Making Requests in a Playground

You are about to see examples of network requests that use the Parse and Udacity APIs. If you would like to experiment and run the requests in a Xcode Playground, then you'll need to do a little setup. Specifically, you need to tell the Playground to execute indefinitely. This gives the Playground the ability to continue its execution until the network request finishes and provides you with a response. Here is an example:

```swift
import UIKit
import PlaygroundSupport

// this line tells the Playground to execute
indefinitely
PlaygroundPage.current.needsIndefiniteExecution =
true

let urlString = "http://quotes.rest/qod.json?
category=inspire"
let url = URL(string: urlString)
let request = URLRequest(url: url!)
let session = URLSession.shared
let task = session.dataTask(with: request) { data,
response, error in
    if error != nil { // Handle error
        return
    }
    print(String(data: data!, encoding: .utf8)!)
}
task.resume()
```

I. Udacity API

Why the Udacity API?

For "On the Map", the Udacity API will be used to authenticate users of the app and to retrieve basic user info before posting data to Parse.

Getting Started

Besides having a Udacity account, there is no setup required to use the Udacity API.

Special Note on Udacity JSON Responses

FOR ALL RESPONSES FROM THE UDACITY API, YOU WILL NEED TO SKIP THE FIRST 5 CHARACTERS OF THE RESPONSE. These characters are used for security purposes. In the upcoming examples, you will see that we subset the response data in order to skip over the first 5 characters.

POSTing a Session

To authenticate Udacity API requests, you need to get a session ID. This is accomplished by using Udacity's session method:

- **Method:** https://www.udacity.com/api/session
- **Method Type:** POST
- **Required Parameters:**
    - *udacity* - (Dictionary) a dictionary containing a username/password pair used for authentication
    - *username* - (String) the username (email) for a Udacity student
    - *password* - (String) the password for a Udacity student
- **Example Request:**

```
var request = URLRequest(url: URL(string: "https://
www.udacity.com/api/session")!)
request.httpMethod = "POST"
request.addValue("application/json",
```

```
forHTTPHeaderField: "Accept")
request.addValue("application/json",
forHTTPHeaderField: "Content-Type")
request.httpBody = "{\"udacity\": {\"username\":
\"account@domain.com\", \"password\":
\"********\"}}".data(using: .utf8)
let session = URLSession.shared
let task = session.dataTask(with: request) { data,
response, error in
   if error != nil { // Handle error…
       return
   }
   let range = Range(5..<data!.count)
   let newData = data?.subdata(in: range) /* subset
response data! */
   print(String(data: newData!, encoding: .utf8)!)
}
task.resume()
```

DELETEing a Session

Once you get a session ID using Udacity's API, you should delete the session ID to "logout". This is accomplished by using Udacity's session method:

- **Method:** https://www.udacity.com/api/session
- **Method Type:** DELETE
- **Example Request:**

```
var request = URLRequest(url: URL(string: "https://
www.udacity.com/api/session")!)
request.httpMethod = "DELETE"
var xsrfCookie: HTTPCookie? = nil
let sharedCookieStorage = HTTPCookieStorage.shared
for cookie in sharedCookieStorage.cookies! {
```

```
  if cookie.name == "XSRF-TOKEN" { xsrfCookie =
cookie }
}
if let xsrfCookie = xsrfCookie {
  request.setValue(xsrfCookie.value,
forHTTPHeaderField: "X-XSRF-TOKEN")
}
let session = URLSession.shared
let task = session.dataTask(with: request) { data,
response, error in
  if error != nil { // Handle error…
      return
  }
  let range = Range(5..<data!.count)
  let newData = data?.subdata(in: range) /* subset
response data! */
  print(String(data: newData!, encoding: .utf8)!)
}
task.resume()
```

GETting Public User Data

The whole purpose of using Udacity's API is to retrieve some basic user information before posting data to Parse. This is accomplished by using Udacity's user method:

- **Method Name:** https://www.udacity.com/api/users/<user_id>
- **Method Type:** GET
- **Example Request:**

```
let request = URLRequest(url: URL(string: "https://
www.udacity.com/api/users/3903878747")!)
let session = URLSession.shared
let task = session.dataTask(with: request) { data,
response, error in
```

```
if error != nil { // Handle error...
    return
}
let range = Range(5..<data!.count)
let newData = data?.subdata(in: range) /* subset
response data! */
print(String(data: newData!, encoding: .utf8)!)
}
task.resume()
```

II. Parse API

Base URL for all endpoints changed.
From: https://api.parse.com/1/classes
To: https://parse.udacity.com/parse/classes

You will be required to specify the Parse Application ID and REST API Key for all requests to the Parse API:

Parse Application ID: QrX47CA9cyuGewLdsL7o5Eb8iug6Em8ye0dnAbIr
REST API Key: QuWThTdiRmTux3YaDseUSEpUKo7aBYM737yKd4gY

What is a Student Location?

