Portable Classes: Documentation

David Krauskopf-Greene & Anthony Ramirez

Overview of App:

Portable Classes is an app intended for high school and college students who wish to keep all their information pertaining to their classes and be able to share it with others in one easy to use, convenient app. Users can organize their content by semesters, then by classes, then by class features like deadlines, notes, pictures, and flash cards. They can add and remove as much content as they would like, and the best part is they can share it with one another. There is a map with every public user, and each one's content is accessible to everyone else.

Technical Aspects:

Logic/How to use app:

When the user first creates an account in the register view controller, we use Firebase's authentication feature to create a user with an email and password. To validate users, the email must end in "@nyu.edu," and the password must be at least 8 characters and contain a lowercase letter, capital letter, number, and special character. Duplicate accounts are not allowed. We also allow the user to be public or private for showing them on the map. Once the user is logged in, the main screen appears, which includes buttons that link to their classes and to a map of other public users. The map view controller has a map that focuses on the logged in user's location and displays all public users. Underneath the map is a table of all these public users; tapping on a cell takes the logged in user to the semesters view controller. Back on the overview page, tapping "View Your Classes" also takes you to the semesters view controller, but the content is the logged in user's, whereas if the segue comes from the map view controller, the content is the other user's. Within the semesters view controller, a logged in user can add or remove a semester, then can create courses. To remove a cell, the user can slide the cell to the left or tap the edit button and then the delete button in the cell. A sound plays when the semester is stored in the database. Once a semester cell is tapped, it takes the user to the courses page, which has the same functionality as the semesters page. Once a newly created course cell is tapped, its class features (deadlines, notes, pictures, and flash cards) show up in a table with clickable cells. For deadlines, a user can create a new one by tapping the add button; before adding, the user has the option to add the deadline to the phone's calendar. For notes, the user can add/delete notes, and then by tapping on a newly created note cell, view the note. Upon viewing a note, the user can then edit it if desired. When tapping on pictures, the user can take a photo or import one from the phone's camera roll. Upon tapping on a picture cell, a new view controller is presented

with the image at a larger size, and the user has the ability to delete it by clicking on the trash button. Lastly, for flash cards, the user can add a collection (such as chapter 1 vocabulary). Within each collection, there is a horizontal scroll view with page control for each term/definition. Upon tapping the add button, a view controller with a text field for the term and a text view for the definition appears; these cannot be empty or duplicates. For each term/definition pair, there is a "Tap to toggle button" where the page rotates 180º to switch between the term and definition. The more pairs added, the more page control dots appear, and the user can swipe to view them all.

iOS Features:

The navigation bar on each page is consistent throughout the app. We use a navigation controller for most of the view controllers, so there is a back button. The title of the page is set to large and the tint color is set globally to yellow, while the background color/tint color is the same for every table and cell . For adding a deadline, flashcard term/definition, and note, however, different navigation is used–these view controllers are presented modally, so there is no back button. Instead, there is an X to dismiss the view.

Most of the content is displayed in tables, so we use several UITableViewControllers. With these, we include editing functionality. For each cell, there is a disclosure indicator to indicate the cells are meant to be tapped, and we clear the selection color upon returning to that table. Additionally, anytime a user tries to add a cell, a UIAlertController pops up and asks the user to enter a string. We do not allow duplicates or empty strings.

For displaying the map and users, we use Apple's MapKit and CoreLocation. MapKit shows the map, while CoreLocation makes it possible to access users' current geolocations. Until the user opens the map, the default location is set to coordinates [0º, 0º].

For adding a deadline, the date's text field keyboard is a UIDatePicker, which allows the user to enter a time and date. For adding the deadline to the phone's calendar, we use EventKit, which allows the user to choose a time for the event, its duration, and notes about the event–but we set the start and end time to be the same and do not allow notes to be added to the event.

For adding pictures, we use a UIImagePickerController, which allows the user to either take a picture or select one from the camera roll. We then allow the user to click on any individual image to take them to a view of the full image. Here, they can zoom in/out and scroll the image in the case that all the content is not viewable from the initial full screen view. We do this by adding pinch gestures for zooming and swipe gestures for

scrolling. Lastly, we use CGAffineTransform to scale the view and CGRect to change the frame when there is a swipe.

