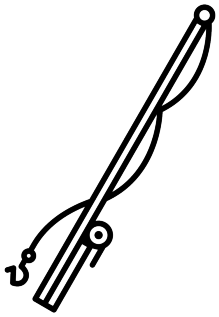

BENEFISHER

Software Test Specification

Version 2.0

11/25/2014

TEAM WAKATI

TABLE OF CONTENTS

1. INTRODUCTION

This document is the Software Test Specification for the Benefisher project sponsored by Code for Sacramento, to be completed by Team Wakati.

Project Team

Team Wakati is comprised of undergraduate students majoring in Computer Science at California State University, Sacramento. The team members are enrolled in a two-semester senior project course required of all undergraduate majors. Successful delivery of the desired software product will fulfill the senior project requirement for the student team members.

Name	Email	Phone
Adrian Chambers	adr510909@gmail.com	(707) 430-3775
Anthony Cristiano	cristiano@csus.edu	(925) 321-7648
James Doan	jhdoan@gmail.com	(949) 690-4212
Daniel Green	djgreensolving@gmail.com	(209) 402-3658
Jesse Rosato	jesse.rosato@gmail.com	(916) 541-5386

Table 1.1 Team Wakati Members

Project Sponsor

Code for Sacramento is a Code for America Brigade, whose mission is "to help government work for the people, by the people" [1]. Code4Sac has aligned the goal of improving access to public services data with their central missions of “connecting citizens and governments to design better services” and “open[ing] civic data” [1]. To that end, the sponsor has proposed the project under discussion.

Name	Role	Email
Brandon Pugh	Brigade Captain	bpugh143@gmail.com
Ash Roughani	Community Organizer	ash@publicinnovation.org

Table 1.2 Code for America Representatives

1.1 Purpose

The purpose of the STS is to describe the plan for testing the software, and to specify the test cases and test procedures necessary to demonstrate that the software satisfies the requirements as specified in the project's System Requirements Specification document.

1.2 Scope

The plan contains a list and brief description of the use cases to be tested and the software components associated with each test case. The plan also provides a schedule for the testing and the assignment of team members to their respective testing tasks. The process for documenting resolving software errors and/or anomalies that are found during the testing is also specified. The test specification includes a list of the features to be tested for each of the use cases, the description each test case needed to fully test the use case, and the test procedures, or steps, necessary to execute each of the test cases.

1.3 Abbreviations, Acronyms, and Definitions

TERM	DEFINITION
Applications Program Interface (API)	Implemented declarations of how a software component will interact with other software components. A common example of an API is a web service that provides data via a collection of resource addresses.
Microsoft Internet Explorer	A Microsoft Windows web browser.
End-to End Testing	Tests user scenarios and various path conditions by verifying that the system runs and performs tasks accurately with the same of data from beginning to end, as intended.
Google Chrome	A multi-platform web browser.
Graphical User Interface (GUI)	The visible, 'tactile' interface of a software system, usually a mouse- or touch-based system.
Mozilla Firefox	An open-source, multi-platform web browser.
N/A	Not Applicable
Quality Assurance (QA)	A set of methods for monitoring the software development process to ensure quality deliverables.
Requirements Traceability Matrix (RTM)	A series of rows and columns used to relate portions of a software engineering document to specific requirements.
Software feature	A distinguishing characteristic associated with a use case (e.g. its functionality, performance, ease of use, performance, etc.).

TERM	DEFINITION
Sprint.ly	A web application that helps organize software development tasks.
TBD	To Be Determined
Test case specification	A specification of inputs, expected results, and a set of execution steps associated with the testing of a feature (or features) associated with a use case.
Software problem report	A document reporting on any event that occurs during the testing process which requires investigation (see appendix A for a copy of the Software Problem Report form).
Software Design Specification (SDS)	Software Design Specification
Software Requirements Specification (SRS)	A software engineering document that establishes the requirements for the system under consideration.
Software Testing Specification (STS)	A software engineering document that establishes a baseline plan for testing a system.
System test report	A document summarizing testing activities and results. It also contains an evaluation of the degree to which the software product satisfies to the system requirements for each of the use cases.
Test log	A chronological record of relevant details about the execution of tests.
Web browser	An application for making HTTP requests and handling HTTP responses by rendering web pages and executing their included scripts.

Table 1.3 Abbreviations, Acronyms, and Definitions

1.4 References

1. Software Requirements Specification. Chambers, A., Cristiano, A., Doan, J., Green, D., and Rosato, J., Team Wakati, Sacramento, CA, May 1, 2014.
2. Software Design Specification. Chambers, A., Cristiano, A., Doan, J., Green, D., and Rosato, J., Team Wakati, Sacramento, CA, Oct. 17, 2014.
3. “AngularJS API Docs”, [online], AngularJS, https://docs.angularjs.org/api/ngMock/service/$httpBackend, [Oct. 25, 2014].
4. "Nock", Teixeira, P. [online], https://github.com/pgte/nock, [Oct. 24, 2014].
5. “Open Eligibility Project”, [online], http://openeligibility.org/, [April 14, 2014]

1.5 Overview of Contents of Document

Test Plan Description

This section provides a summary of the Use Cases and the plan for carrying out the system test phase of the team’s software development process. More specifically, this section contains a brief description of each Use Case to be tested, the team member (or members) assigned to test each Use Case, the testing schedule, and the risk management plan.

Test Design Specification

This section describes the details of the test approach, lists the use cases that are and are not to be tested, lists the environmental needs, and details the pass/fail and suspension/resumption criteria.

Test Specification

This section contains subsections for each of the features to be tested. Each subsection specifies the use cases to be tested, the procedures necessary to run the test cases, and the items being tested.

Requirements Traceability

This section provides for a cross referencing of each use case to its test specification and also to its design components. The appropriate section and its title in each document are provided.

Approvals

This section contains the list of the key signatories necessary to sign-off on the STS, thereby agreeing to the scope and content of the test plan and test cases specified within the document. Approval constitutes a guarantee that the development team has produced a test specification sufficient for validating the software to be delivered to the sponsor.

2. TEST PLAN DESCRIPTION

The Benefisher application is made up of many features that allow users to search for public services. Each feature is made up of a number of use cases, and each use case can be tested in a number of ways. This section details each of the features, their respective use cases, how Team Wakati plans to test each use case, who will be responsible for testing specific use cases, the planned schedule, and the risk plan in case certain use cases are not tested properly.

2.1 Product Summary

Benefisher is a public services search application that makes it easy for users to find the best services related to their needs. Benefisher is a single-page application designed for ease-of-use. That simplicity belies a fairly complex interconnection of components.