A student location, or **StudentLocation** object in Parse, is used by "On the Map" to populate its map view with pins and its other views with information submitted by users.
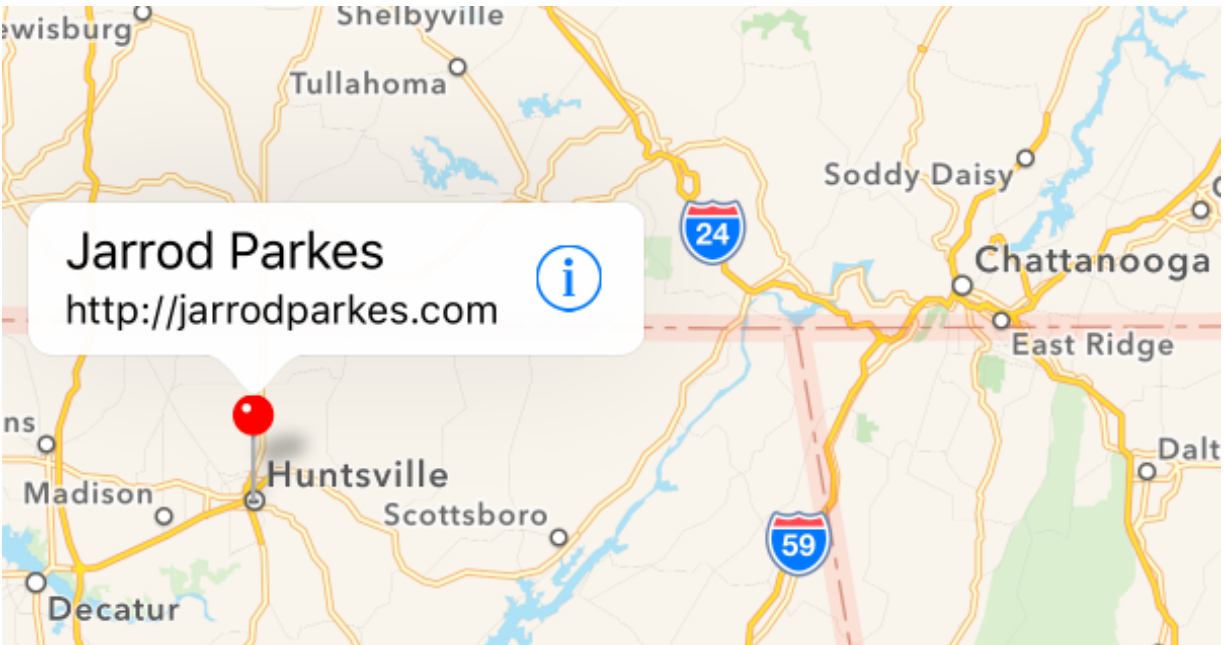
In Parse, each StudentLocation is represented as a collection of key/value pairs:

- **Key:** objectId
    - **Description:** an auto-generated id/key generated by Parse which uniquely identifies a StudentLocation
    - **Parse Type:** String
    - **Example Value:** 8ZExGR5uX8
- **Key:** uniqueKey
    - **Description:** an extra (optional) key used to uniquely identify a StudentLocation; *you should populate this value using your Udacity account id*
    - **Parse Type:** String
    - **Example Value:** 1234
- **Key:** firstName
    - **Description:** the first name of the student which matches their Udacity profile first name
    - **Parse Type:** String
    - **Example Value:** John
- **Key:** lastName

- **Description:** the last name of the student which matches their Udacity profile last name
- **Parse Type:** String
- **Example Value:** Doe