For adding/viewing notes and adding definitions for flash cards, the text appears in a UITextView.

For creating the flash cards, we use a horizontal paging UIScrollView with UIPageControl. An individual flash card is made from a xib file, which makes it possible to design the layout of the page and essentially embed it in the horizontal scroll view controller. Upon adding a new flash card, UIPageControl adds dots at the bottom of the page to indicate however many number of flash cards there are.

To avoid the plain, boring System font, we have always known we wanted to use Avenir. It is a minimal yet fun and professional font, and it suits our app very well.

Lastly, anytime a user successfully adds a document (such as a semester, course, deadline, etc.) or toggles between public and private, there is an mp3 file that plays, whereas a wav file plays when a user deletes content. This is done using AVFoundation.

Additional APIs:

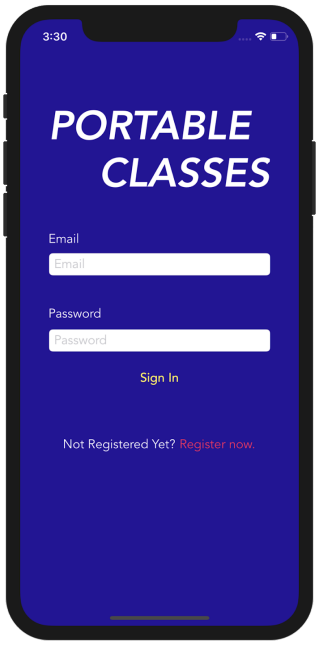
For storage, we use Firebase Cloud Firestore, which is a NoSQL cloud database. It is better than Firebase's realtime database because the GUI aspect makes it easier to view the data. Each user is a document, and each document contains a semester collection. Each semester collection contains a course document. Each course document contains class feature collections (deadlines, notes, pictures, and flash cards). Each class feature collection contains deadline documents, notes documents, downloadable image url documents, and flash card documents (which contain collections, such as chapter 1 vocab, and within each collection, there is an array of term/definition documents). To be able to get any of Firebase's features to work, we have to install the Firebase's Core, FirebaseUI, Auth, Database, Firestore, and Storage pods.

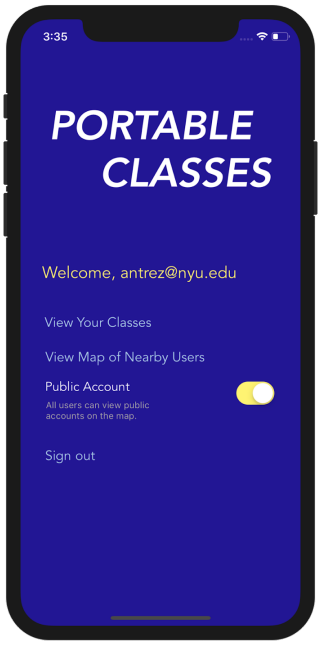
Clever Code:

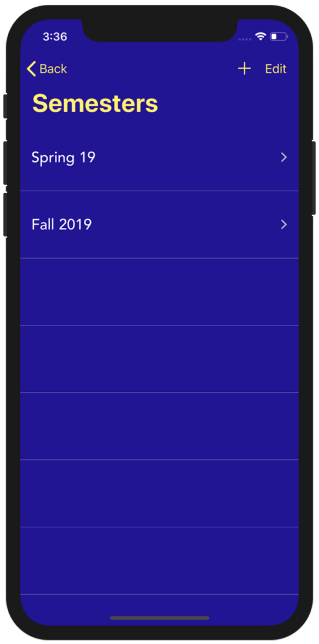
Lines 90-136 in our CameraViewController.swift script make up our most clever code. This is the doneView func which adds a new image to our collection view of images. It is clever because it compresses the image (for faster load time). It then adds the image to Firebase Storage and to the current user's folder of images. Then, the path of this image is converted into a downloadable url and passed as a string to the our database. This is where we add the image string to our pictures array and pass it back to PicsCollectionViewController.swift through a callback method.

