Robert Vo
Mobile Programming for iOS Development - 590
Final Project
7/19/17

Final Project

Objective

The project I chose to create using Swift was a Dog Journal application. The function of the application can be used to keep track of a pet dog's personal records or information. The application can also be expanded to other pets such as cats, horses, or any other animal that are common pets. However, for simplistic reasons, I decided to focus on dogs.

The application contains three main features: journal, map, and camera/photo section. The purpose of the journal is similar to a journal used by people, to record any significant events that happened throughout the day. The map is used to track runs or walks that are taken by the dog and owner, and the camera/photo section can be used capture moments.

The application, having only a few weeks to work on it, is in its early stages and can be altered as the full application progresses. Throughout the report, I will describe its current state and what can be added to improve the application.

Methods

The first part of the application created was the home screen, which is the journal of the application. The journal is created with the use of a navigation controller. The navigation controller controls two views for the journal, the table view and the new journal entry view. Figure 1A shows the blank table view at startup and Figure 1B shows the result of clicking the "+" icon on the top right of the nav bar to switch to a new journal entry view.

Figure 1A

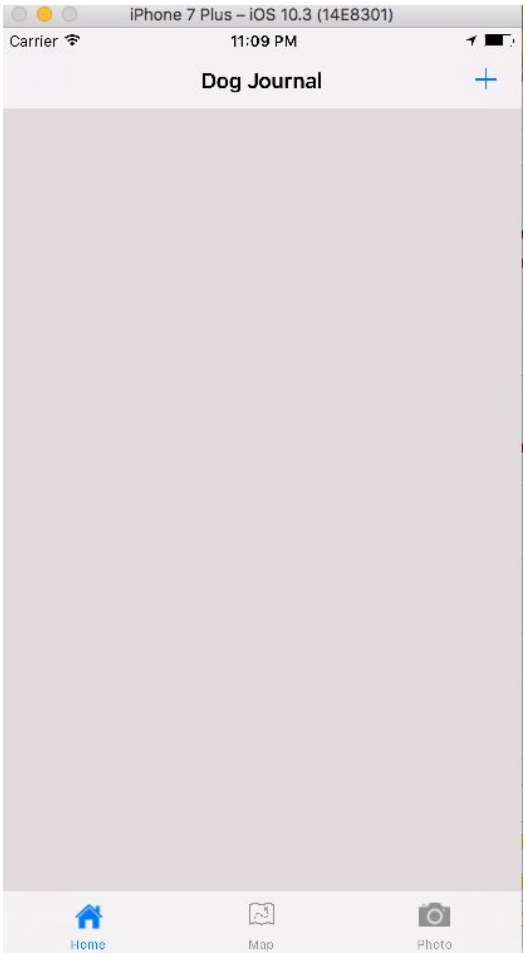
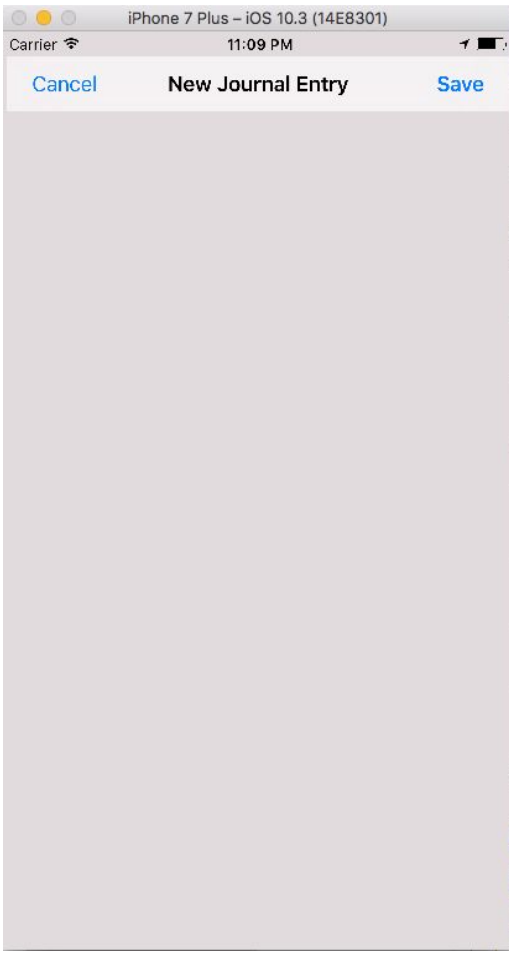