Features	Use Cases	Components			
		View	Client	Server	Database
Search	Search for Service Search by Need Search by Situation Search by Need & Situation Browse Select Pre-Defined Need Select Pre-Defined Situation Remove from Screen Save Search Query	layout.jade index.jade scripts.jade map.jade notification.jade	app.js services.js map-controller.js search-service.js interaction-service.js notification-service.js	app.js index.js search.js neuralnet.js interactions.js	search_result search_query
Map	Search for Service Browse Map Interact with Results Remove from Screen Click Result on Map	layout.jade index.jade scripts.jade search.jade	app.js services.js search-controller.js search-service.js interaction-service.js	app.js index.js search.js neuralnet.js	search_result search_query search_result_interaction service
Results	Search for Service Browse Save Interaction Data Interact with Results Expand Results Get Directions Call Service Navigate to Site Email Service Remove from Screen Click Result on Map	layout.jade index.jade scripts.jade results.jade notifcation.jade	app.js services.js results-controller.js search-service.js interaction-service.js notification-service.js	app.js index.js search.js interactions.js	search_result search_result_interaction

Table 2.1 Feature-Use Case-Component Matrix

2.2 Responsibilities

The roles and responsibilities of each team member during the testing process are listed in the table below. These roles are intended to be fluid, and each member may assume various roles as needed during testing.

Name	Role	Responsibilities
Adrian Chambers	Test Lead	Oversee and coordinate test process.
Anthony Cristiano	Recorder	Record test results.
James Doan	Tester	Execute test procedures and augment automated tests.
Daniel Green	Tester	Execute test procedures and augment automated tests.
Jesse Rosato	Tester	Execute test procedures and augment automated tests.

Table 2.2 Testing Roles and Responsibilities

2.3 Schedule

The schedule for testing the application is as follows:

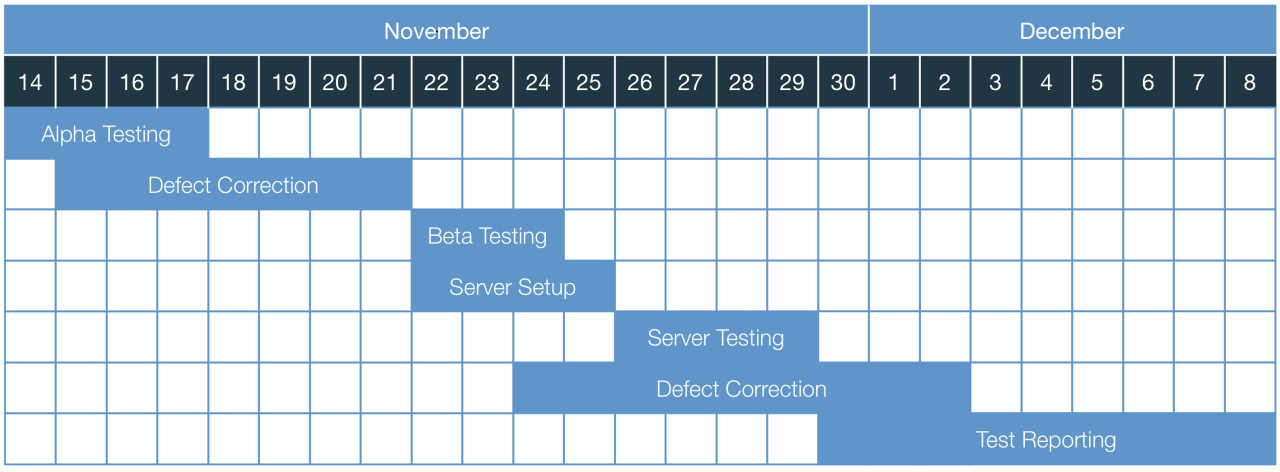

Figure 2.1 Testing Schedule

Activity	Description	Exit Criteria
Alpha Testing	All tests are executed and recorded according to specified test procedures.	All tests have been executed and reported.
Alpha Defect Correction	Correct critical defects, and as many non-critical defect as time allows. Add remaining non-crtical defects to the project's backlog.	All critical defects have been corrected, and all non-critical defects have been corrected or the time limit for this activity has passed.
Beta Testing	All tests are executed and recorded according to specified test procedures.	All tests have been executed and reported.
Server Testing	The application is installed on a staging server, and all tests are executed and recorded according to specified test procedures.	All tests have been executed and reported.
Beta Defect Correction	Correct critical defects, and as many non-critical defect as time allows. Add remaining non-crtical defects to the project's backlog.	All critical defects have been corrected, and all non-critical defects have been corrected or the time limit for this activity has passed.
Test Reporting	Use the results of testing to generate the test report.	The test report is complete.

Table 2.3 Testing Activities

3. TEST DESIGN SPECIFICATION

This section describes the details of the test approach, lists the use cases that are and are not to be tested, the environmental needs, and details the pass/fail and suspension/resumption criteria that Team Wakati is using to ensure the quality of Benefisher.

3.1 Testing Approach

Team Wakati used the SRS [1] and the SDS [2] to prepare the design test cases and their procedures. These tests are designed to verify the documentation previously listed.

To test Benefisher, Team Wakati is accounting for the following testing approaches:

Conversion testing

Benefisher will be ingesting data from the Ohana API and must test that this information is being manipulated in the correct way for use within the application.

Interface testing

Benefisher has many modules and components, including, but not limited to, search, map, and results. Information from these components must be passed between each other and to other components throughout the application. Interface testing will be used to evaluate whether these components and modules pass data and control correctly to one another.

Regression testing

Benefisher is being developed in an iterative fashion. After each build/code change, Team Wakati's environment is set up to run all test cases against the current code base with each new push to the main repository [1].

Coverage testing

As per the SRS [1], Team Wakati will be writing their own unit tests on each new code push. Because of this requirement, in addition to other testing methods, Team Wakati aims to have near 100% code coverage. This means that 100% (or extremely close to 100%) of the source code written passes through a test at some point in the testing suite.

Black box testing

Benefisher is an interactive frontward-facing web application. Due to the nature of this application, Team Wakati will be conducting manual testing on the webpage to ensure that the application reacts and responds in the way that it is expected to.

White box testing

Team Wakati is utilizing unit and integration testing. All of the unit and integration test cases are constructed with detailed knowledge of the code base, and have been automated to run with each code base change.

3.2 Feature or Combination of Features Not To Be Tested

The following list describes features that will be excluded during the testing phase

- Load Testing - Ability to cope with volume, load and hardware faults. Test cases will not include testing the systems ability to deal with multiple yesers and any hardware issues that may arise.
- Session Time Out - Time related bugs such as session time out will not be tested. It is assumed that once a user's session times out the session variables are lost and the user must start over
- Browser Cookies - The use of browser cookies will not be tested
- Server Reliability - Server uptime and reliability will not be tested

3.3 Environmental Needs

This subsection contains the properties that are needed to test the Benefisher application. An environment matching the following criteria can be used in tests, and will later be referred to as *Acceptance Server*.