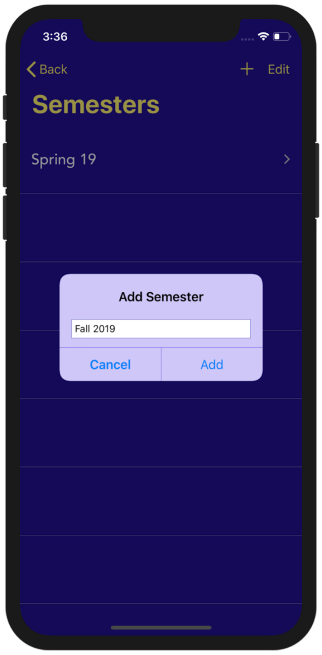
```swift
@IBAction func doneView(_ sender: Any) {
    // if an image has been added
    if myImg.image != nil {
        // play the success sound when adding an image
        let path = Bundle.main.path(forResource: "add", ofType:"mp3")!
        let url = URL(fileURLWithPath: path)
        do {
            audioPlayer = try AVAudioPlayer(contentsOf: url)
            audioPlayer.play()
        } catch {

        }
        // compress the image for faster load time
        var data = Data()
        data = myImg.image!.jpegData(compressionQuality: 0.3)!
        // add the image to firebase storage
        let imageRef = Storage.storage().reference().child((Auth.auth().currentUser?.email)! + "/" +  randomString(20))
        // download the storage url
        _ = imageRef.putData(data, metadata: nil) { (metadata, error) in
            imageRef.downloadURL { url, error in
                if error != nil {}
                else {
                    // add the url to the array of images
                    self.allImages.append(url?.absoluteString ?? "")
                    // get reference to firebase
                    let db = Firestore.firestore()
                    // reference to all the users
                    let allUsersRef: CollectionReference? = db.collection("users")
                    // reference to the document of handNotes
                    let currUserRef: DocumentReference? =
                        allUsersRef?.document((Auth.auth().currentUser?.email)!).collection("semesters").document("semesters").colle
                        ction(self.currSemester).document("classes").collection(self.currClass).document("handNotes")
                    // add the image to the database
                    currUserRef?.setData([
                        "handNotes": FieldValue.arrayUnion([url?.absoluteString ?? ""])
                    ], merge: true) { err in
                        if err != nil {}
                        else {
                            // send the image through the callback
                            self.callback?(self.myImg)
                        }
                    }
                }
            }
        }
    }
    // dismiss the view controller
    self.dismiss(animated: true, completion: nil)
}
```

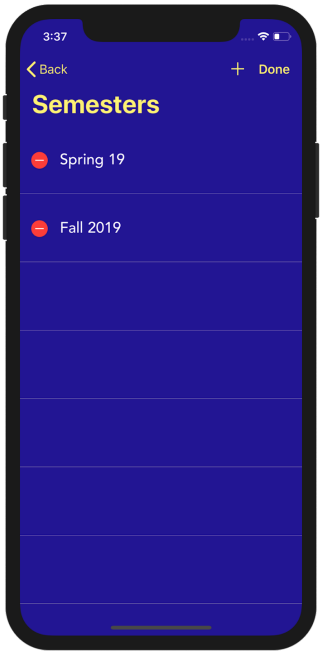
Screenshots:

