the Realm Database will perform a write that saves the name of the new habit and is associated color.

Figure 5: Editing and rearranging habits

The screen above is displayed and allows users to edit habit names, color, or even delete them from the list entirely. It also makes use of a draggable, reordering interface that can dynamically update the order in which the habits are displayed in the main screen’s container view. Clicking the list button on the bottom left will dismiss the screen and bring the user back to the main calendar page, while clicking the pencil icon on any cell will allow the user to edit the corresponding habit. Editing a habit will trigger the appropriate updates to the Realm database, which makes use of asynchronous table and view reloading to ensure that the views update accordingly upon changes in the database.
Realm, Database

I decided to use a popular third-party library called Realm (recently acquired by MongoDB) to support my database and model objects. The rationale behind this decision was because of Realm’s light-weight setup, ease of use across all mobile platforms (iOS, Android, watchOS, tablets), and its faster performance. I had previously considered using Core Data as it is gold standard recommendation across the iOS development community. However, having previously dabbled in Core Data with practice projects throughout the term, I wanted to familiarize myself with other technologies, specifically pertaining to database mechanisms.

Creating objects in Realm is incredibly simple, and allows for fast data transactions and queries. Realm also supports data persistence, thread-safe transactions and reactive architecture. As with typical databases, I am able to create one-to-many relationships between objects in my realm DB, or many-to-one or many-to-many relationships. I am also able to perform specific queries using predicates.

My model objects ended up being quite simple in that the main object was a habit with a unique ID, name, color, order, and a counter of days completed. All of these properties were created using basic data types like strings or integers, and I make sure to call the appropriate functions in the file Database.swift whenever I need to perform an update or query.

Testing

While testing was initially a goal I sought to complete for this project, I ran out of time and struggled to come up with effective tests for the objects and views I had created. With test driven development being a new paradigm for me, I made sure to familiarize myself with best practices regarding iOS application testing. This meant learning about XCTest and the associated method calls, how to add unit and UI test targets to an existing XCode project, and how to manually and automatically run each test upon a project’s build. I largely made use of this tutorial (https://www.raywenderlich.com/709-ios-unit-testing-and-ui-testing-tutorial) to familiarize myself with testing in iOS, but simply ran out of time to implement tests unique to Betterment.
Discussion

With the ultimate goal of my senior project being to learn about mobile app development in iOS and the associated best practices, I felt very fulfilled by the final product and how much of a comprehensive survey I was able to conduct over the course of a semester. I very much felt like I picked up various skillsets pertaining to software engineering, object oriented design principles, UI/UX principles, and mobile application architecture. This, coupled with learning a new programming language in Swift, a new IDE in XCode, and a new programming framework in iOS was no small feat, and I owe my newfound learnings to the plentiful documentation online and mentorship I received from peers.

A great pain point I encountered throughout this semester was how tedious and painstaking it can be to create clean, elegant user experiences and views that behave appropriately upon different variables. From different device sizes, different screen orientations, and even accidental user gestures, it was a cumbersome process to debug all possible edge cases and resolve these one-by-one. I felt like I covered most of my bases in the time I had; however, with the app’s limited functionality, I can only imagine the countless hours required to address new bugs introduced by new feature sets.

Future Work

As it stands, Betterment is a completely functional albeit minimalistic application with limited feature sets. A key feature I had hoped to implement but did not have enough time to refine was tracking and displaying habit streaks. For this, I ran into the challenge of how to efficiently keep track of the longest subsequence of dates up to the present, while being sure to minimize space and time restrictions. I was unable to find a satisfactory solution in time for the project’s submission, though I aim to explore approaches made by those who developed similar applications to find potential workarounds.

I had initially set out to create a more social app by integrating Betterment with third-party SDKs such as Facebook and Google. However, I felt hamstrung in the ideation phase of how to effectively integrate these features without cluttering the app’s core functionality. As the non-social elements are more fleshed out, I am eager to revisit this topic with new perspectives on how to connect users and provide social authentication.

Overall, I was quite pleased with the final product in terms of its design and polish. Since I had initially stated that my goal was to create a clean, well-designed application, I believe I successfully accomplished these objectives despite a rather simplistic app. The animations are smooth, the feature set is clear, and there is no ambiguity in regards to any of the design decisions made. Thus, as an aspiring mobile developer, I feel that I have grown tremendously in terms of programmatic and design thinking ability.

Writing effective integration, UI, and unit tests was something I also ran out of time to achieve, as I mostly struggled with coming up with what tests would be informative and beneficial to the project. As I am still a novice when it comes to test-driven development, I aim to further refine these skills and revisit this issue in the future.
Finally, as I aim to submit to application the App Store for approval, I had hoped to explore XCode’s instrumentation tools to analyze app performance and address potential bugs that could hinder a clean user experience. Since my focus was on delivering a minimum viable product, I did not have much bandwidth to make optimization relating to animations, caching, etc.

Acknowledgements

I would like to thank my advisor, Professor Ruzica Piskac, for advice throughout the term and for pushing me to my intellectual limits. I also would like to especially thank Sophie Novati for her guidance and mentorship throughout the term in regards to iOS development.

I would also like to express my sincerest gratitude and appreciation to Angela Yu for her incredible Udemy, Brian Vuong for his fantastic YouTube demonstrations on iOS concepts, and the greater RayWenderlich.com community for their informative articles and walkthroughs.

References and Tutorials

Swift Programming Language:
- Official Swift Documentation
- WeHeartSwift Tutorials

General iOS:
- The Complete iOS App Development Bootcamp from Udemy
- RayWenderlich.com tutorials and forums
- Let’s Build That App YouTube channel

Design:
- Apple iOS Human Interface Guidelines
- Material Design Documentations

Realm:
- Official Realm Documentation
- RayWenderlich.com Realm tutorial

JTAppleCalendar:
- Official GitHub repo and documentation

Testing:
- RayWenderlich iOS Unit Testing and UI Testing Tutorial