CP Maps

By Brian Fung, Carl Lind III

Senior Project

Advisor: Aaron Keen

Spring 2015

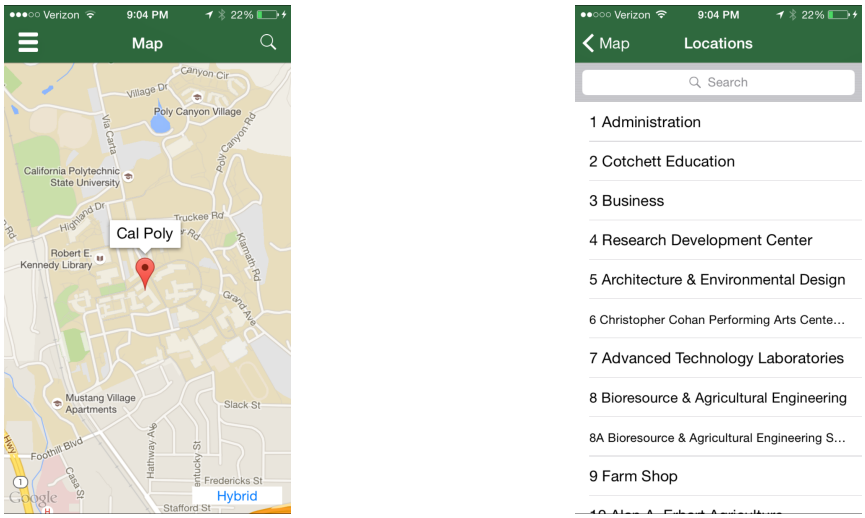
Abstract

According to *Forbes* magazine, Cal Poly ranks No. 9 among California's public universities and is the top-ranked California State University for this past year[7]. More than 55,000 undergraduates seek admission for Fall 2015[11]. These visiting students and their families and as well as 20,000 current students, faculty and staff need an convenient way of navigating around campus[8]. Our iOS app—CP Maps—was built from the ground up to address the difficulty of navigating around campus.

Contents

1 Initial Goals and Vision

When we were discussing what we would improve about our Cal Poly experience with our friends and colleagues, we realized that there was need for an app that would improve our current mobile solution for Cal Poly's campus maps: a solution that would allow students to find locations of buildings and classes, as specific as viewing the exact location of their classroom on a floor plan. Students would also be able to save their classes from the website onto the app and have the app map to the location at that time. All of this combined with the ability to find the most optimized routes from their location to their class, which included shortcuts through buildings. This could also incorporate the recording commonly traversed routes to recommend to all users.



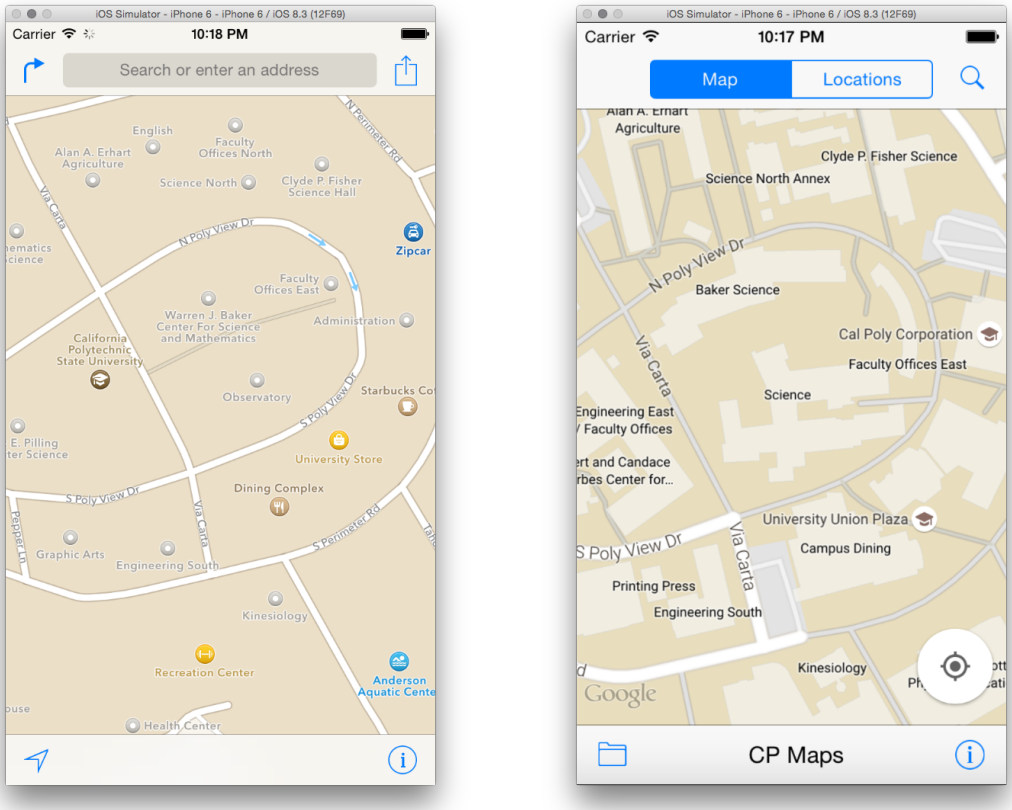
(a) Default Location on start (b) Building selection with search

Figure 1: Current Cal Poly iOS Map

2 Design Details

2.1 Google Maps vs. AppleMapKit

Upon deciding our goals and vision for this project, our next step was agreeing on some design choices. Considering this project is aimed toward mobile, we had to make a choice between iOS and Android development. Both platforms are widely