- **Key:** mapString
  - **Description:** the location string used for geocoding the student location
  - **Parse Type:** String
  - **Example Value:** Mountain View, CA
- **Key:** mediaURL
  - **Description:** the URL provided by the student
  - **Parse Type:** String
  - **Example Value:** https://udacity.com
- **Key:** latitude
  - **Description:** the latitude of the student location (ranges from -90 to 90)
  - **Parse Type:** Number
  - **Example Value:** 37.386052
- **Key:** longitude
  - **Description:** the longitude of the student location (ranges from -180 to 180)
  - **Parse Type:** Number
  - **Example Value:** -122.083851
- **Key:** createdAt
  - **Description:** the date when the student location was created
  - **Parse Type:** Date
  - **Example Value:** Feb 25, 2015, 01:10
- **Key:** updatedAt
  - **Description:** the date when the student location was last updated
  - **Parse Type:** Date
  - **Example Value:** Mar 09, 2015, 23:34
- **Key:** ACL
  - **Description:** the Parse access and control list (ACL), i.e. permissions, for this StudentLocation entry
  - **Parse Type:** ACL

- **Example Value:** Public Read and Write

When you are parsing StudentLocations, the Parse Types can be casted into equivalent Swift types. Any value with the type "Number" may be casted into a floating-point value in Swift. **YOU DO NOT HAVE TO WORRY ABOUT PARSING DATE OR ACL TYPES**.

GETting Multiple Student Locations

To get multiple student locations at one time, you'll want to use the following API method:

- **Method:** `https://parse.udacity.com/parse/classes/StudentLocation`
- **Method Type:** GET
- **Optional Parameters:**
  - *limit* - (Number) specifies the maximum number of StudentLocation objects to return in the JSON response
    - Example: `https://parse.udacity.com/parse/classes/StudentLocation?limit=100`
  - *skip* - (Number) use this parameter with limit to paginate through results
    - Example: `https://parse.udacity.com/parse/classes/StudentLocation?limit=200&skip=400`
  - *order* - (String) a comma-separate list of key names that specify the sorted order of the results
    - Prefixing a key name with a negative sign reverses the order (default order is ascending)
    - Example: `https://parse.udacity.com/parse/classes/StudentLocation?order=-updatedAt`
- **Example Request:**

```
var request = URLRequest(url: URL(string: "https://
parse.udacity.com/parse/classes/StudentLocation")!)
request.addValue("QrX47CA9cyuGewLdsL7o5Eb8iug6Em8ye0dnA
bIr", forHTTPHeaderField: "X-Parse-Application-Id")
request.addValue("QuWThTdiRmTux3YaDseUSEpUKo7aBYM737yKd
```

```
4gY", forHTTPHeaderField: "X-Parse-REST-API-Key")
let session = URLSession.shared
let task = session.dataTask(with: request) { data,
response, error in
  if error != nil { // Handle error...
      return
  }
  print(String(data: data!, encoding: .utf8)!)
}
task.resume()
```

GETting a Single Student Location

To get a single student location, you'll want to use the following API method:

- **Method:** https://parse.udacity.com/parse/classes/StudentLocation
- **Method Type:** GET
- **Required Parameters:**
  - *where* - (Parse Query) a SQL-like query allowing you to check if an object value matches some target value
    - Example: https://parse.udacity.com/parse/classes/StudentLocation?where=%7B%22uniqueKey%22%3A%221234%22%7D
    - the above URL is the escaped form of… https://parse.udacity.com/parse/classes/StudentLocation?where={"uniqueKey":"1234"}
    - you can read more about these types of queries in **Parse's REST API documentation**
- **Example Request:**
```
let urlString = "https://parse.udacity.com/parse/classes/StudentLocation"
let url = URL(string: urlString)
```

```swift
var request = URLRequest(url: url!)
request.addValue("QrX47CA9cyuGewLdsL7o5Eb8iug6Em8ye0dnA
bIr", forHTTPHeaderField: "X-Parse-Application-Id")
request.addValue("QuWThTdiRmTux3YaDseUSEpUKo7aBYM737yKd
4gY", forHTTPHeaderField: "X-Parse-REST-API-Key")
let session = URLSession.shared
let task = session.dataTask(with: request) { data,
response, error in
  if error != nil { // Handle error
      return
  }
  print(String(data: data!, encoding: .utf8)!)
}
task.resume()
```

POSTing a Student Location

To create a new student location, you'll want to use the following API method:

