1 Overview

I found this project after being approached by the Spring Fling committee looking for someone to build an app. I thought this would be a great opportunity to learn something applicable while also contributing to the larger Yale community.

Starting off this project I wasn’t sure what to expect. I didn’t have any experience creating a project which combined front-end design, back-end logic, and had a database to connect the two. Furthermore, I had no background in mobile app development. Thus, I was excited and optimistic to start learning from scratch.

After a few weeks or so of looking at examples, reading documentation, and getting an idea of a basic application structure, I started to work on basic views, view controllers, and hard-coded models. When the app started to look good with and was navigable with data, I began to think about how to incorporate a database, which would serve data to the app - this seemed to be the crucial step in making it a useful application. The last step was figuring out how to write to the database directly from the app, so the Spring Fling committee (the people responsible for providing data to the app) wouldn’t have to deal directly with the database.

The current version of the app has five tabs along with an admin login which upon success enables the tabs to be editable. The tabs are:

- FAQ
- Announcements
- Schedule
- Merchandise
- Resources

2 Technical Contributions

I built this app in collaboration with Alexis Dornan, so we split the technical responsibilities.

**Language:**
I chose wrote the app in Swift because of the ease of creating views with storyboards in Xcode combined with the detailed Apple documentation plethora of example apps developed in swift which I found online. In hindsight it may have been more beneficial to learn a more flexible language such as React, but I am happy with the progress I’ve made.

**Database:**
Along the same lines I chose to use Google’s Firebase Database because of detailed documentation and examples. This is a non-relation database and so I designed it where data for each tab would be housed in a different bucket, and then added another layer within which would house the tab specific models. In this manner it was nicely organized to test and debug the various tabs.
Specifics:
I was responsible for creating the FAQs, Announcements, and Resources tabs and the authentication functionality.

Main:

FAQs:

- Q.A.swift
  - Model for each question/answer combination. It holds the question and answer strings, the database name location, timestamp it was added, and if the answer is showing
• **QuestionCell.swift & AnswerCell.swift**
  - The are the custom TableCell classes. They are responsible for displaying the text and the QuestionCell displays the carrot in the manner the view controller says to. In admin mode the QuestionCell class also has functions for determining if the ”Edit” or ”Delete” button were clicked.

• **FAQViewController.swift**
  - Initializes a table view and class variable which will hold an array of Q_A objects. Then it fetch the Q_A models from the database, create objects and adds them to the array, and then uses the class variable to populate a table view with cells. It also utilizes the DidSelect function to determine what cell was clicked, and if the corresponding object in the array was closed, it reveals the answer, and vice versa. There is a segue between this controller and the admin version because before it does anything it authenticates, so if someone is login in it transitions to the admin version.

• **AdminFAQViewController.swift**
  - Populates the table and deals with selecting a row in the same manner as in the previous controller.
  - Now though, it has functions for adding, editing, and deleting. If the + button is tapped there will be a segue to the ContentFAQViewController, if edit or delete is tapped then question cell will call a function in this class which is responsible for handling. The edit function will also segue to the ContentFAQViewController and it will pass in the Q_A object that should be edited. After all these functions AdminFAQViewController.swift will re-fetch content from the database and repopulate the TableView. Additionally there is a segue back to the non-admin version because if someone signs-out then we need to transition back.

• **ContentFAQViewController.swift**
  - This view controller has a class variable which represents the Q_A object being passed in from the AdminFAQViewController. If it is in add mode this will be nil, and in edit mode is will be an object.
  - The associate view has different TextViews which will be pre-populated (in edit mode) or empty (in add mode). It also has a submit button which after being tapped will write to the database.

**Announcements:**

![Diagram of app flow](image)
This tab is structured very similar to the FAQ tab. It is actually a simpler version, because there is only 1 row per section (as opposed to the 2 in the FAQ for question and answer cells). There is the similar structure of how the admin features are incorporated - a separate view which is transitioned to if the user if authenticated, and also the same structure for a ContentView which writes to the database.

**Resources:**

- **Resource.swift**
  - This is the model for this tab. It stores a name of the resource, phone number, link, and picture. All resources have a name but only one of the other 3 possibilities (they are all stored as optionals).

- **ResourceCell.swift**
  - This is a simple class responsible for altering a label to store the name of the resource.

- **ResourceViewController.swift**
  - This class has a table view and array of resource objects. After speaking with the Spring Fling committee, these resources will stay constant so I hard-coded the values into different Resource objects. When one of these cells gets selected depending on if it is a phone number, link, or picture the view controller preforms different operations. For phone numbers and links the UIApplication.open() function appropriately either provides the option to call or redirects to the link. If it is a picture there is a transition to ResourceImageViewController and appropriately passes the resource that was selected.
  - A special resource either "Admin Login" or "Sign Out" (depending if the user is authenticated) also gets appended to the end of the array. Upon selecting the "Admin Login" resource there is a segue to the LoginController which will allow an admin to be authenticated. If the user is already authenticated then the "Sign Out" resource will appear which will allow them to sign out.

- **ResourceImageViewController.swift**
  - This is a simple controller which has a label and an ImageView. It is embedded within the navigation controller so after a picture is selected this controller loads the image from the resource passed in and changes the label to reflect the resource name. Because it is in the navigation controller there is a back button to go back to the ResourceViewController.
This is the LoginController. It utilizes the Firebase Authentication pod to determine an unsuccessful attempt, in which a label gets populated with the error, or determines a successful attempt. Upon success the user is redirected back to the ResourceViewController, and now instead of an "Admin Login" resource a "Sign Out" one will appear. At this point navigating to the FAQ or Announcement tabs will reveal the special features to add, edit, or delete.

3 Learning Process

Over the course of the semester I have come a long way. The initial stages of setting up the Xcode environment and learning the basics were intimidating and definitely the hardest part.

Once I understood the MVC paradigm and had a good grasp on how ViewControllers and TableViews were supposed to function I was able to make quick progress. I utilized TableViews in all my main ViewControllers and the setup was the same, so this allowed me to use one working tab as a skeleton for the others. The resources online were awesome for this, especially a YouTube account called Let’s Build That App.

After populating the tables with hard-coded data, I needed to figure out how to incorporate a database, and Firebase made this pretty simple. Having never worked with a database it was nice introduction because the code to read and write data were well documented. It was exciting to get this functionality working because I could edit the contents of the app without changing any source code.

The last step was creating a user-friendly way to edit the database rather than logging into the Firebase console from a desktop. The login controller took a long time to figure out, but once this was working I was able to make the app look different if the user was authenticated (by redirecting the administrator to the special view controllers). Now there was a way to provide special functionality for writing to the database, and in turn an admin could make edits directly on the app in which all people who downloaded the app would be able to see.
4 Future Plans

The Spring Fling committee was thrilled with the current state of the app, but there are still more features I want to implement before it goes on the App Store.

The most pressing issue is testing. While building it I often was debugging, but never wrote unit tests and if this is going to be used in practice I want to be sure it won’t crash.

Another feature that would be beneficial would be push notifications. I was hesitant to implement these because I didn’t know what the Spring Fling committee’s plans were (i.e. how, when, and what they wanted to send out). After I speak with them I will have a better idea of how to incorporate them in the app.

Finally, I need to learn how to deploy this app to the App Store. I’ve been waiting to create an apple developer account, but once this happens I’m optimistic this will be a simple process.

5 Reflection

Looking back I realized that it was cool learning how to develop an app, but more importantly I was able to learn how to build a large project from scratch. This project gave me the confidence that it is OK to struggle in the beginning and moreover, that I could learn how to create something concrete.