Client

- The environment will have a network connection. Preferably the environment will have a wireless connection to simulate the network environment under which a typical user will access the Benefisher application.
- The environment will be able to run the web browser specified by a given test.
- That web browser will have JavaScript enabled.

Server

- The environment will access a test application database, used for storing application usage statistics.
- The environment will access a demo version of the Code for Sacramento data source.

3.4 Suspension / Resumption Criteria

Any test suspended before completion will be abandoned and resumed from the first step of the test. Any preconditions for that test will be reestablished before the test is resumed. If a test cannot be completed due to continue application failure, that will be noted in the report for that test case.

3.5 Risks and contingencies

This subsection contains a list of the possible risks that are most likely to affect the testing schedule and the ability to deliver the software according to schedule.

Schedule Overrun

If the testing process cannot proceed according to the schedule established in this document, the Server Testing phase will be postponed.

Test Environment Shortfall

If the necessary devices or software specified in test cases are not available at the time of testing, efforts will be made to procure them. Any test environments that cannot be procured will be noted in the relevant test reports.

Server Test Environment Delays

The setup of the server test environment poses a significant potential to delay the testing schedule. If this occurs, an alternate server setup may be used, or the Server Testing phase may be postponed. If an alternative server testing environment is used, this will be noted in the test report.

4. TEST SPECIFICATION

This section contains subsections for each of the features to be tested. Each subsection specifies the USE CASES to be tested, the procedures necessary to run the test cases, and the items being tested. Use Cases spanning multiple features will be listed multiple times, but specified only in their first appearance. Each Use Case includes automated tests. Figures 4.1 and 4.2 depict the sample output of running these automated tests. The following table (4.1) details the procedures for executing these automated tests automated tests. This procedure is executed for each Use Case:

Automated Test Procedure		
Step	Description	Expected Result
1	Execute testing suite by running 'grunt test' via command line.	All tests pass.
2	Analyze test output. File a sprint.ly bug report for any failing tests.	Any defects are added to sprint.ly backlog.

Table 4.1 Automated Test Procedure

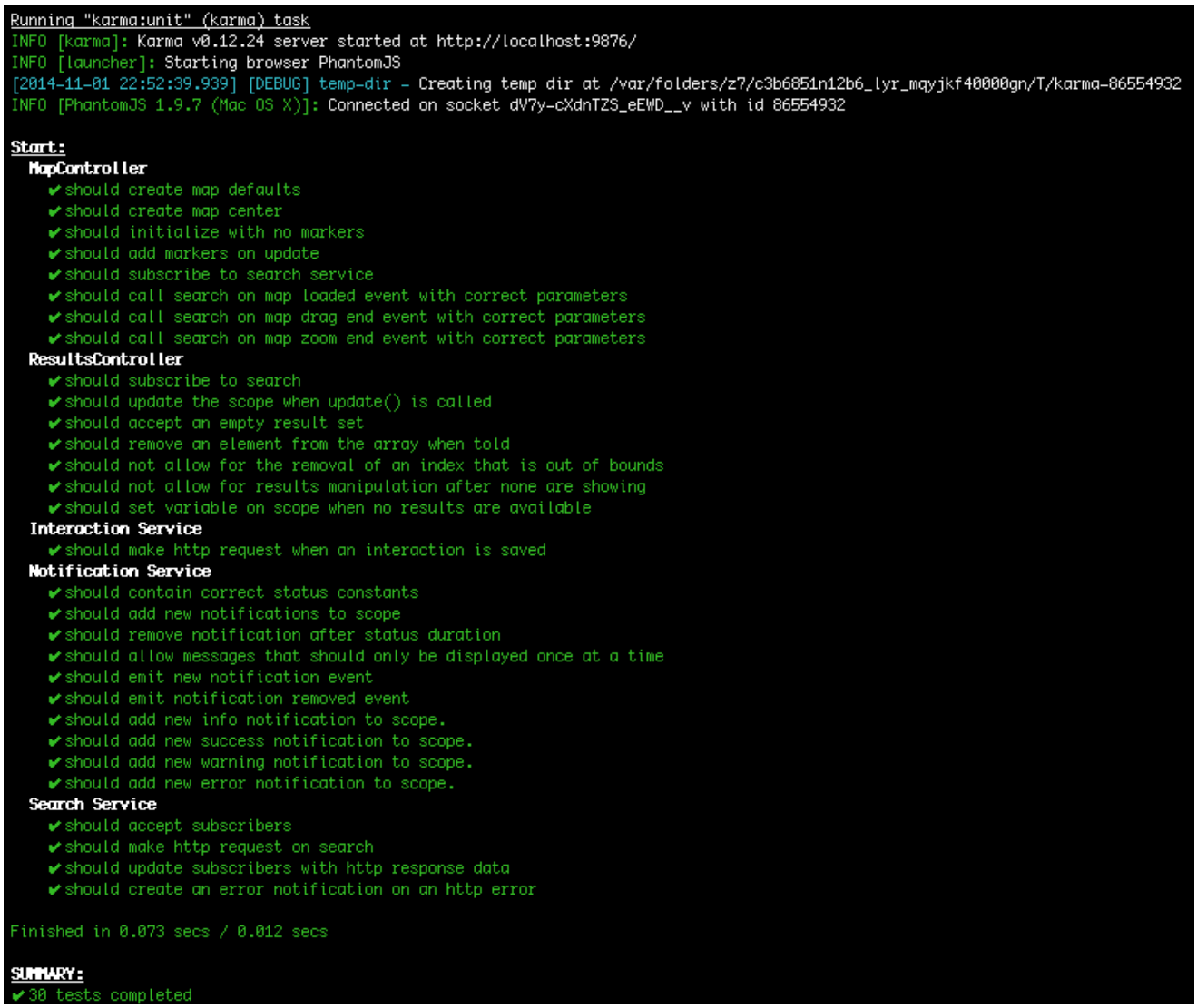

Figure 4.1 Sample Screen Print of Successful Automated Server Test Results

Figure 4.2 Sample Screen Print of Successful Automated Client Test Results

4.1 Search

The Search feature manages user interactions with the application's search inputs, along with the end-to-end process of retrieving results from the Code for Sacramento data source, sorting results in order of relevance, and providing results to other components for display and interaction. This functionality presents several challenges in testing:

- On the client-side, the SearchService component (*search-service.js*) makes HTTP requests to the server. During automated client unit testing, HTTP requests must be intercepted, and their responses must be mocked. This is achieved using Angular's *$httpBackend* service [3].
- On the server-side, the SearchController (*search.js*) component makes another HTTP request to the Code for Sacramento data source. This HTTP request must be intercepted and its response must be mocked during automated server integration tests. This is achieved using the N*ock* library [4].
- The SearchController also includes several database interactions. In automated server unit tests, the database models are mocked, and injected into the SearchController class. In automated server integration tests, a clean instance of a test database is prepared for each test.