- **Method:** https://parse.udacity.com/parse/classes/
  StudentLocation
- **Method Type:** POST
- **Example Request:**

```swift
var request = URLRequest(url: URL(string: "https://
parse.udacity.com/parse/classes/StudentLocation")!)
request.httpMethod = "POST"
request.addValue("QrX47CA9cyuGewLdsL7o5Eb8iug6Em8ye0dnA
bIr", forHTTPHeaderField: "X-Parse-Application-Id")
request.addValue("QuWThTdiRmTux3YaDseUSEpUKo7aBYM737yKd
4gY", forHTTPHeaderField: "X-Parse-REST-API-Key")
request.addValue("application/json",
forHTTPHeaderField: "Content-Type")
request.httpBody = "{\"uniqueKey\": \"1234\",
```

```
\"firstName\": \"John\", \"lastName\": \"Doe\",
\"mapString\": \"Mountain View, CA\", \"mediaURL\":
\"https://udacity.com\",\"latitude\": 37.386052,
\"longitude\": -122.083851}".data(using: .utf8)
let session = URLSession.shared
let task = session.dataTask(with: request) { data,
response, error in
  if error != nil { // Handle error…
      return
  }
  print(String(data: data!, encoding: .utf8)!)
}
task.resume()
```

PUTting a Student Location

To update an existing student location, you'll want to use the following API method:

- **Method:**https://parse.udacity.com/parse/classes/
  StudentLocation/<objectId>
- **Method Type:** PUT
- **Required Parameters:**
  - *objectId* - (String) the object ID of the StudentLocation to update; specify
    the object ID right after StudentLocation in URL as seen below
    - Example: https://parse.udacity.com/parse/classes/
      StudentLocation/8ZExGR5uX8
- **Example Request:**

```
let urlString = "https://parse.udacity.com/parse/
classes/StudentLocation/8ZExGR5uX8"
let url = URL(string: urlString)
var request = URLRequest(url: url!)
request.httpMethod = "PUT"
request.addValue("QrX47CA9cyuGewLdsL7o5Eb8iug6Em8ye0dnA
```

```swift
bIr", forHTTPHeaderField: "X-Parse-Application-Id")
request.addValue("QuWThTdiRmTux3YaDseUSEpUKo7aBYM737yKd
4gY", forHTTPHeaderField: "X-Parse-REST-API-Key")
request.addValue("application/json",
forHTTPHeaderField: "Content-Type")
request.httpBody = "{\"uniqueKey\": \"1234\",
\"firstName\": \"John\", \"lastName\": \"Doe\",
\"mapString\": \"Cupertino, CA\", \"mediaURL\":
\"https://udacity.com\",\"latitude\": 37.322998,
\"longitude\": -122.032182}".data(using: .utf8)
let session = URLSession.shared
let task = session.dataTask(with: request) { data,
response, error in
  if error != nil { // Handle error…
      return
  }
  print(String(data: data!, encoding: .utf8)!)
}
task.resume()
```

III. Facebook Login

Adding Facebook Login (Optional)

NOTE: Adding the ability to login with Facebook is **OPTIONAL** and not a requirement to complete "On the Map".

If you'd like to add the ability to login/authenticate to Udacity via Facebook, then before you dig into the code, you'll need to take the following steps:

1. Read through **Facebook's Getting Started documentation**, **Facebook's Preparing iOS 9 Guide**, and investigate the login feature
   - You will need to use the Facebook API ID "365362206864879"
   - You will need to use the Facebook URL Scheme Suffix "onthemap"
2. Connect your Udacity user to your Facebook account in your Udacity **Account Settings**

POSTing a Session with Facebook

To authenticate Udacity API requests, you need to get a session ID. This is accomplished by using Udacity's session method **with a special consideration when using Facebook**:

Udacity's session method only accepts one credential at a time. So, when using Facebook Authentication, you do not need to supply values for *udacity/username/ password*. If you do, then you will receive an error that says "Did not specify exactly one credential". Likewise, if you are using *udacity/username/password*, then do not supply values for *facebook_mobile/access_token*.

- **Method:** `https://www.udacity.com/api/session`
- **Method Type:** POST
- **Udacity Facebook App ID:** 365362206864879
- **Udacity Facebook URL Scheme Suffix:** onthemap
- **Required Parameters:**
   - *facebook_mobile* - (Dictionary) a dictionary containing an access token from a valid Facebook session
   - *access_token* - (String) the user access token from Facebook
      - an access token is made available through the FBSDKAccessToken class
   - The Facebook SDK was recently updated to version 4.0. According to the upgrade guide: `FBSession.activeSession` has been replaced with `[FBSDKAccessToken currentAccessToken]` and `FBSDKLoginManager`. There is no concept of session state. Instead, use the manager to login and this sets

the `currentAccessToken` reference.