(a) Apple MapKit (b) Google Maps

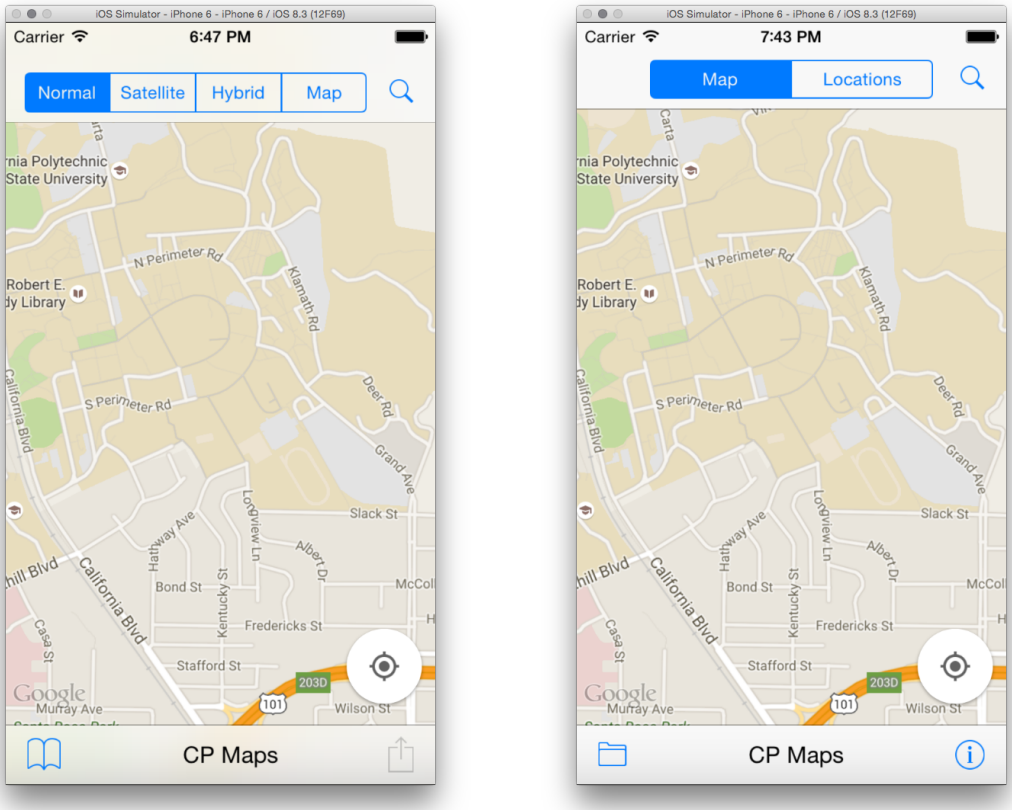
Figure 2: Apple Maps vs. Google Maps

used across our campus as well as across the world. With this in mind we decided to lean toward our personal preference of learning iOS since we both are interested in iOS development.

We decided to use Google Maps over Apple MapKit. The benefits of Apple MapKit is easier implementation and better integration; however there is a large loss of data quality in comparison with Google Maps.

Although Apple Maps contains the necessary road and building locations, there would be missed details with the map and buildings. Based on these components and the versatility, we chose to incorporate Google Maps into our project.

Apple released their new programming language Swift on October 22, 2014. The previous language used for many years was Objective C, and is still supported to-



(a) Old UI (b) Refactored UI

Figure 3: Old Design vs. New Design

day. Ideally we would have liked to to learn both: Objective-C for documentation, support, reliability, and legacy and Swift for state of the art technology that is Apple's replacement for Objective-C. In hopes to stay current with the most recent technology, we decided to use Swift for our project.

At one point toward the end of our project, we decided to restructure our user interface. We saw that our initial design wasn't beneficial for the end user with the placement of UI items. Specifically, the map type was replaced with the map and saved locations toggle. Also our floor plans button was only accessible after searching a building, but the new ui allows for any building to be selected to show the images.