Search for Service (UC1)

The Search for Service use case is responsible for retrieving search results from the Code for Sacramento data source and returning them to the user's client.

Automated Testing

Test Case ID:	STS-1
Test Name:	Search for Service (Automated)
Description:	Ensure that components pass unit and integration testing, and perform basic operations as expected.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Conversion, Regression, White Box

Table 4.2 Test Case Description *STS-1*

Automated Test Description (STS-1)		
Test	Description	Expected Result
SearchController (Server Unit Tests)		
1	It should make an http request (all results found in DB path)	Request endpoint is called and results are found.
2	It should make an http request (no results found in DB path)	Request endpoint is called and results are not found.
3	It should render results on http success (*all results found in DB* path)	A JSON array of results is rendered in an HTTP response.
4	It should render results on http success (*no results found in DB* path)	A JSON array of results is rendered in an HTTP response.
5	It should limit search results by lat/long bounds	The results found are within the given latitude and longitude boundaries.
6	It should respond with 500 when http request fails	An HTTP repsonse with status 500 is generated.
SearchController (Server Integration Tests)		
7	It should respond with JSON	When a search request is made, the response should use the *application/json* content type.
8	It should respond with 500 when http request fails	When the request to the external data source fails, the server should respond with status 500.
SearchService (Client Unit Tests)		
9	It should make HTTP request on search	An HTTP request to the Code for Sacramento data source is made
10	It should update subscribers with http response data.	Subscribers are updated with an array of results.
11	It should create an error notification on an http error	An error notification is created.

Table 4.3 Automated Test Description *STS-1*

Manual Testing

Test Case ID:	STS-2
Test Name:	Search for Service (Manual)
Description:	Ensure that the various terms/inputs return the expected result(s), each denoted on the map with a marker and the results pane with an entry containing more information.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Conversion, Black Box

Table 4.4 Test Case Description *STS-2*

Manual Test Procedure (STS-2)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Type "Clothing" in the What do I need? search input bar.	An autocomplete dropdown is displayed below the "What's my Need?" search input. All terms except those containing the word 'Food' are filtered from the autocomplete dropdown.
3	Click the first term or hit the enter key.	A tag with the first term from the list is added to the left of the "What's my situation?" search input, and relevant results are displayed in the results pane and on the map.
4	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.5 Manual Test Procedure *STS-2*

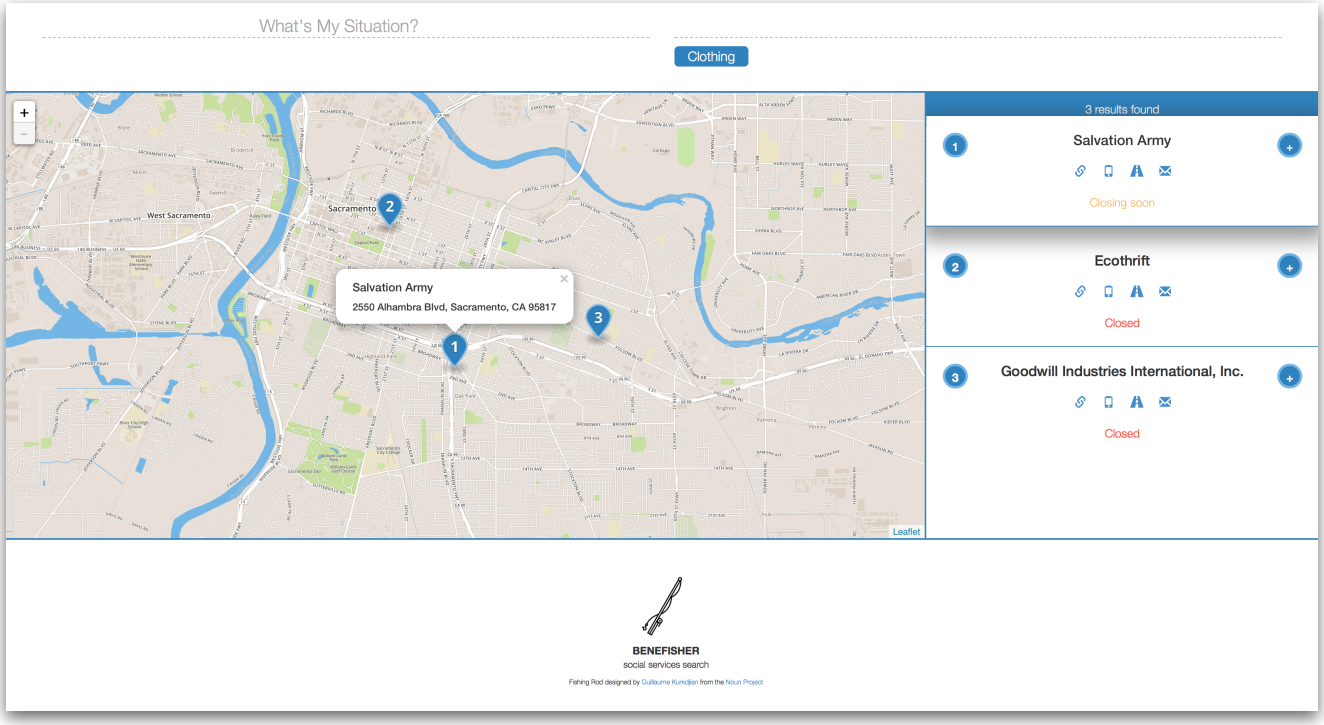

Figure 4.3 Search By Need Tag

Search by Need (UC2)

Search by Need is a specific means of searching for services (UC1). The user searches for services for people with a particular 'need', as defined by OEP terms [5]. The automated tests for this Use Case are covered by the tests for UC1.

Search by Situation (UC3)

Search by Situation is a specific means of searching for services (UC1). The user searches for services for people in a particular 'situation', as defined by OEP terms [5].

Automated Testing

The automated tests for this Use Case are covered by the tests for UC1.