- **Example Request:**

```swift
var request = URLRequest(url: URL(string: "https://
www.udacity.com/api/session")!)
request.httpMethod = "POST"
request.addValue("application/json",
forHTTPHeaderField: "Accept")
request.addValue("application/json",
forHTTPHeaderField: "Content-Type")
request.httpBody = "{\"facebook_mobile\":
{\"access_token\":
\"DADFMS4SN9e8BAD6vMs6yWuEcrJlMZChFB0ZB0PCLZBY8FPFYxIPy
1WOr402QurYWm7hj1ZCoeoXhAk2tekZBIddkYLAtwQ7PuTPGSERwH1D
fZC5XSef3TQy1pyuAPBp5JJ364uFuGw6EDaxPZBIZBLg192U8vL7mZA
zYUSJsZA8NxcqQgZCKdK4ZBA2l2ZA6Y1ZBWHifSM0slybL9xJm3ZBbT
XSBZCMItjnZBH25irLhIvbxj01QmlKKP3iOnl8Ey;
\"}}".data(using: .utf8)
let session = URLSession.shared
let task = session.dataTask(with: request) { data,
response, error in
  if error != nil { // Handle error...
      return
  }
  let range = Range(5..<data!.count)
  let newData = data?.subdata(in: range) /* subset
response data! */
  print(String(data: newData!, encoding: .utf8)!)
}
task.resume()
```

Logout with Facebook

If you're implementing Facebook login, then deleting a session with Udacity's API is just

the first step of "logging out". To be thorough, you should also logout of your Facebook session.

This can be accomplished using Facebook's `FBSDKLoginButton` or programmatically. With the `FBSDKLoginManager`, the logout behavior is built-in; however, if you want to log out programmatically, you'll need to use the `FBSDKLoginManager`. For help, consult **Facebook's iOS documentation**.

IV. MapKit

Working with Maps

For "On the Map" you will need to use the MapKit framework. To help you get started, download and run the **Pin Sample App**. It uses the following items from MapKit:

- `MKMapView`
  - This is the view that displays a map. In the sample code, it fills the entire view.
- `MKMapViewDelegate`
  - The view controller conforms to this protocol to respond to pin taps.
- `MKPointAnnotation`
  - The class that is used to hold the data for the pins.

A map view can be supplied with an array of annotation objects. In "On the Map", you will create `MKPointAnnotation` objects from data that you download using the Parse API.

V. On the Map specs

App Specification: On the Map

iOS Developer Nanodegree

[Note that this is an informal app description. It will give you an idea how the app should work, but when you submit your app it will be rated based on the [Rubric](.)]

The On The Map app allows users to share their location and a URL with their fellow students. To visualize this data, On The Map uses a map with pins for location and pin annotations for student names and URLs, allowing students to place themselves "on the map," so to speak.

First, the user logs in to the app using their Udacity username and password. After login, the app downloads locations and links previously posted by other students. These links can point to any URL that a student chooses. We encourage students to share something about their work or interests.

After viewing the information posted by other students, a user can post their own location and link. The locations are specified with a string and forward geocoded. They can be as specific as a full street address or as generic as "Costa Rica" or "Seattle, WA."

The app has three view controller scenes:

Login View: Allows the user to log in using their Udacity credentials, or (as an extra credit exercise) using their
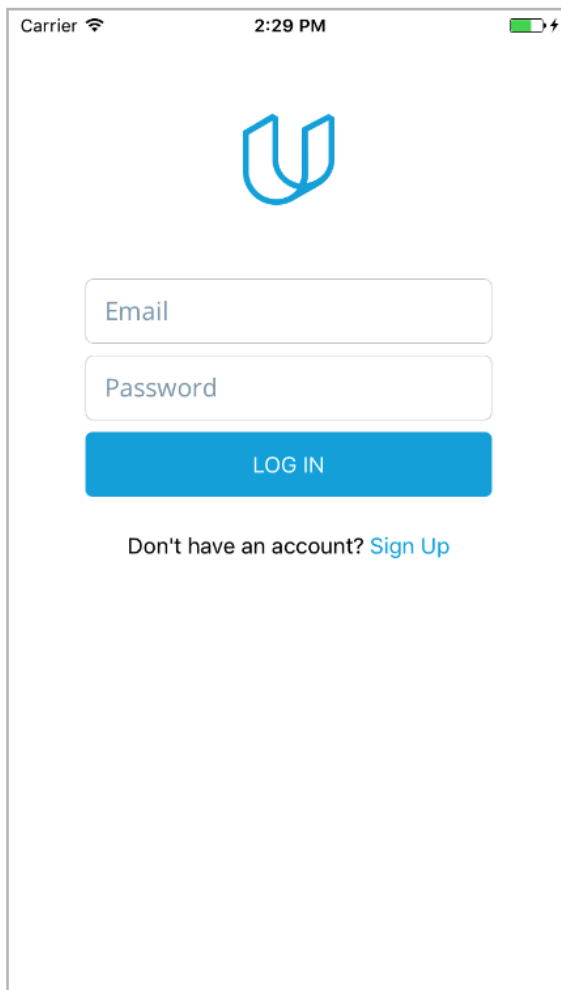
Facebook account