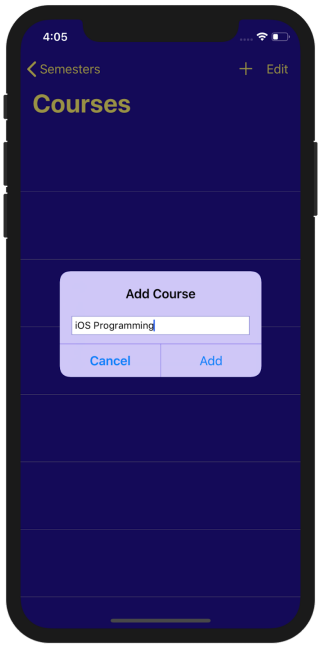


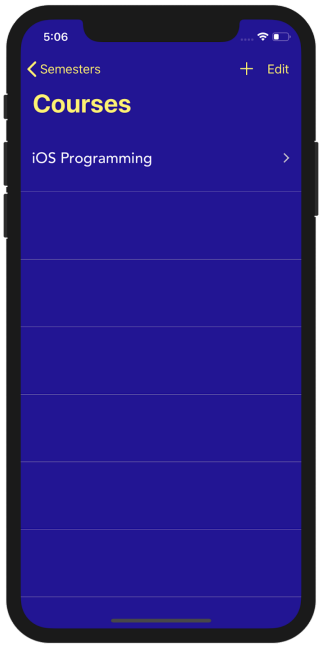


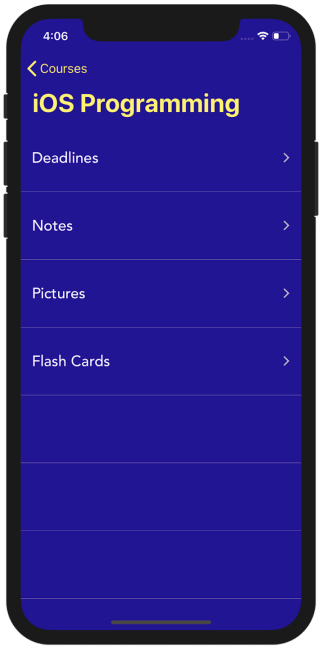

3:36
Back
Edit
Semesters
Spring 19
Fall 2019

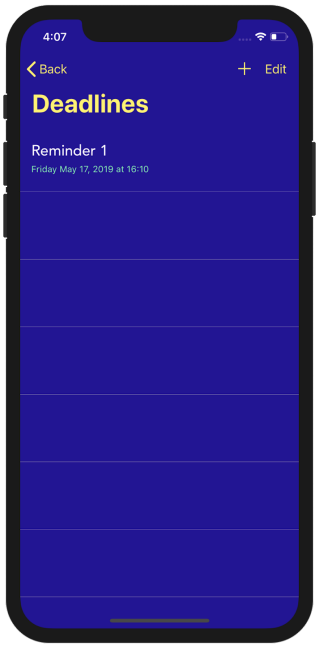

3:36
Back
Edit
Semesters
Spring 19
Add Semester
Fall 2019
Cancel
Add

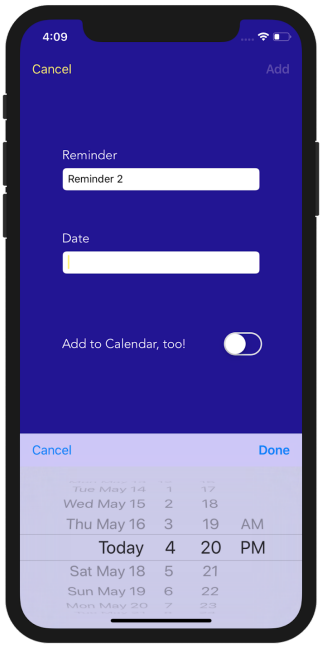

3:37
Back
Done
Semesters
Spring 19
Fall 2019

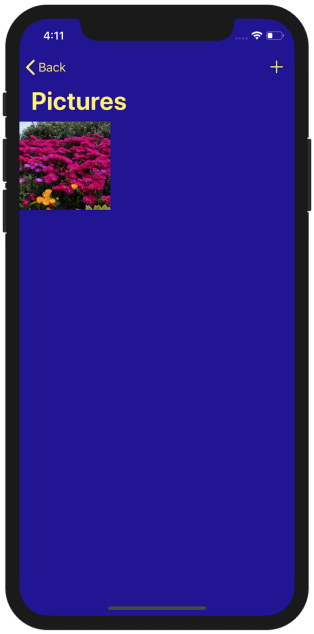

4:05
Semesters
Edit
Courses
Add Course
iOS Programming
Cancel
Add

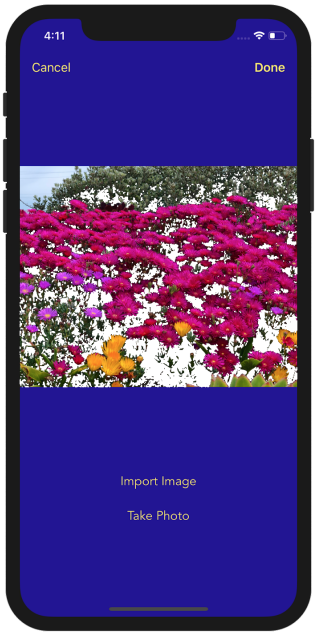

5:06
Semesters
Edit
Courses
iOS Programming

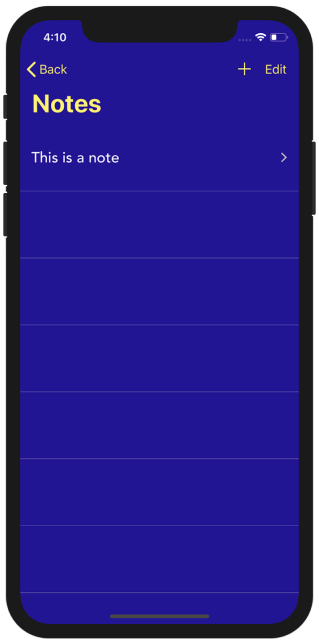

4:06
Courses
iOS Programming
Deadlines
Notes
Pictures
Flash Cards

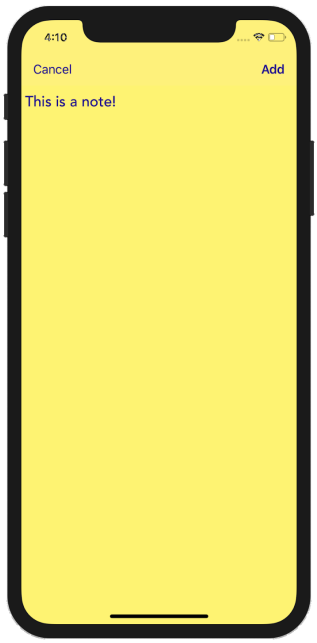

4:07
Back
Edit
Deadlines
Reminder 1
Friday May 17, 2019 at 16:10

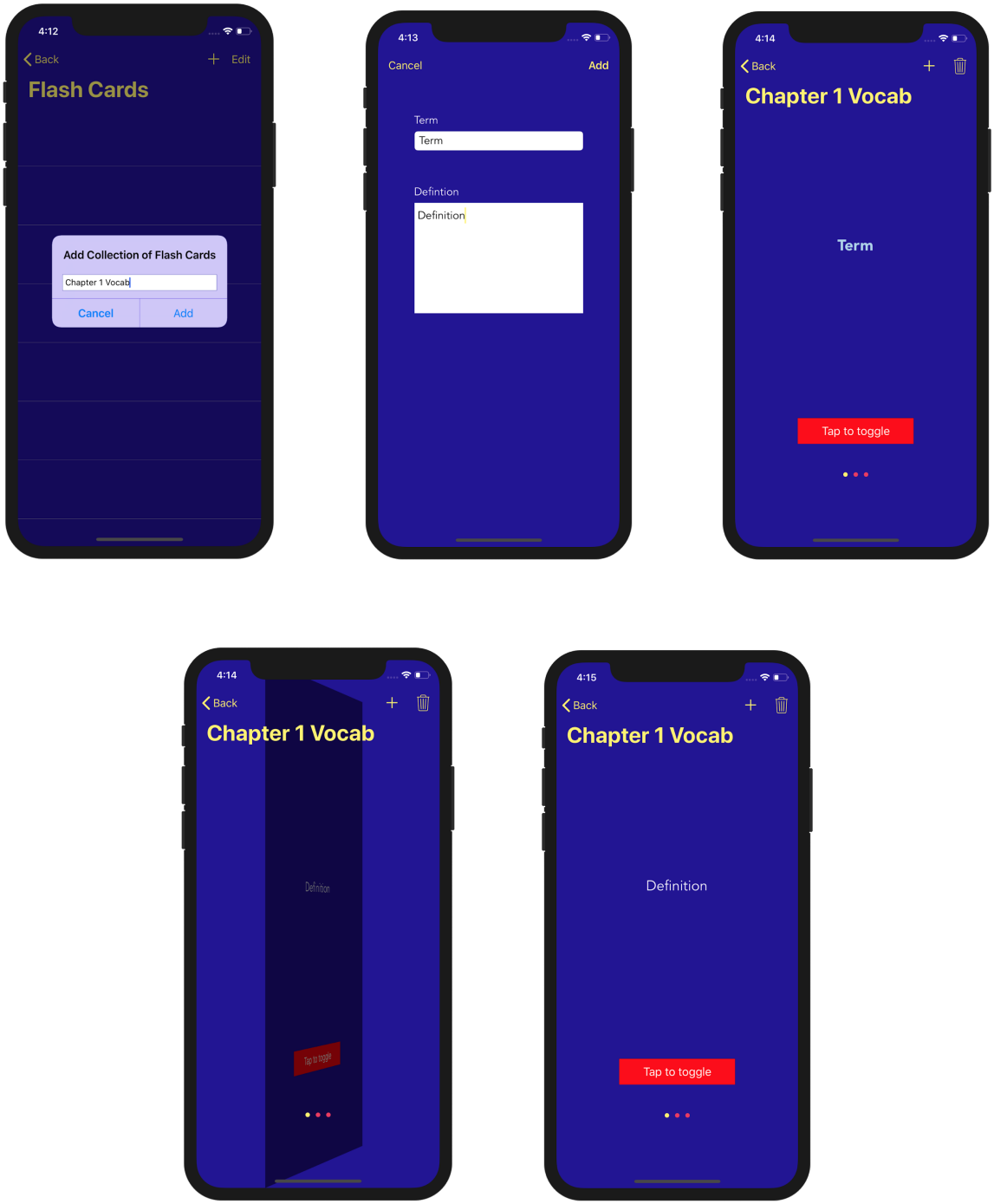

4:09
Cancel
Add
Reminder
Reminder 2
Date
Add to Calendar, too!
Cancel
Done
Wed May 15 2 18
Thu May 16 3 19 AM
Today 4 20 PM
Sat May 18 5 21
Sun May 19 6 22


4:11
Back
Pictures


4:11
Cancel
Done
Import Image
Take Photo


4:10
Back
Edit
Notes
This is a note


4:10
Cancel
Add
This is a note!



Challenges:

Because this was a group project, one of toughest parts was working on the project simultaneously. We used GitHub, and modifying the scripts was fine, but we found that we could

not work on the storyboard at the same time. Thus, most of the time, we had to wait for each other to finish, or we would work together in person.