Manual Testing

Test Case ID:	STS-3
Test Name:	Search by Situation (Manual)
Description:	Ensure that various terms/inputs return the expected result, each denoted on the map with a marker and the results pane with an entry containing more information.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Conversion, Interface, Black Box

Table 4.6 Test Case Description *STS-3*

Manual Test Procedure (STS-3)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Type "Homeless" in the "What's my situation?" search input.	An autocomplete dropdown is displayed below the "What's my Situation?" search input. All terms except those containing the word 'Veteran' are filtered from the autocomplete dropdown.
3	Click the first term or hit the enter key.	A tag with the first term from the list is added to the left of the "What's my situation?" search input, and relevant results are displayed in the results pane and on the map.
4	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.7 Manual Test Procedure *STS-3*

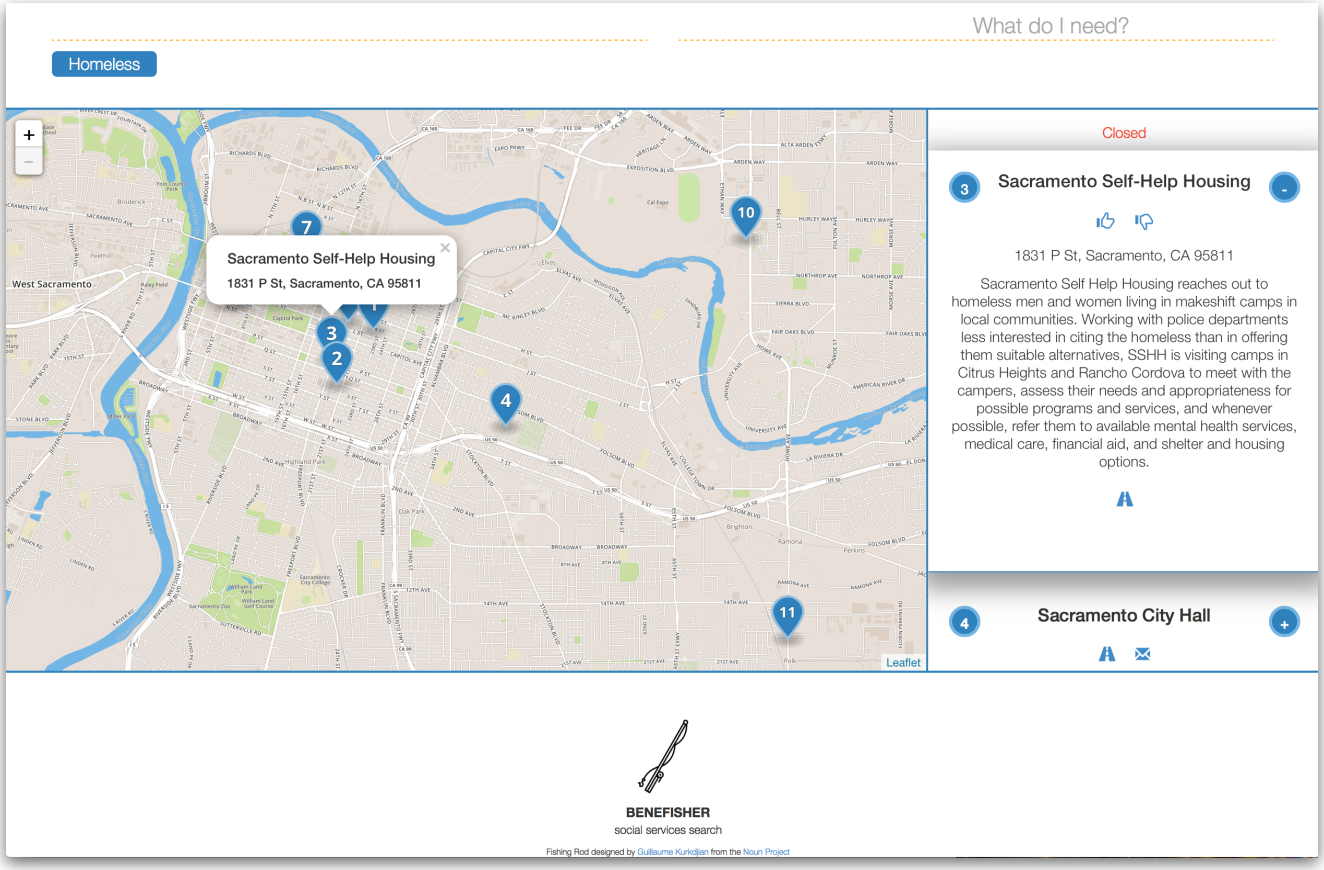

Figure 4.4 Search By Situation Tag

Search by Need & Situation (UC4)

Search by Need & Situation is a specific means of searching for services (UC1). The user searches for services for people in a particular 'situation' and with a particular 'need', as defined by OEP terms [5].

Automated Testing

The automated tests for this Use Case are covered by the tests for UC1.

Manual Testing

Test Case ID:	STS-4
Test Name:	Search by Need & Situation (Manual)
Description:	Ensure that various terms/inputs return the expected result, each denoted on the map with a marker and the results pane with an entry containing more information.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Conversion, Interface, Black Box

Table 4.8 Test Case Description *STS-4*

Manual Test Procedure (STS-4)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Type "Mothers" in the What is my situation? search input bar.	An autocomplete dropdown is displayed below the "What's my Situation?" search input. All terms except those containing the word 'Veteran' are filtered from the autocomplete dropdown.
3	Click the first term or hit the enter key.	A tag with the first term from the list is added to the left of the "What's my situation?" search input. Relevant results are displayed in the results pane and on the map.
4	Type "Daytime Care" in the What do I need? search input bar.	An autocomplete dropdown is displayed below the "What's my Need?" search input. All terms except those containing the word 'Food' are filtered from the autocomplete dropdown.
5	Click the first term or hits the enter key.	A tag with the first term from the list is added to the left of the "What's my need?" search input. The results displayed in the results pane and on the map are narrowed to those only relevant to both the need and situation terms entered.
6	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.9 Manual Test Procedure *STS-4*