Map and Table Tabbed View: Allows users to see the locations of other students in two formats.

Information Posting View: Allows the users specify their own locations and links.

These three scenes are described in detail below.

Login View

The login view accepts the email address and password that students use to login to the Udacity site. User credentials are **not** required to be saved upon successful login.

When the user taps the **Login** button, the app will attempt to authenticate with Udacity's servers.

Clicking on the **Sign Up** link will open Safari to the Udacity [sign-up page](#).

If the connection is made and the email and password are good, the app will segue to the **Map and Table Tabbed View**.

If the login does not succeed, the user will be presented with an alert view specifying whether it was a failed network connection, or an incorrect email and password.

Optional (but fun) task: The "Sign in with Facebook" button in the image authenticates with Facebook. Authentication with Facebook may occur through the device's accounts or through Facebook's website.

Map And Table Tabbed View
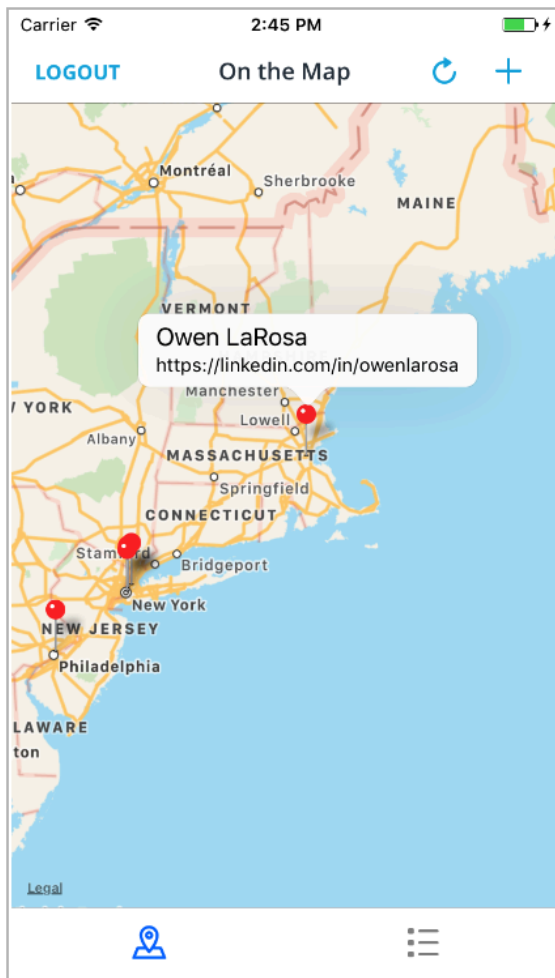
This view has two tabs at the bottom: one specifying a map, and the other a table.

When the **map tab** is selected, the view displays a map with pins specifying the last 100 locations posted by students.

The user is able to zoom and scroll the map to any location using standard pinch and drag gestures.

When the user taps a pin, it displays the pin annotation popup, with the student's name (pulled from their Udacity profile) and the link associated with the student's pin.