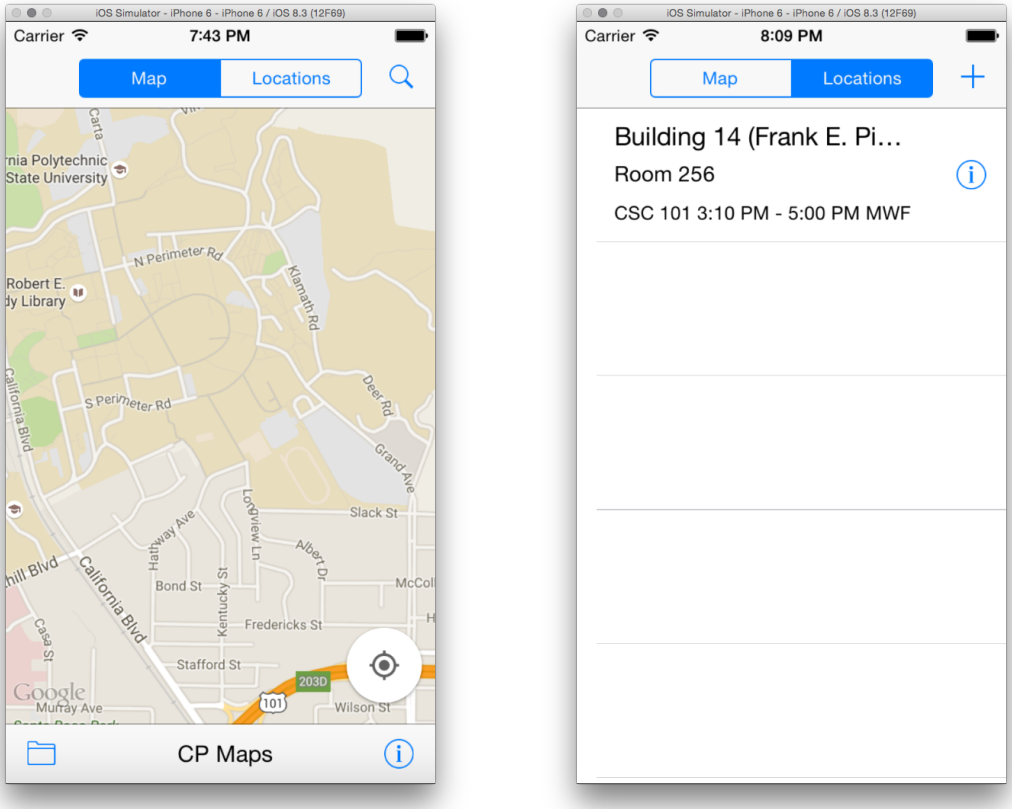
3 Implementation Details

Our app starts with the map displayed. The map and the locations can be using the segmented control at the middle of the navigation bar. The map and the locations view are held within a container view.

3.1 Home Page

3.1.1 Overview

When the app starts, the home page is displayed. At the top of the home screen, a toggle for maps and locations and a search button is displayed.



(a) Viewing Map (b) Viewing Saved Locations

Figure 4: Viewing the Map or Viewing Saved Locations

The map is selected by default. Tapping on the search button brings up a list of buildings for Cal Poly. Selecting a building then brings the map view into focus, drops a pin onto the building and creates a route from the user's location to the building.

If the locations is selected, the plus button replaces the search button. Tapping on the plus button will allow a user to add a new location.

3.1.2 `MainViewController`

The `MainViewController` is a container controller that has a `UINavigationBar` that has `UISegmentedControl` with two segments, Map and Locations, in the its `titleView` and has a `UIView` that displays a `MapViewController` or `LocationsViewController`, depending on which is toggled and inherits `UIViewController`. The bar's right button displays a default search button if map is selected and a default add button if locations is selected. Every time the segments are toggled, the `UIView` loads the correct view controller. The `MapViewController` and `LocationsViewController` are created when the `MainViewController` is initialized and saved as variables. The code to make the container controller is from Yari D'areglia's article "Working with custom container view controllers"[5].

Tapping the search button displays the `ChooseBuildingRoomTableViewController`. Tapping on a building cell will pass the selected building to `MapViewController`'s function `showSelectedBuilding` so that the `MainViewController` will display the location and, from the user's location, route to it.

3.2 Viewing the Map

3.2.1 Overview

As mentioned in the Home View, the Maps page is selected as default on launch of the app. We are using Google Maps SDK to populate the maps view and interact with it. The center location defaults to the UU. If the user allows, the map will display their current location as a blue dot on the map.

Google Maps will display a marker which contains the location info for a selected building and give directions to that location. Long press will create a marker and give directions. A short tap will remove any marker that was placed on the map.

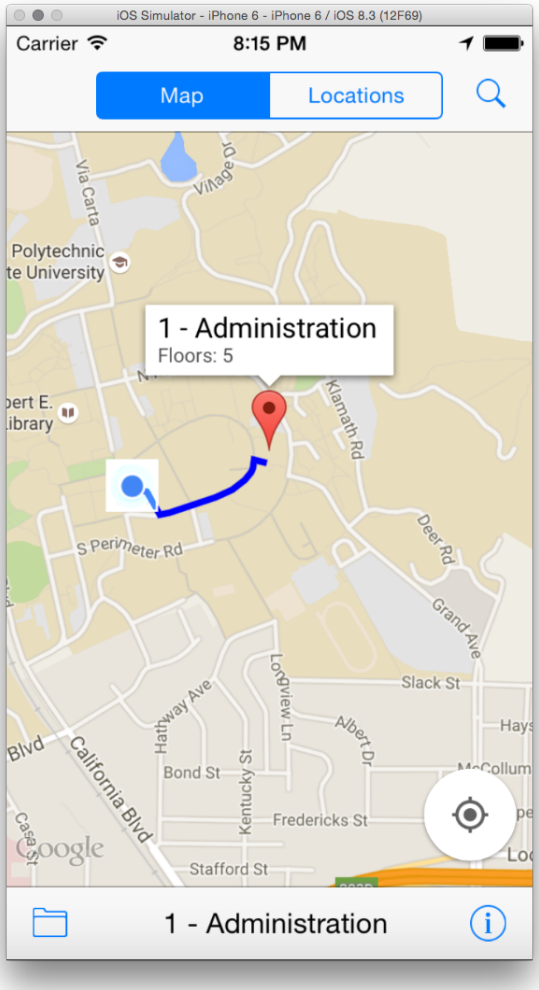


Figure 5: Map marker and directions to building 1

3.2.2 `MapViewController`

The `MapViewController` extends `UIViewController`, `GMSMapViewDelegate`, and `CLLocactionManagerDelegate`. `UIViewController` handles the UI navigation between actual display of Google Maps as well as various buttons and labels. The `GMSMapViewDelegate` is a part of Google Maps SDK which is used for the user's interaction with the map. Lastly the `CLLocationManagerDelegate` is used for retrieving the user's location from the device if they have given permission. We made

sure to handle and test our code for the use case of a user not allowing permission to their location.