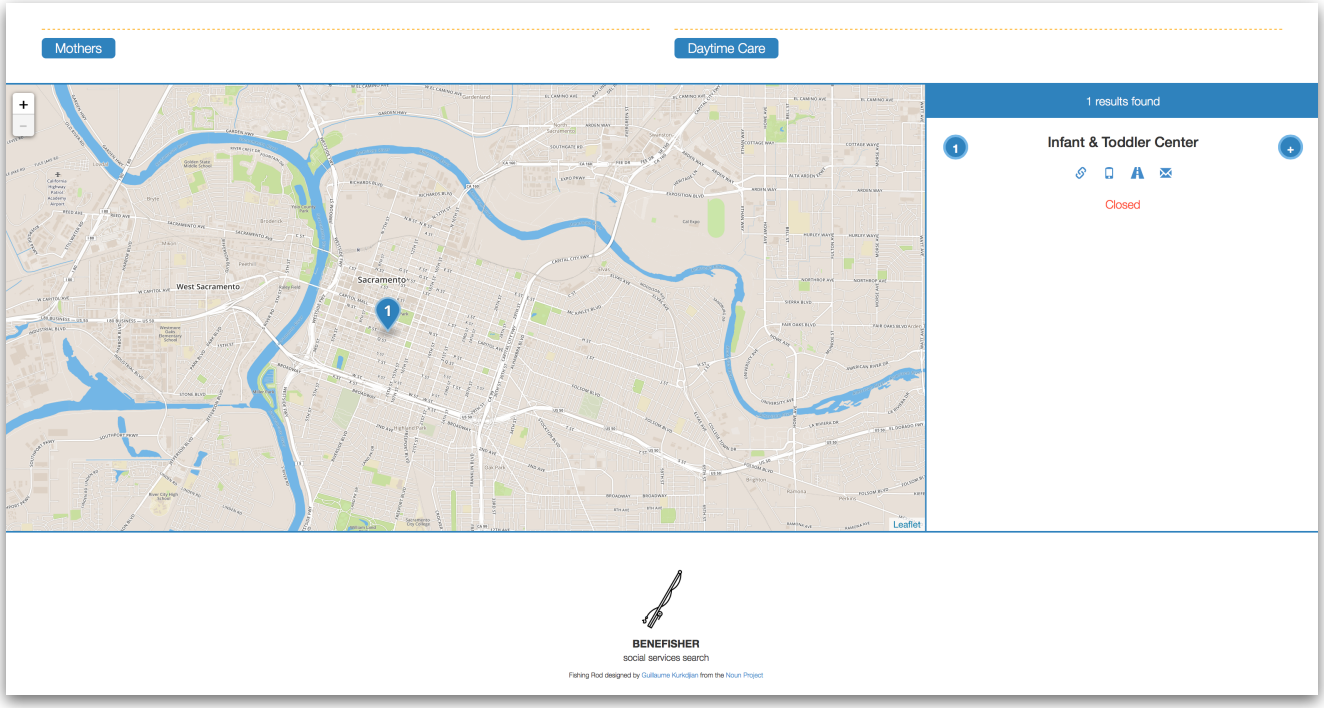

Figure 4.5 Search By Need and Situation Tags

Browse Map (UC5)

This Use Case provides the primary means of user interaction with the application. The user can pan and zoom the map in order to find needed services in a specific geographic location.

Automated Testing

Test Case ID:	STS-5
Test Name:	Browse Map (Automated)
Description:	The map should be created, centered and have markers reflecting the test data that is fed to it.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Conversion, White Box

Table 4.10 Test Case Description *STS-5*

Automated Test Description (STS-5)		
Test	Description	Expected Result
MapController (Client Unit Tests)		
1	It should call search on map loaded event with correct parameters.	Map results should appear in their correct position as soon as the map is completely loaded.
2	It should call search on map drag end event with correct parameters.	Map results should appear in their correct position after a user finishes panning the map.
3	It should call search on map zoom end event with correct parameters.	Map results should appear in their correct position after a user finishes zooming in/out the map.

Table 4.11 Automated Test Description *STS-5*

Manual Testing

Test Case ID:	STS-6
Test Name:	Browse Map (Manual)
Description:	Ensure that any manipulations done to the map while browsing (including dragging and zooming) perform the proper updates to the map element.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Conversion, Interface, Black Box

Table 4.12 Test Case Description *STS-6*

Manual Test Procedure (STS-6)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Wait for map to get populated with results	Map is populated with results in the area.
3	Move the map by dragging it.	The map's results are updated to encompass the new boundaries of the map.
4	Zoom into the map by scrolling up.	The map's results are updated to encompass the new boundaries of the map.
5	Zoom out of the map by scrolling down.	The map's results are updated to encompass the new boundaries of the map.
6	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.13 Manual Test Procedure *STS-6*

