**On The Map**

**Login View**

* The app has a login view that accepts email and password strings from users, with a “Login” button.
* 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**

* The app contains a StudentInformation struct with appropriate properties for locations and links.
* The app downloads the 100 most recent locations posted by students.
* The struct has an init() method that accepts a dictionary as an argument, or the struct conforms to the Codable protocol.
* The StudentInformation structs are stored as an array (or other suitable data structure) inside a separate model class.
* The app gracefully handles a failure to download student locations.
* The app displays downloaded data in a tabbed view with two tabs: a map and a table.
* The map view has a pin for each student in the correct location.
* Tapping the pins shows an annotation with the student's name and the link the student posted.
* Tapping a student’s pin annotation opens the student’s link in Safari or a web view.
* The table view has a row for each downloaded record with the student’s name displayed.
* The table is sorted in order of most recent to oldest update.
* Tapping a row in the table opens the default device browser to the student's link.
* The Student Locations Tabbed View has a pin button in the upper right corner of the navigation bar.
* The button presents the Information Posting View so that users can post their own information to the server.
* The Student Locations Tabbed View has a logout button in the upper left corner of the navigation bar.
* The logout button causes the Student Locations Tabbed View to dismiss, and logs out of the current session.

**Information Posting View**

* The Information Posting view prompts users to enter a string representing their location.
* The text view or text field where the location string should be typed is clearly present.
* The app allows users to add a URL to be included with their location.
* The app provides a readily accessible "Submit" button that the user can tap to post the information to the server.
* 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().
* An activity indicator is displayed during geocoding, and returns to normal state on completion.
* The app informs the user if the geocoding fails.
* The app shows a placemark on a map via the geocoded response. The app zooms the map into an appropriate region.
* The app successfully encodes the data in JSON and posts the search string and coordinates to the RESTful service.
* The app provides a readily accessible button that the user can tap to cancel (dismiss) the Information Posting View.
* The app inform the user if the post fails.

**Networking Architecture**

* 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.
* The networking code uses Swift's built-in URLSession library, not a third-party framework.
* The JSON parsing code uses Swift's built-in JSONSerialization library or Codable, not a third-party framework.