The driving object located in this class is the `GMSMapView`. This controls and manages all interaction with the map itself. Any functions that override the Google Maps SDK require this field in the class. Google Maps is quite easy to work with but it is somewhat limiting on the features implemented for the iOS SDK. The basis of this controller was based on tutorials provided by Ray Wenderlich[10]

3.3 Building Information

3.3.1 Overview

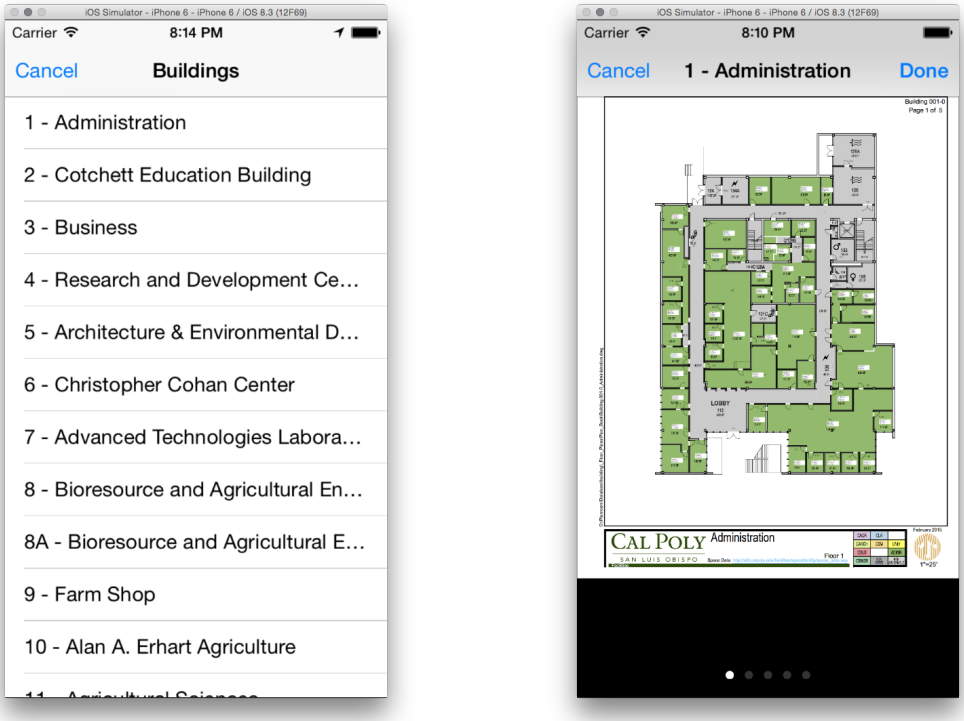
The core aspect of this app was incorporating the data specific to Cal Poly such as building numbers, building names, building floor count, and building floor plans. The raw information is provided in online resources by Cal Poly[9]. The information provided is extremely in depth and because some parts are not needed for our project, we had to strip out the useful data and store it using our own means. This included the building number and name as well as the floor plan images. We manually computed each building's coordinates using a resource from Cal Poly's Physics department[6]. We stored this data as images and a parseable CSV file included in the App's assets.

3.3.2 `ChooseBuildingRoomTableViewController`

The `ChooseBuildingRoomTableViewController` extends `UITableViewController` which is a simple table view list that handles the display of the Cal Poly buildings. An important aspect of this class is that it is reused multiple times throughout our interface including the building search, add-edit location, and when choosing a floor plan to view. Using the data we gathered from external resources, we display the building number and name on each row. On selection of a building, multiple actions can occur depending on where this controller is used. Our goal was to implement search functionality on this page, however we weren't able to complete it due to time constraints.

3.3.3 `FloorPlanPagedScrollViewController`

The `FloorPlanPagedScrollViewController` extends the classes: `UIViewController` and `UIScrollViewDelegate` and handles the display of all floor plans. This is a singleton controller that reloads the images each time a new building is selected. The



(a) The list of all Cal Poly Buildings (b) Floor plans for a selected building

Figure 6: Building Information Display

floor plan images to display are determined by the given filenames and the floors count of the building object. Although modified, the basis of this code arose from a Ray Wenderlich tutorial[10]. Our goal was to implement zooming on each page, however we weren't able to complete this task due to time constraints.

3.4 Viewing Saved Locations

3.4.1 Overview

If a cell in the locations view controller is swiped from right to left, two options are displayed: edit or delete. Tapping the edit button will allow a user to the location. Tapping the delete button will display a confirmation to delete. If delete is tapped, then the location is deleted forever.