Another technical challenge was using Swift. We started the project when we barely knew the language, so there was a bit of a learning curve. Additionally, neither one of us has used Firebase in the past, so there was a learning curve to that, as well. For example, we had to learn about snapshots and how to reference/update all the data using Google's methods and syntax. However, with previous knowledge of other backend technologies, we were able to create an effective database design to store all the users and their content nicely.

Furthermore, we struggled to load the images into the view controller after the user adds one. Because the image is first downloaded, it takes awhile for the image to be retrieved and loaded into the view controller. We tried compressing the file size, but there is still a small lag.

Lastly, we struggled to allow the logged in user to view other users' content, but we then realized it was simply passing along the email (either of the logged in user's or one from the map table) in each segue.

Member Roles:

Anthony was in charge of the UI/UX–he made the app logo, and chose the color scheme and placement of all of the UI Objects within the Storyboard. Additionally, he was responsible for creating the flash cards functionality–like adding them to the database and creating PageControl for all the term/definition pairs that toggle on being tapped– as well as the deadlines functionality and adding to the phone's calendar.

David took control of the login/register functionality using Firebase's authentication system, as well as the functionality for making/editing notes, displaying the map, storing/displaying images, and setting constraints on each view controller. He used MapKit to display the map and storing the users' geolocations into the database, while Anthony worked on reading the users' geolocations and annotating the map with them. Furthermore, Anthony worked on getting public users' content to show up when segueing from the map view controller. For the images, David was responsible for figuring out how to download the images to Firebase and then getting them to display once they are stored.

We split the work on the functionality on all the tables, such as adding and deleting cells to the table and updating the newly created/deleted content in the database.

Future Steps:

Now that we have the social aspect of the app, which we did not include in our MVP, we would like to add a feature to allow users to be able to add other users' content into their own pages. For example, a user can add a collection of flash cards from another user. We could even incorporate a messaging feature and the ability to add friends. Down the line, we would also like

to get in touch with schools and pitch our app to them. This way, we could truly authenticate user emails, and we could even create an aspect intended for teachers/professors to post material to share with students.

Other small features we would like to implement but did not have time to do include allowing the user to create a username and set a profile picture. This way, on the table of other users under the map, students can recognize each other more easily. Lastly, with a paid developer license from Apple, we would like to incorporate push notifications to remind the user when their deadlines are approaching.