

## PROJECT SPECIFICATION

## Neighborhood Map

## Interface Design

CRITERIA	MEETS SPECIFICATIONS
Responsiveness	All application components render on-screen in a responsive manner.
Usability	All application components are usable across modern desktop, tablet, and phone browsers.

## App Functionality

CRITERIA	MEETS SPECIFICATIONS
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	<p>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.</p> <p>Clicking a location on the list displays unique information about the location, and animates its associated map marker (e.g. bouncing, color change.)</p> <p>List functionality is responsive and runs error free.</p>
Map and Markers	<p>Map displays all location markers by default, and displays the filtered subset of location markers when a filter is applied.</p> <p>Clicking a marker displays unique information about a location in either an <code>infoWindow</code> or <code>DOM</code> element.</p> <p>Markers should animate when clicked (e.g. bouncing, color change.)</p> <p>Any additional custom functionality provided in the app functions error-free.</p>

## App Architecture

CRITERIA	MEETS SPECIFICATIONS
----------	----------------------

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

### Asynchronous Data Usage

CRITERIA	MEETS SPECIFICATIONS
Asynchronous API Requests	<p>Application utilizes the <code>Google Maps API</code> and at least one non-Google third-party <code>API</code>. Refer to <a href="#">this documentation</a></p> <p>All data requests are retrieved in an asynchronous manner.</p>
Error Handling	<p>Data requests that fail are handled gracefully using common fallback techniques (i.e. <code>AJAX</code> error or fail methods). 'Gracefully' means the user isn't left wondering why a component isn't working. If an <code>API</code> doesn't load there should be some visible indication on the page (an alert box is ok) that it didn't load. <i>Note:</i> You do not need to handle cases where the user goes offline.</p>

### Location Details Functionality

CRITERIA	MEETS SPECIFICATIONS
Additional Location Data	<p>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 <code>infoWindow</code>, or in an <code>HTML</code> element in the <code>DOM</code> (a sidebar, the list view, etc.)</p> <p>Provide attribution for the source of additional data. For example, if using Foursquare, indicate somewhere in your UI and in your README that you are using Foursquare data.</p>
Error Free	Application runs without errors.
Usability	Functionality is presented in a usable and responsive manner.

### Documentation

CRITERIA	MEETS SPECIFICATIONS
README	A <code>README</code> file is included detailing all steps required to successfully run the application.

Comments	Comments are present and effectively explain longer code procedures.
Code Quality	<p>Code is formatted with consistent, logical, and easy-to-read formatting as described in the <a href="#">Udacity JavaScript Style Guide</a>.</p> <p>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 <code>src</code> and <code>dist</code> respectively.</p>

---

### 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.