If the cell's accessory button is tapped, three options are displayed: Move, Edit,

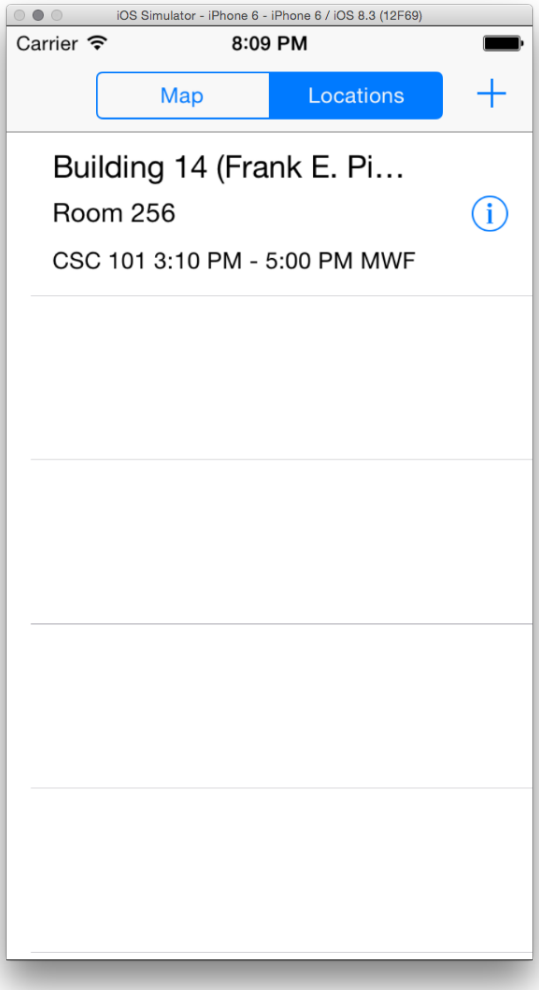
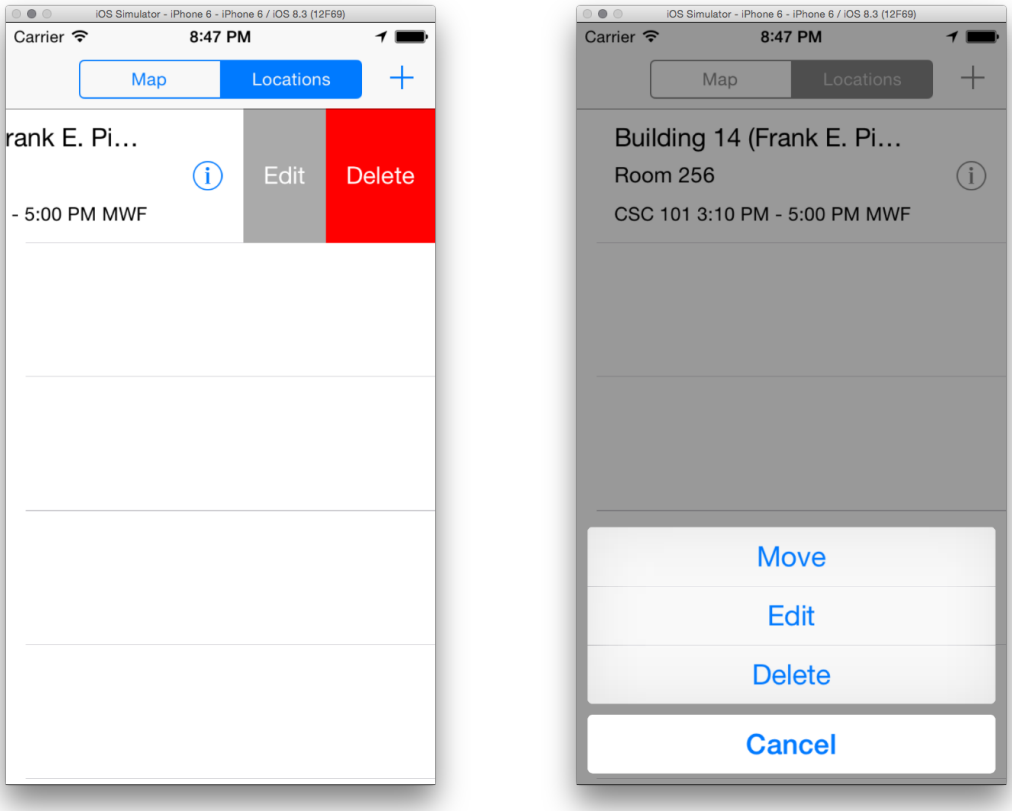


Figure 7: Viewing Saved Locations

and Delete. Tapping the edit button will allow a user to the location. Tapping the delete button will delete the location forever. Although modified, the basis of this code arose from a Ray Wenderlich tutorial[10].

3.4.2 `LocationsViewController`

`LocationsViewController` has a toolbar with alpha set to 1 and a `UITableView` and inherits `UIViewController`. A `UIViewController` was used rather than a



(a) Using swipe gesture to reveal edit and delete

(b) Using the accessory button to reveal edit and delete

Figure 8: Using swipe gesture vs accessory button

`UITableViewController` because a toolbar would offset the cells in the `UITableView`. Without the toolbar, the table cells would display under the `UINavigationBar` of `MainViewController` and a toolbar could not be added to `UITableViewController`. Thus, a `UIViewController` with a `UITableView` was used. The `UITableView` had both its delegate and source set to the `LocationsViewController`.

The cell in the `UITableView` are custom cells that displays the building number and name in the first line, the room number in the second, and the location's name, the start and end times, and the days that the class occurs. Also, a variable `isEditLocation` was used to determine whether to display the `AddEditLocationViewController` with information populated. If `isEditLocation` is true, then the location's data is

updated; else, a new location is created using the `CPMapsLibraryAPI`.

