How will I complete this project?

1. Review our course JavaScript Design Patterns and check out the Neighborhood Map project rubric.
2. Download the Knockout framework.
   1. Knockout must be used to:
      1. handle the list,
      2. filter,
      3. and any other information on the page that is subject to changing state.
   2. Things that should not be handled by Knockout include anything the Maps API is used for:
      1. creating markers,
      2. tracking click events on markers,
      3. making the map, refreshing the map

Note 1: Tracking click events on list items should be handled with Knockout.

Note 2: Creating your markers as a part of your ViewModel is allowed (and recommended). Creating them as Knockout observables is not.

1. Asynchrony and Error Handling.
   1. Note that all data APIs used in the project should load asynchronously and errors should be handled gracefully. In case of error (e.g. in a situation where a third party API does not return the expected result) we expect your webpage to do one of the following:
      1. A message is displayed notifying the user that the data can't be loaded, OR,
      2. there are no negative repercussions to the UI.

Note: Please note that we expect students to handle errors if the browser has trouble initially reaching the 3rd-party site as well. For example, imagine a user is using your Neighborhood Map, but her firewall prevents her from accessing the Instagram servers. Here is a reference article on how to block websites with the hosts file.

It is important to handle errors to give users a consistent and good experience with the webpage. Read this blogpost to learn more. Some JavaScript libraries provide special methods to handle errors.

For example: refer to .fail() method discussed here if you use jQuery's ajax() method. We strongly encourage you to explore ways to handle errors in the library you are using to make API calls.

1. Write code required to add a full-screen map to your page using the Google Maps API. For sake of efficiency, the ***map*** API should be called only once.
2. If you are prompted to do so, you may want to get a Google Maps API key, and include it as the value of the key parameter when loading the Google Maps API in index.html: <script src="http://maps.googleapis.com/maps/api/js?libraries=places&key=[YOUR\_API\_KEY]"></script> You may have some initial concerns with placing your API key directly within your JavaScript source files, but rest assured this is perfectly safe. All client-side code must be downloaded by the client; therefore, the client must download this API key - it is not intended to be secret. Google has security measures in place to ensure your key is not abused. It is not technically possible to make anything secret on the client-side.
3. Write code required to display map markers identifying at least 5 locations that you are interested in within this neighborhood. Your app should display those locations by default when the page is loaded. ***BP – pt asta ei vor knockout.js cu array***
4. **Implement a list view** of the set of locations defined in step 5.
5. **Provide a filter option** that uses an input field to filter both the list view and the map markers displayed by default on load. The list view and the markers should update accordingly in real time. Providing a search function through a third-party API is not enough to meet specifications. This filter can be a text input or a dropdown menu.
6. Add functionality using **third-party APIs to provide information when a map marker or list view entry is clicked** (ex: Yelp reviews, Wikipedia, Flickr images, etc). Note that StreetView and Places don't count as an additional 3rd party API because they are libraries included in the Google Maps API.

If you need a refresher on making AJAX requests to third-party servers, check out our Intro to AJAX course. Please provide attribution to the data sources/APIs you use. For example if you are using Foursquare, indicate somewhere in your interface and in your README that you used Foursquare's API.

1. Add functionality **to animate a map marker** when either the list item associated with it or the map marker itself is selected.
2. Add functionality to open an infoWindow with the information described in step 9 (you can also populate a DOM element with this info, but you should still open an infoWindow, even with minimal info, like location name) when either a location is selected from the list view or its map marker is selected directly.
3. The app's interface should be intuitive to use. For example,
   1. the input text area to filter locations should be easy to locate.
   2. It should be easy to understand what set of locations is being filtered.
   3. selecting a location via list item or map marker should cause the map marker to bounce or in some other way animate to indicate that the location has been selected and associated info window should open above the map marker with additional information.

***CRITERIA for MEETS SPECIFICATIONS***

1. **Interface Design**

Responsiveness - All application components render on-screen in a responsive manner.

Usability - All application components are usable across modern desktop, tablet, and phone browsers.

1. **App Functionality**

Filter Locations - Includes a text input field or dropdown menu that filters the map markers and list items to locations matching the text input or selection. Filter function runs error-free.

List View

* A list-view of location names is provided which displays all locations by default, and displays the filtered subset of locations when a filter is applied.
* Clicking a location on the list displays unique information about the location, and animates its associated map marker (e.g. bouncing, color change.)
* List functionality is responsive and runs error free.

Map and Markers

* Map displays all location markers by default, and displays the filtered subset of location markers when a filter is applied.
* Clicking a marker displays unique information about a location in either an infoWindow or DOM element.
* Markers should animate when clicked (e.g. bouncing, color change.)
* Any additional custom functionality provided in the app functions error-free.

App Architecture

* Code is properly separated based upon Knockout best practices (follow an MVVM pattern, avoid updating the DOM manually with jQuery or JS, use observables rather than forcing refreshes manually, etc). Knockout should not be used to handle the Google Map API.
* There are at least 5 locations in the model. These may be hard-coded or retrieved from a data API

1. **Asynchronous Data Usage**

Asynchronous API Requests

* Application utilizes the Google Maps API and at least one non-Google third-party API. Refer to this documentation
* All data requests are retrieved in an asynchronous manner.

Error handling

* Data requests that fail are handled gracefully using common fallback techniques (i.e. AJAX error or fail methods). 'Gracefully' means the user isn’t left wondering why a component isn’t working. If an API doesn’t load there should be some visible indication on the page (an alert box is ok) that it didn’t load. Note: You do not need to handle cases where the user goes offline.

1. **Location Details Functionality**

Additional Location Data

* Functionality providing additional data about a location is provided and sourced from a 3rd party API. Information can be provided either in the marker’s infoWindow, or in an HTML element in the DOM (a sidebar, the list view, etc.)

Error Free

* Application runs without errors.

Usability

* Functionality is presented in a usable and responsive manner.

1. **Documentation**

README - A README file is included detailing all steps required to successfully run the application.

Comments - Comments are present and effectively explain longer code procedures.

Code Quality

* Code is formatted with consistent, logical, and easy-to-read formatting as described in the Udacity JavaScript Style Guide.
* If build tools (such as Gulp or Grunt) are used, both source and production code are submitted in the same repository in separate directories. These directories are usually named src and dist respectively.

**Suggestions to Make Your Project Stand Out!**

Add unique functionality beyond the minimum requirements (i.e. the ability to “favorite” a location, etc.).

Incorporate a build process allowing for production quality, minified code, to be delivered to the client.

Data persists when the app is closed and reopened, either through localStorage or an external database (e.g. Firebase).

Include additional third-party data sources beyond the minimum required.

Style different markers in different (and functionally-useful) ways, depending on the data set.

Implement additional optimizations that improve the performance and user experience of the filter functionality (keyboard shortcuts, autocomplete functionality, filtering of multiple fields, etc).

Integrate all application components into a cohesive and enjoyable user experience.