Figure 1B



The following snapshots show the random text entered as a journal entry. The journal entries are then saved and the timestamp shows when they were created. Figure 2A shows the new journal entry with the text and Figure 2B shows the table with four created entries.

Figure 2A

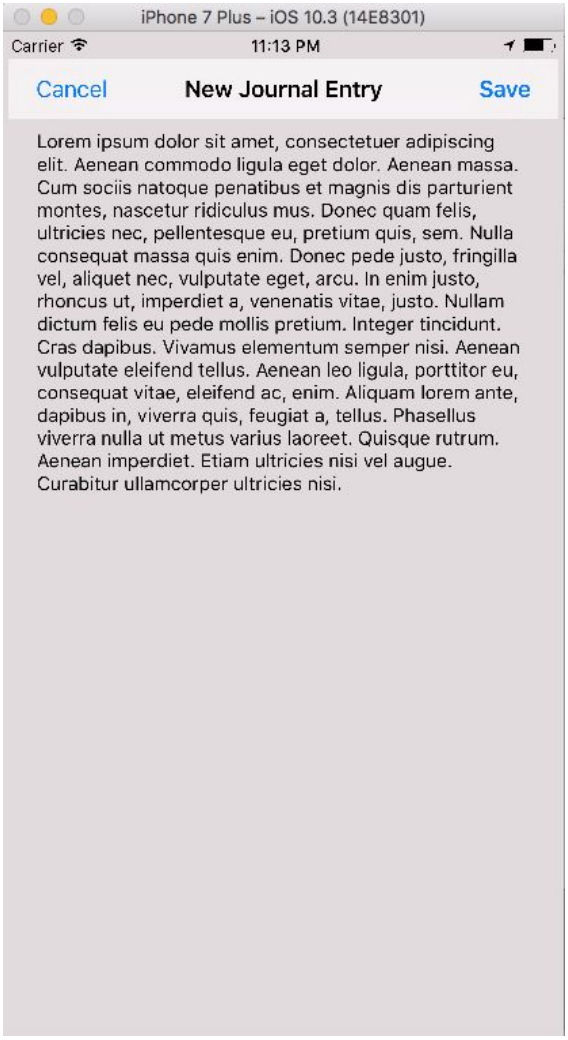


Figure 2B

A required feature that needs to be added to the application is a way to save the data or journal entries. Therefore, closing and reopening the application, journal entries from past months can be viewed. Features that can be added to enhance user function are definitely cosmetic features or animations. However, one feature that can be added is an image view so that an image can be attached to the journal entry.

From the previous snapshots, the bottom of the application shows the tab bar that allows the user to switch between the various states of the application. The initial view is also the home screen which is the journal part of the application.