3.5 Adding or Editing a Location

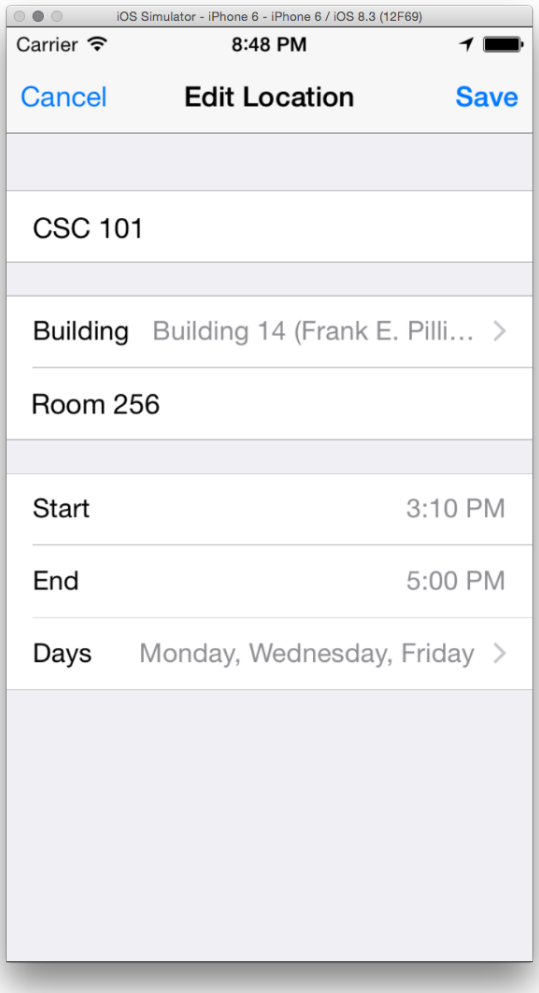


Figure 9: Viewing Saved Locations

3.5.1 Overview

When adding or editing a location, tapping on the name field allows a name to be added to a location. Tapping on building displays a list of buildings at Cal Poly. A chosen building will have a checkmark next to it. Once a building is selected, a room number can be added. Tapping on the start or end displays a date picker. Tapping on days displays the days of week. Selected day(s) will have checkmark(s) next to it. Although modified, the basis of this code arose from a Ray Wenderlich tutorial[10].

3.5.2 `AddEditLocationsViewController`

`AddEditLocationsViewController` has a `UINavigationBar` from being embedded in a `UINavigationController` and 3 sections to it.

The `UINavigationBar` has a cancel and save button. Tapping cancel will display the `LocationsViewController` without any information saved. Tapping save with at least a building selected will save the building; tapping save without a building will display a `UIAlertView` that notifies the user that a building needs to be selected first.

The first section has one cell with a `UITextField`. It allows a user to enter the name of the location.

The second section has two cells. The first cell segues to `ChooseBuildingRoomTableViewController`, which displays a list of buildings. A selected building will have a checkmark next to it. The second cell is user enabled only after a building has been selected. It has a `UITextField` that stores the room number of the location. However, anything can be typed into it and used as the room number.

The third section has 5 cells. The first and third cells are shown when the controller is initialized and display the current time. They hold the time as strings. The second and fourth cells are hidden until the first and third cells are tapped, respectively. They hold a `UIDatePicker` that only allows the time to be changed. The `UIDatePickers` display the current time. Changing the time will change the time in the first and third date cells. The fifth cell segues to `ChooseDaysViewController`, which displays a list of buildings and selected day(s) have checkmark(s) next to them [3][4].

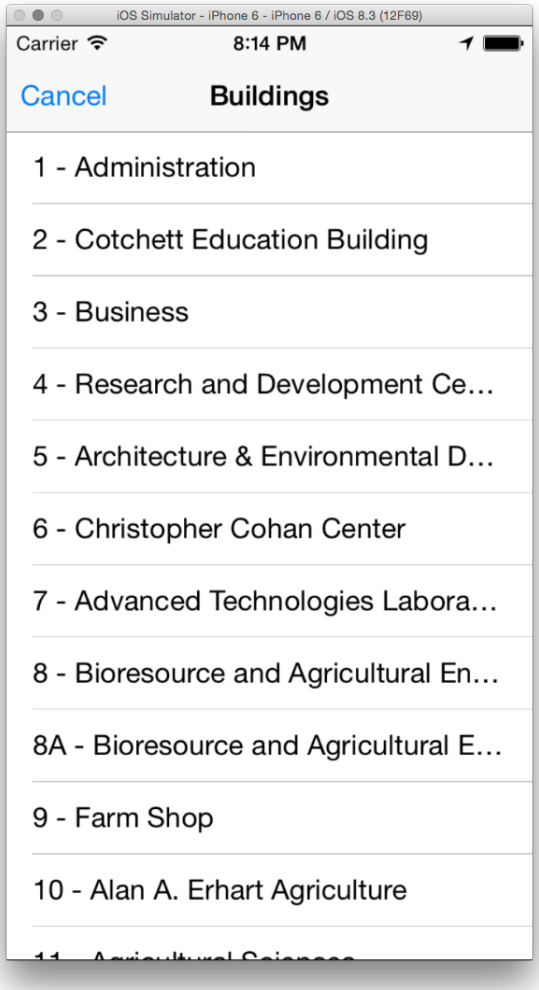


Figure 10: Saving a location without a building

3.6 Persistence

3.6.1 `CPMapsLibraryAPI`

`CPMapsLibraryAPI` is the interface that all other classes interact with. Although modified, the basis of this code arose from a Ray Wenderlich tutorial[10]. This creates a level of abstraction so that if the way the information is stored, the code would not have to change as much. For example, a class would have to get a location