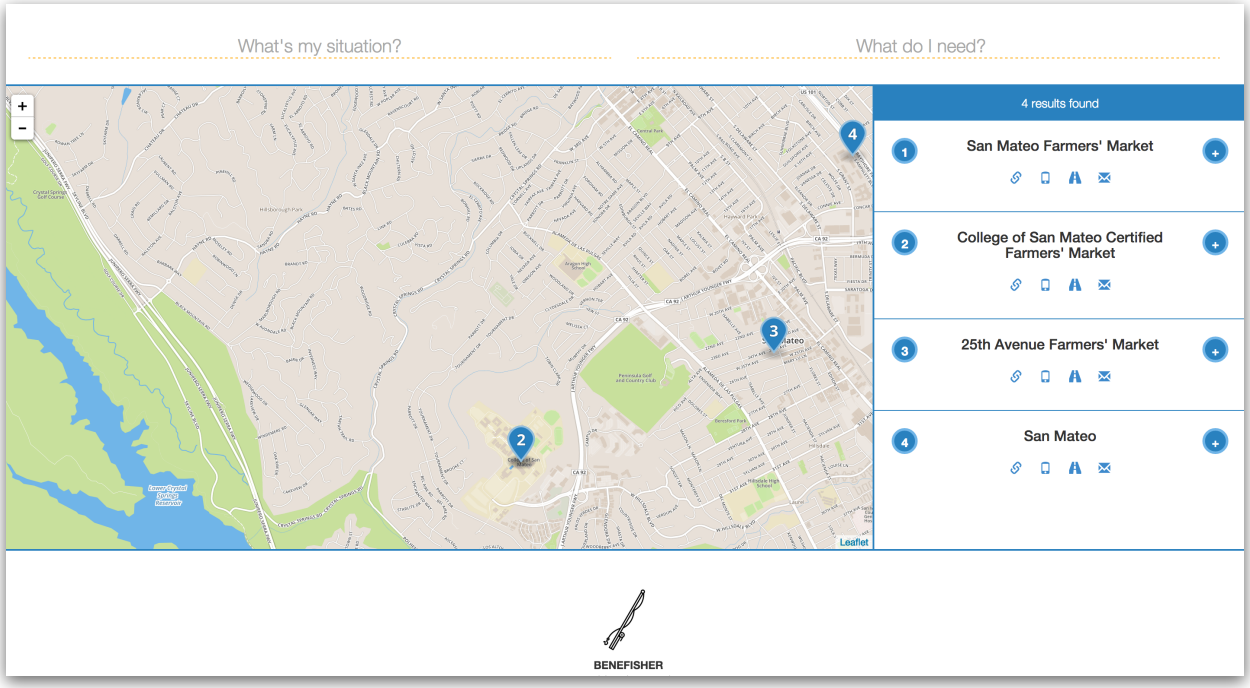

Figure 4.6 Map Zoomed Out

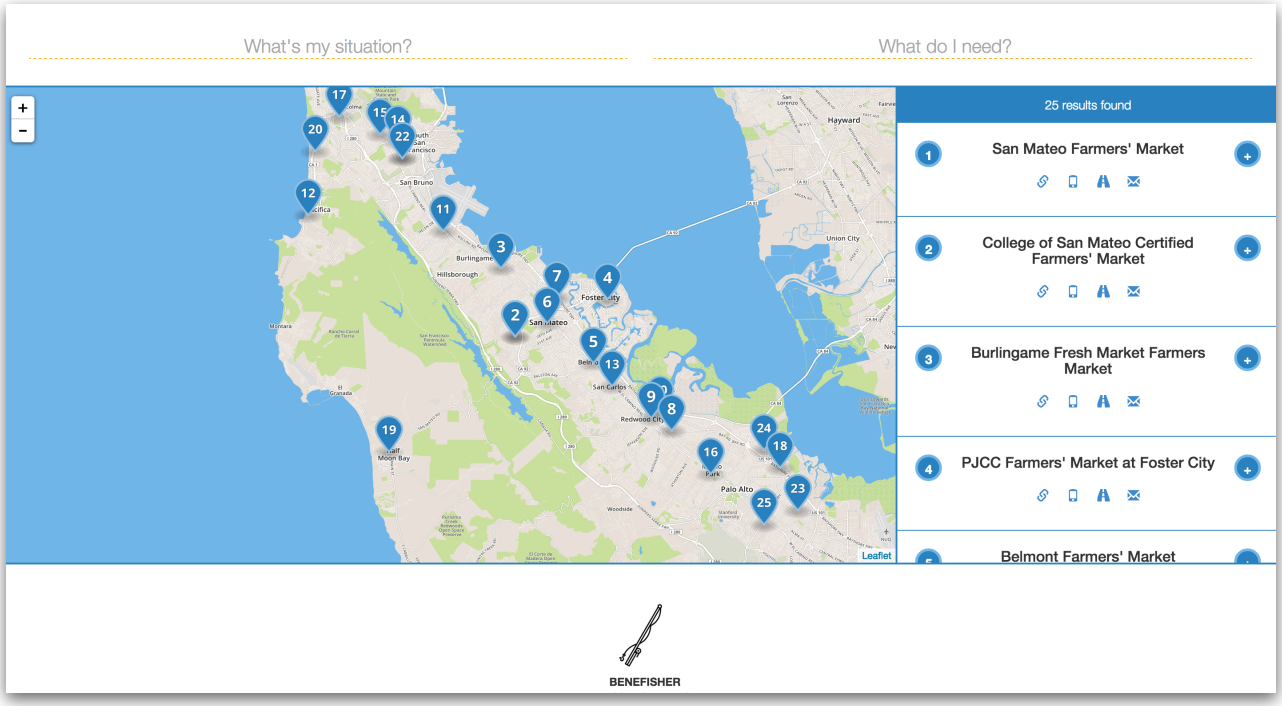

Figure 4.7 Map Zoomed In

Select Predefined Need (UC6)

This Use Case provides an 'autocomplete' input for users to select one or more predefined 'needs' from a list of OEP terms [5].

Automated Testing

The automated tests for this Use Case are covered by *STS-1*.

Manual Testing

Test Case ID:	STS-7
Test Name:	Select Predefined Need (Manual)
Description:	Test 'need' text input autocomplete and tagging features.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Conversion, Interface, Black Box

Table 4.14 Test Case Description *STS-7*

Manual Test Procedure (STS-7)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Type "Veteran" in the "What's my situation?" search input.	An autocomplete dropdown is displayed below the "What's my Situation?" search input. All terms except those containing the word 'Veteran' are filtered from the autocomplete dropdown.
3	Click the first term or hit the enter key.	A tag with the first term from the list is added to the left of the "What's my situation?" search input.
4	Click the 'close' icon in the tag created in Step 3.	The tag is removed.
5	Enter "Xyz" in the needs box	A dropdown displaying no matching OEP terms is shown with a message telling the user to revise their search. Suggested terms are shown.
6	Enter 123	A dropdown displaying no matching OEP terms is shown with a message telling the user to revise their search. Suggested terms are shown.
7	Enter \enter (blank input)	An autocomplete dropdown displaying all OEP terms is displayed.
8	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.15 Manual Test Procedure *STS-7*

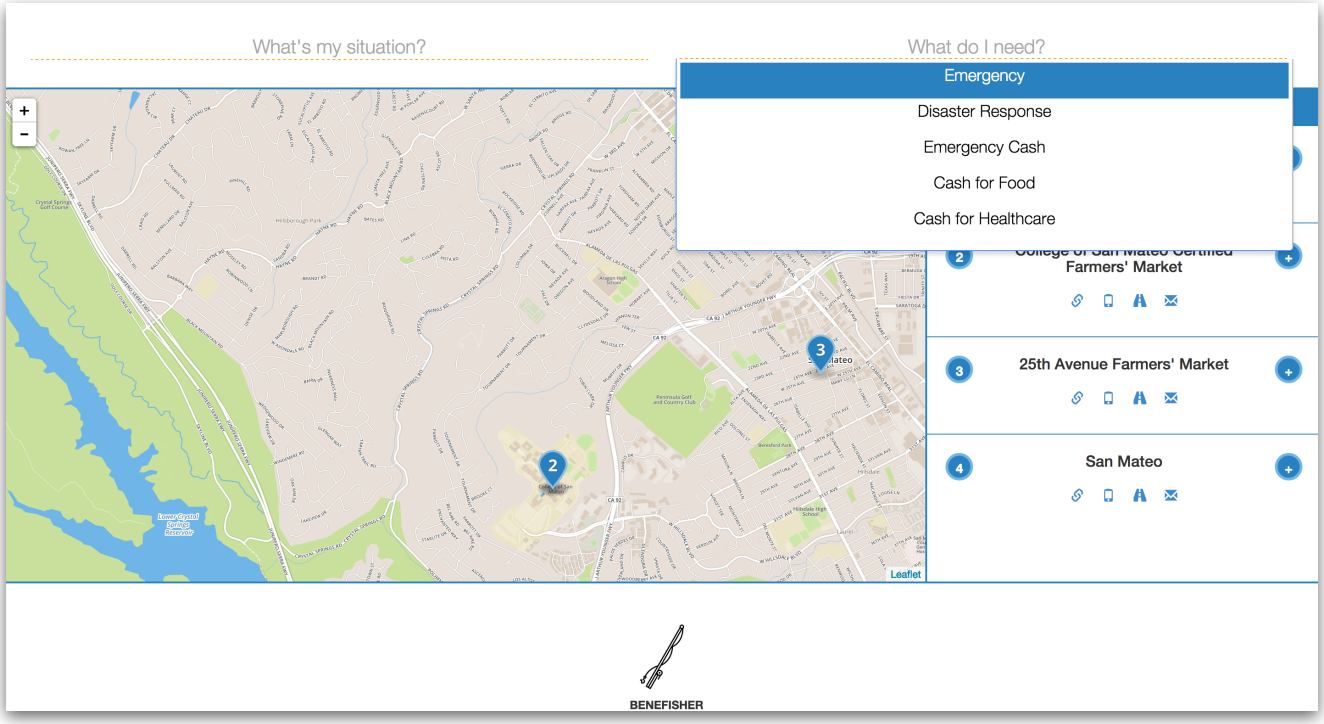

Figure 4.8 OEP Terms Dropdown

Select Predefined Situation (UC7)

This Use Case provides an 'autocomplete' input for users to select one or more predefined 'situations' from a list of OEP terms [5].