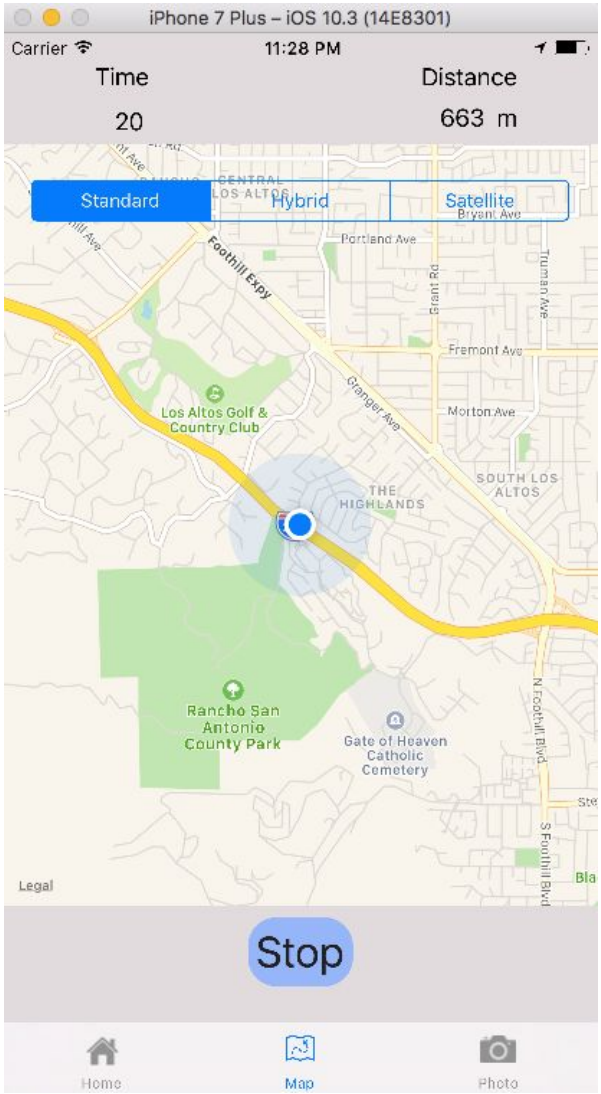
The second part of the application is the map section. The map is used as a gps tracker for walks or runs. Clicking on the map shows the map in the middle of the screen with the segmented control to switch from hybrid, standard, or satellite view, a start button at the bottom, and two labels at the top to

show the time of the tracking and the distance covered in meters. Figure 3 shows the map view in its entirety.

Figure 3

Clicking the Start button activates the timer and the location tracker. Figure 4 shows the result of clicking the start button and what it shows after a few seconds of running. The figure also shows the Start button switching to Stop in order to stop the timer.

Figure 4



The one feature I would like to add to the map is a line that traces the route of the path taken. I tried to implement the function for the mapView in order to overlay a polyline, but could not get it to add on for some reason. The function shown in Figure 5 is the current function. It does not draw any errors, but also does not add the line to the route. I believe that there is an update issue since the only examples that I could find were for previous iOS versions and was different syntax.

Figure 5

```
func mapView(_ mapView: MKMapView!, rendererFor overlay: MKOverlay!) -> MKOverlayRenderer! {

    if overlay is MKPolyline {
        var polylineRenderer = MKPolylineRenderer(overlay: overlay)        Variable 'polylineRenderer' was
        polylineRenderer.strokeColor = UIColor.blue
        polylineRenderer.lineWidth = 4
        return polylineRenderer
    }
    return nil
}
```

The final part for the application is the camera feature. The camera allows for users to take photos, or upload photos from the camera roll. This could also be used to import an image to the journal entry as previously discussed. Currently, there is only one image view as shown in Figure 6.

Figure 6

Clicking on the button to import an image from the camera roll takes to the default pictures. I added a photo of my dog in there for the sake of the application. Figure 7A shows the result of clicking “import image” and Figure 7B shows what happens after the selection of the image.