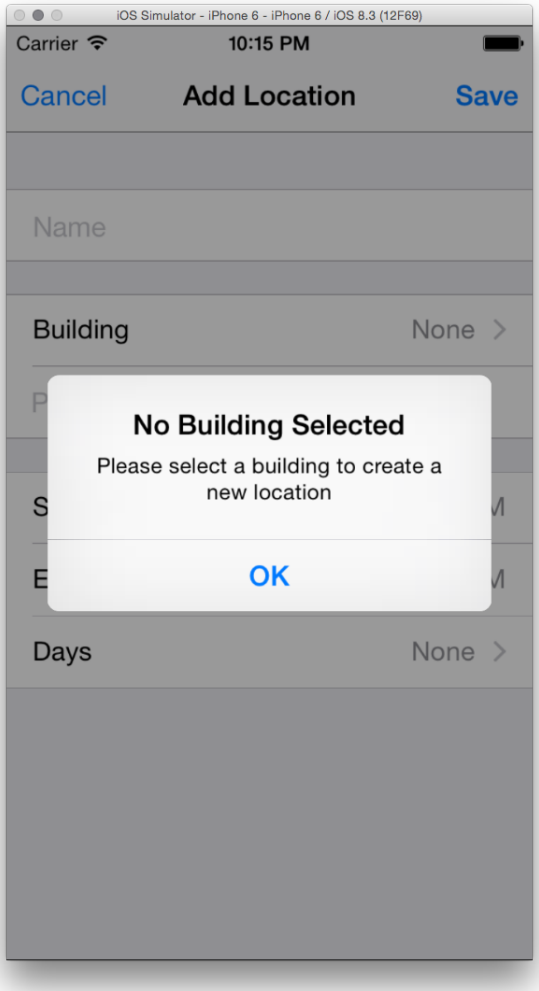


Figure 11: Showing list of buildings

using a function of the API and then getting the location's name, for example, using the location's function.

It stores a copy of `LocationsPersistencyManager` and `BuildingsPersistencyManager` and if a class wants to use it, it calls the static public variable `sharedInstance`.

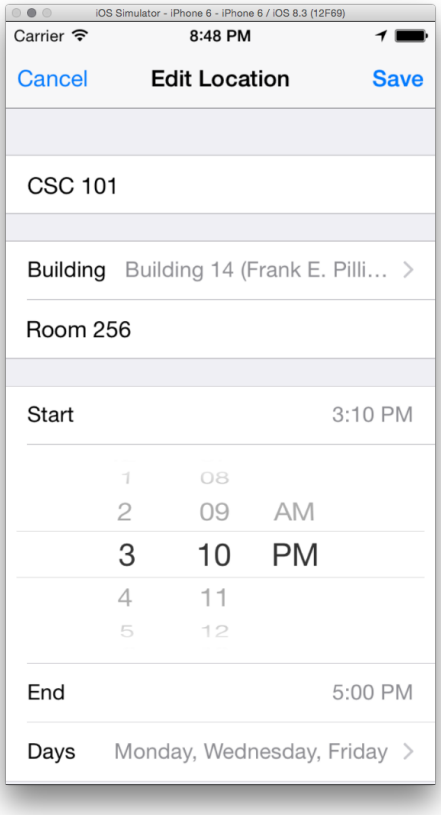


Figure 12: Showing inline `UIDatePicker`

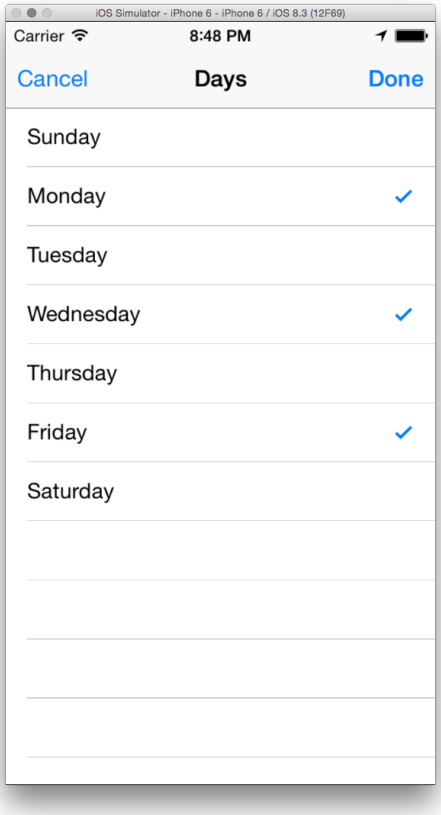


Figure 13: Showing selected days

3.6.2 `LocationsPersistencyManager`

`LocationsPersistencyManager` directly interacts with Core Data to store the locations persistently. It uses `NSFetchedResultsController` to interact with Core Data [2].

3.6.3 `BuildingsPersistencyManager`

`BuildingsPersistencyManager` saves the building data once it is loaded onto the app. It is stored in memory. We did not have time to implement the ability to store the buildings into `CoreData` if the app is put into sleep. However, we were able to save the locations persistently.

4 Challenges

Building our first iOS app using Swift presented many challenges: designing implementation with lots of technical debt, changing the UI once the tools for iOS were clear and learning Swift.

4.1 Technical Debt and Code Refactoring

The biggest challenge was that the app had multiple instances of technical debt. The code was refactored three times. The first time we refactored code, we were learning Swift. We followed tutorials mostly from Ray Wenderlich[1] and added custom functionality where we wanted. After this first phase, we refactored the code so that it was not full of commented out code. It also involved cutting unnecessary functions and adding needed ones where our code needed to be combined. We also added `CPMapsLibraryAPI`, which abstracted out the functions and would help us greatly in the second time we refactored code.