Tapping anywhere within the annotation will launch Safari and direct it to the link associated with the pin.
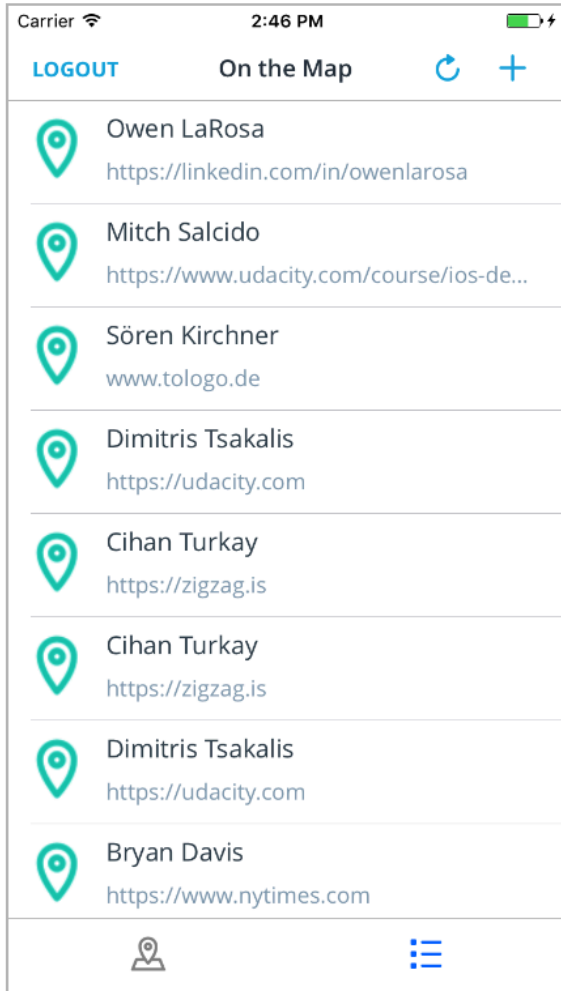
Tapping outside of the annotation will dismiss/hide it.

When the **table tab** is selected, the most recent 100 locations posted by students are displayed in a table. Each row displays the name from the student's Udacity profile. Tapping on the row launches Safari and opens the link associated with the student.
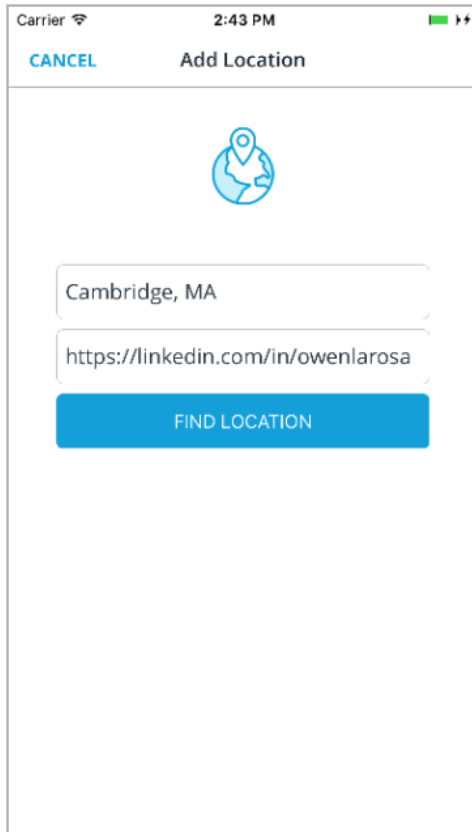
Both the map tab and the table tab share the same top navigation bar.

The rightmost bar button will be a refresh button. Clicking on the button will refresh the entire data set by downloading and displaying the most recent 100 posts made by students.

The bar button directly to its left will be a pin button. Clicking on the pin button will modally present the **Information Posting View**.

Optional (but fun) task: If authentication with Facebook is enabled, consider placing a bar button in the top left corner which will allow to user to logout.

Information Posting View

The Information Posting View allows users to input their own data.

When the **Information Posting View** is modally presented, the user sees two text fields: one asks for a location and the other asks for a link.

When the user clicks on the "Find Location" button, the app will forward geocode the string. If the forward geocode fails, the app will display an alert view notifying the user. Likewise, an alert will be displayed if the link is empty.

If the forward geocode succeeds then text fields will be hidden, and a map showing the entered location will be displayed. Tapping the "Finish" button will post the location and link to the server.

If the submission fails to post the data to the server, then the user should see an alert with an error message describing the failure.

If at any point the user clicks on the "Cancel" button, then the Information Posting View should be dismissed, returning the app to the Map and Table Tabbed View.

Likewise, if the submission succeeds, then the Information Posting View should be dismissed, returning the app to the Map and Table Tabbed View.

VI. Grading Rubric

On the Map

Login View

CRITERIA	MEETS SPECIFICATIONS
Does the app allow the user to login?	The app has a login view that accepts email and password strings from users, with a "Login" button.
Does the app notify the user if the login fails?	The app informs the user if the login fails. It differentiates between a failure to connect, and incorrect credentials (i.e., wrong email or password).