Figure 7A

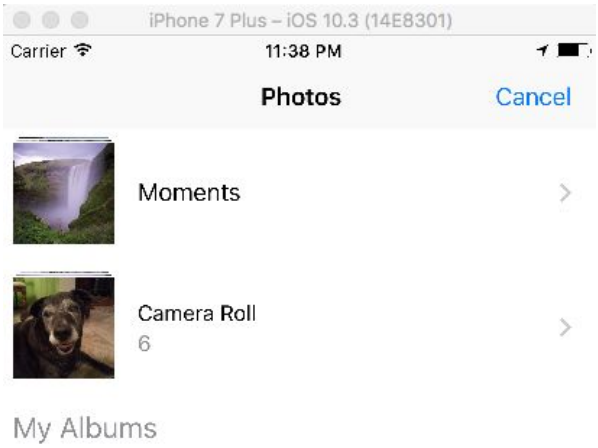
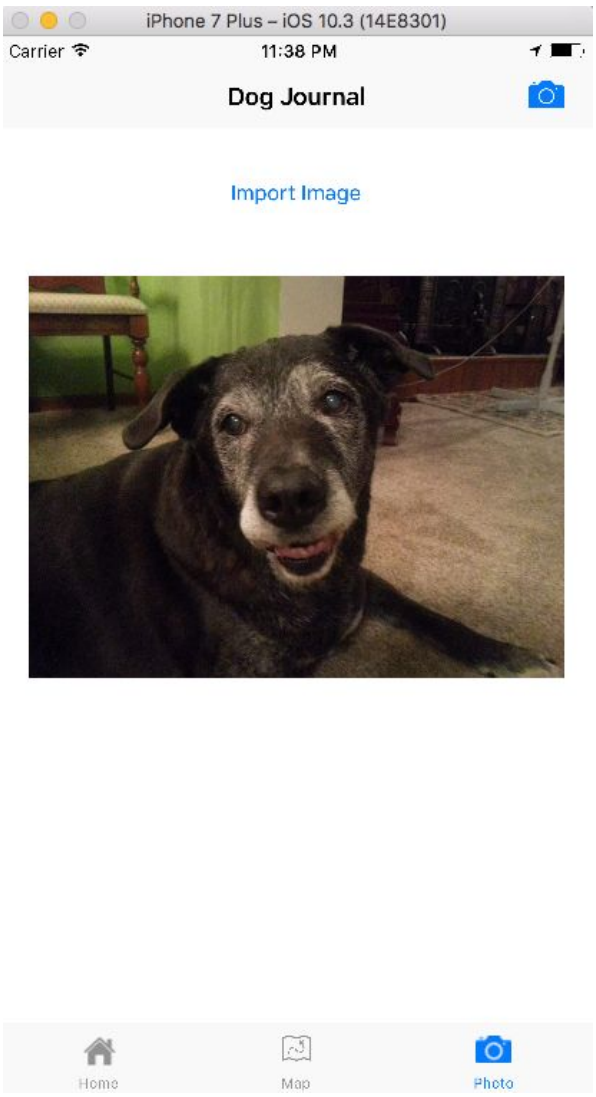


Figure 7B



The photo viewer of the application can be made into an application like Instagram where there are multiple image views for many photos on the screen. Details of the image can also be added such as date or a caption. Figure 7B shows the camera icon at the top right in order to switch from the photo viewer to the actual camera. Clicking the button takes to a blank screen on the simulator as shown in Figure 8A. Figure 8B shows the camera view controller with an image view.