The second time we refactored code, we decided to implement Core Data. Not knowing that Core Data required inheritance of a completely different class, we had to entirely rewrite how we store our data. Also, there we a lot of random crashes and sometimes that data would not save. However, we finally fixed it using `NSFetchedResultsController` and not registering a cell when we were using the storyboard. The abstraction from the `CPMapsLibraryAPI` helped a lot because we only had to change how it was implemented internally without touching the other parts of the app much.

The third time we refactored code, we decided that we needed a new UI with the UX flow of the current app feeling too cluttered. So, we decided to add a

`UISegmentedControl` to the top and create a container controller. Creating the container controller was difficult because we did not know how to display the controller in the container controller's `UIView`. Also, the cells always displayed under the container controller's navigation bar (where the `UISegmentedControl` is) and we had to change the `LocationsViewController` that inherited from a `UITableViewController` to a `UIViewController` with a `UITableView`[1].

If we had to redo the app, now that we know how the code would connect and work together, we would design it properly from the first place rather than having to refactor it multiple times. The amount of refactoring we needed to do showed us the amount of technical debt a poorly designed project could have on the amount of time needed to finish the project. This project was still unfinished and is small compared to other projects in the industry.

4.2 Swift and Xcode

The next major challenge was learning Xcode and Swift's API. Learning Swift is not hard because we knew Java and C. But, knowing which functions were built in and which were not was hard. Also, knowing how all the classes connected, like how Core Data required its own class inheritance, really changed some of our design decisions. And, because we were new to Xcode, we did not fully utilize all the features until near the end of the project like using the built in search and replace for Xcode, which is some consider to be the official way to search and replace text.

4.3 Deadlines

During development we had a clear vision for the UI of the app, but we could have improved the process to get there. We started early, but, with our lack of knowledge and experience designing and programming iOS apps, we made design choices that created a lot of technical debt. We could have improved the process by setting hard deadlines for ourselves to finish developing a feature.

5 Future Plans

If we had more time, we would develop the following features:

- Search building
- Zoom floor plans

- Reorder saved locations however the user wants
- Choose to map directions or not (currently defaults to always show directions if your location is shared)
- Maps routes that are not roads (custom cal poly roads)
- Aggregate user data to find most popular or optimized path

We would also add automated testing our app. Due to the size and style of development, we were focusing on functionality over stability. Having these tests would be beneficial for regressive testing when making significant changes to the app in the future.

6 Conclusion

We were able to build an app that is nearly ready to be put into the App Store. We would consider it to be successful because we learned about how to design and implement an iOS app. If we did it a second time around, to reach this point in our app would take four to five weeks rather than two quarters.

We hope that other students at Cal Poly can take up this project and use what we have learned to expand upon our project. This would benefit visitors, students, and staff whether they are looking for something as common as the University Union, or a freshman finding a classroom inside Engineering West.

References

[1] Apple, "Technical qa qa1797: Preventing the status bar from covering your views," 2014. [Online]. Available: https://developer.apple.com/library/ios/qa/qa1797/_index.html

Preventing the Status Bar from Covering Your Views

[2] S. Archak, "Core data swift tutorial," 2015. [Online]. Available: http://shrikar.com/core-data-swift-tutorial/

Core Data Swift Tutorial from Shrikar Archak

[3] V. Costescu, "ios 7 in-line uidatepicker part 1," 2014. [Online]. Available: http://ios-blog.co.uk/tutorials/ios-7-in-line-uidatepicker-part-1/

How to create an inline UIDatePicker - Part 1

[4] ——, "ios 7 in-line uidatepicker part 1," 2014. [Online]. Available: http://ios-blog.co.uk/tutorials/ios-7-in-line-uidatepicker-part-2/

How to create an inline UIDatePicker - Part 2

[5] Y. D'areglia, "Working with custom container view controllers," 2013. [Online]. Available: http://www.thinkandbuild.it/working-with-custom-container-view-controllers/

Container View Controller Tutorial by Yari D'areglia

[6] C. P. P. Dept, "Cal poly latitude/longitude calculator," 2015. [Online]. Available: http://ocean.physics.calpoly.edu/longitude/map.php

A resource to determine the latitude and longitude of a selection

[7] C. Poly, "Cal poly: National honors, ranking and recognition," 2015. [Online]. Available: http://www.calpolynews.calpoly.edu/rankings.html

Cal Poly's listing of national honors, ranking and recognition

[8] ——, "Cal poly quick facts," 2015. [Online]. Available: http://calpolynews.calpoly.edu/quickfacts.html

Quick facts about Cal Poly's academics, student body, campus size, and more

[9] ——, "Maps and plans," 2015. [Online]. Available: https://afd.calpoly.edu/facilities/maps.asp

Cal Poly's online resources for all campus maps and floor plans

[10] R. Wenderlich, "Swift language tutorials," 2015. [Online]. Available: http://www.raywenderlich.com/swift-language-tutorials

Ray Wenderlich's online tutorials for the Swift Language

[11] N. Wilson, "Record numbers apply to cal poly," March 2015. [Online]. Available: http://www.sanluisobispo.com/2015/03/25/3555028_record-numbers-apply-to-cal-poly.html?rh=1

More than 55,000 applicants seeking admission to Cal Poly's undergraduate programs