Student Locations Tabbed View

CRITERIA	MEETS SPECIFICATIONS
Does the app download locations and links previously posted by students? *(to be displayed in the Map and Table tabs)*	The app downloads the 100 most recent locations posted by students.
Does the app contain a `StudentInformation` struct to store individual locations and links downloaded from the service?	The app contains a `StudentInformation` struct with appropriate properties for locations and links.

Does the `StudentInformation` struct initialize instances from a dictionary?	The `struct` has an `init()` method that accepts a dictionary as an argument.
Does the app store the array of `StudentInformation` `structs`in a single location, outside of the view controllers?	The `StudentInformation` `structs` are stored as an array (or other suitable data structure) inside a separate `model` class.
Does the app inform the user if the download fails?	The app gracefully handles a failure to download student locations.
Does the app correctly display the downloaded data in a tabbed view?	The app displays downloaded data in a tabbed view with two tabs: a map and a table.
Does the map view contain a pin for each of the locations that were downloaded?	The map view has a pin for each student in the correct location.
When the pins in the map are tapped, is a pin annotation displayed with the relevant information?	Tapping the pins shows an annotation with the student's name and the link the student posted.
If the pin annotation is tapped, is the link opened in Safari?	Tapping a student's pin annotation opens the student's link in Safari or a web view.

Does the table view contain a row for each downloaded location, with the student's name displayed?	The table view has a row for each downloaded record with the student's name displayed.
Is the table appropriately sorted?	The table is sorted in order of most recent to oldest update.
When a row in the table is tapped, does the app open Safari to the student's link?	Tapping a row in the table opens the default device browser to the student's link.
Does the Student Locations Tabbed View have a pin button in the appropriate location?	The Student Locations Tabbed View has a pin button in the upper right corner of the navigation bar.
Does the pin button in the navigation bar allow users to post their own information to the server?	The button presents the Information Posting View so that users can post their own information to the server.
Does the app have a logout button in the appropriate location?	The Student Locations Tabbed View has a logout button in the upper left corner of the navigation bar.
Does the logout button work as intended?	The logout button causes the Student Locations Tabbed View to dismiss, and logs out of the current session.

Information Posting View

CRITERIA	MEETS SPECIFICATIONS

Does the Information Posting View clearly indicate that the user should enter a location?	The Information Posting view prompts users to enter a `string` representing their location.
Does the Information Posting View clearly provide a place for the user to enter a string?	The text view or text field where the location string should be typed is clearly present.
Does the app allow users to enter a URL to be included with their location?	The app allows users to add a URL to be included with their location.
Does the app provide a button for the user to post the information to the server?	The app provides a readily accessible "Submit" button that the user can tap to post the information to the server.

Does the app geocode an address string when the submit button is tapped?	When a "Submit" button is pressed, the app forward geocodes the address string and stores the resulting latitude and longitude. Foward geocoding can be accomplished using CLGeocoder's `geocodeAddressString()` or MKLocalSearch's `startWithCompletionHandler()`.
Does the app indicate activity during the geocoding?	An activity indicator is displayed during geocoding, and returns to normal state on completion.
Does the app notify the user if the geocoding fails?	The app informs the user if the geocoding fails.
Does the app properly show the geocoded response on a map?	The app shows a placemark on a map via the geocoded response. The app zooms the map into an appropriate region.
Does the app post the search string and coordinates to the RESTful service?	The app successfully encodes the data in JSON and posts the search string and coordinates to the RESTful service.

Does the app readily allow the user to dismiss the Information Posting View?	The app provides a readily accessible button that the user can tap to cancel (dismiss) the Information Posting View.
Does the app notify the user if the post fails?	The app inform the user if the post fails.

Networking Architecture

CRITERIA	MEETS SPECIFICATIONS
Is the networking and JSON parsing code placed in its own class?	The networking and JSON parsing code is located in a dedicated API client class (and not, for example, inside a view controller). The class uses closures for completion and error handling.
Does the networking code use Swift's built-in `URLSession` class?	The networking code uses Swift's built-in `URLSession` library, not a third-party framework.
Does the JSON parsing code use Swift's built-in JSON parsing capabilities?	The JSON parsing code uses Swift's built-in `JSONSerialization` library or `Codable`, not a third-party framework.

Suggestions to Make Your Project Stand Out!

- The login page includes a "Login Using Facebook" option.

  ** If a user logs in with Facebook, then the logout button should log out of

the current Facebook session.

- The app shows additional indications of activity, such as modifying alpha/ transparency of interface elements during geocoding.