Figure 8A

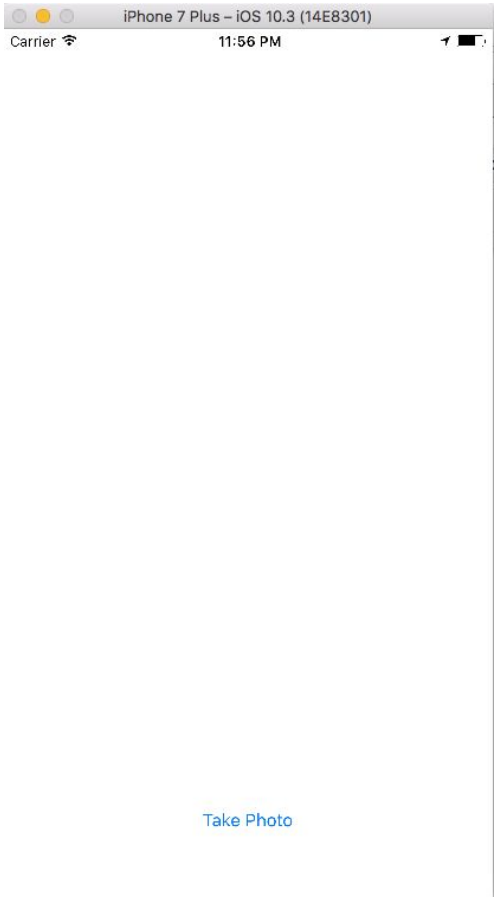


Figure 8B

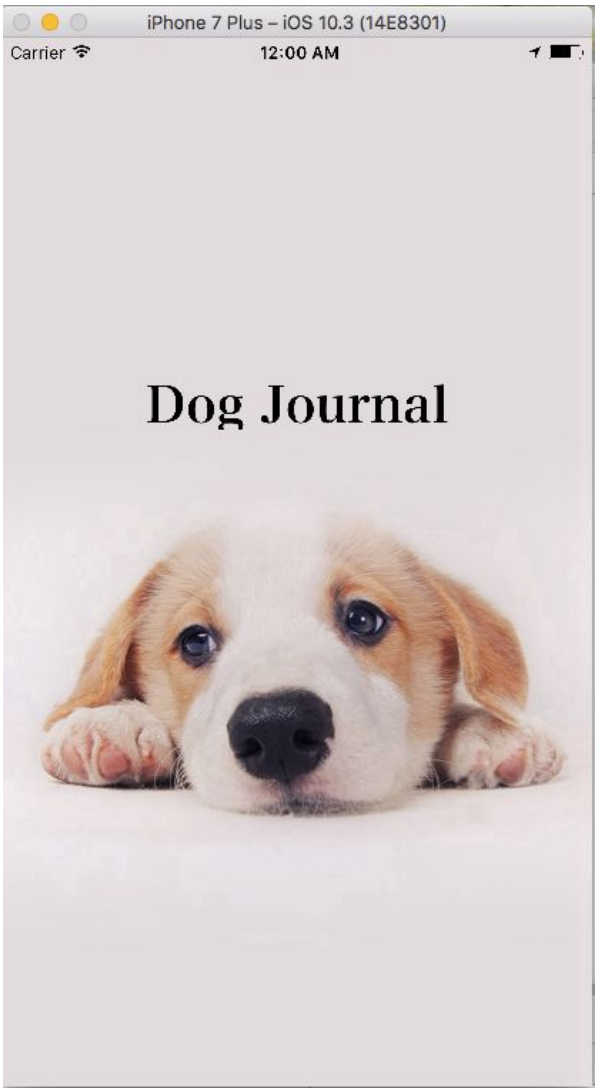
The function for the “Take Photo” button is shown in Figure 9. This main difference between this “Take Photo” button versus the “Import Image” button from Figure 7B is the source from where the photo is taken. Importing the image takes the image from the camera roll, therefore the source is the photo library. The UIImagePickerControllerSourceType for the camera button is “camera.”

Figure 9

```
@IBAction func useCamera(_ sender: Any) {

    if UIImagePickerController.isSourceTypeAvailable(
        UIImagePickerControllerSourceType.camera) {

        let imagePicker = UIImagePickerController()

        imagePicker.delegate = self
        imagePicker.sourceType =
            UIImagePickerControllerSourceType.camera
        imagePicker.mediaTypes = [kUTTypeImage as String]
        imagePicker.allowsEditing = false

        self.present(imagePicker, animated: true,
                     completion: nil)
        newMedia = true
    }
}
```

The final addition for the application was the launch screen. A simple photo of a dog and the title for the application is shown when it loads up as in Figure 10.

Figure 10



Results

The application is in early stages, and plenty of modifications can be added in order to enhance the features. However, the application uses many of the core skills that were learned throughout the course, and further addition will require the use of further reading from Apple's Swift documentation.