Automated Testing

The automated tests for this Use Case are covered by *STS-1*.

Manual Testing

Test Case ID:	STS-8
Test Name:	Select Predefined Situation (Manual)
Description:	Test 'situation' text input autocomplete and tagging features.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Conversion, Interface, Black Box

Table 4.16 Test Case Description *STS-8*

Manual Test Procedure (STS-8)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Type "Veteran" in the "What's my situation?" search input.	An autocomplete dropdown is displayed below the "What's my Situation?" search input. All terms except those containing the word 'Veteran' are filtered from the autocomplete dropdown.
3	Click the first term or hit the enter key.	A tag with the first term from the list is added to the left of the "What's my situation?" search input.
4	Click the 'close' icon in the tag created in Step 3.	The tag is removed.
5	Enter "Xyz" in the needs box	A dropdown displaying no matching OEP terms is shown with a message telling the user to revise their search. Suggested terms are shown.
6	Enter 123	A dropdown displaying no matching OEP terms is shown with a message telling the user to revise their search. Suggested terms are shown.
7	Enter \enter (blank input)	An autocomplete dropdown displaying all OEP terms is displayed.
8	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.17 Manual Test Procedure *STS-8*

Remove from Screen (UC15)

Users can choose to ignore results that don't interest them.

Test Case ID:	STS-9
Test Name:	Remove from Screen (Automated)
Description:	Results should be removed from the list of available results, and should be ignored on future searches.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Coverage, White Box

Table 4.18 Test Case Description *STS-9*

Automated Test Description (STS-9)		
Test	Description	Expected Result
ResultsController (Client Unit Tests)		
1	It should remove the result at the given index.	The search service's 'remove' method is called.
2	It should not remove a result when the given index is out of bounds.	The search service's 'remove' method is not called.
3	It should not remove a result when no results are available.	The search service's 'remove' method is not called.
SearchService (Client Unit Tests)		
4	It should mark a given result as ignored.	The given result should be marked as ignored when the service updates subscribers.

Table 4.19 Automated Test Description *STS-9*

Manual Testing

Test Case ID:	STS-10
Test Name:	Remove from Screen (Manual)
Description:	Test removing a search result.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Conversion, Interface, Black Box

Table 4.20 Test Case Description *STS-10*

Manual Test Procedure (STS-10)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Click the 'thumbs down' button on the first result in the results pane.	The result should be removed from the results pane and the map, and the result count in the results pane should be decremented.
3	Click the 'thumbs down' button on the new first result in the results pane.	The result should be removed from the results pane and the map, and the result count in the results pane should be decremented.
4	Click the 'thumbs down' button on the remaining results in the results pane.	All results should be removed from the results pane and the map, and the result count in the results pane should be zero.
5	Pan the map slightly to cause a new search.	None of the ignored results are displayed.
6	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.21 Manual Test Procedure *STS-10*

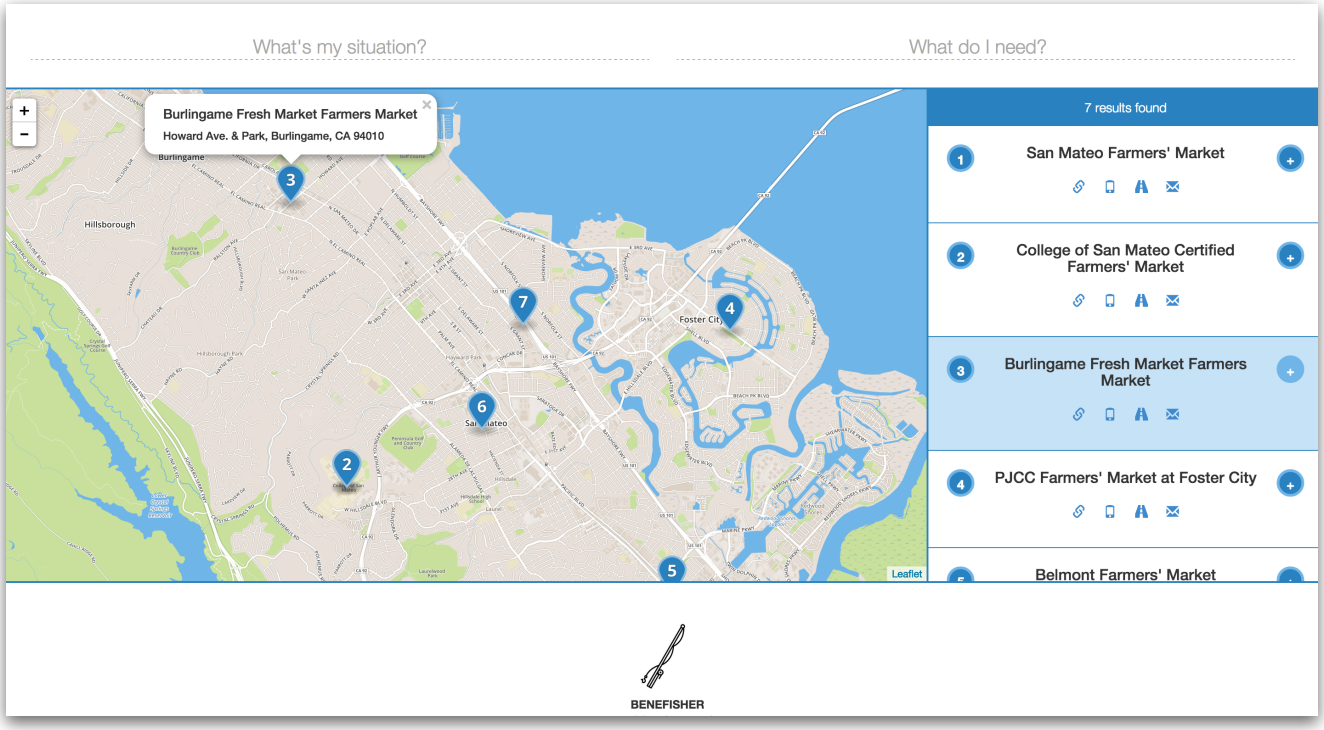

Figure 4.9 Before Removing Result

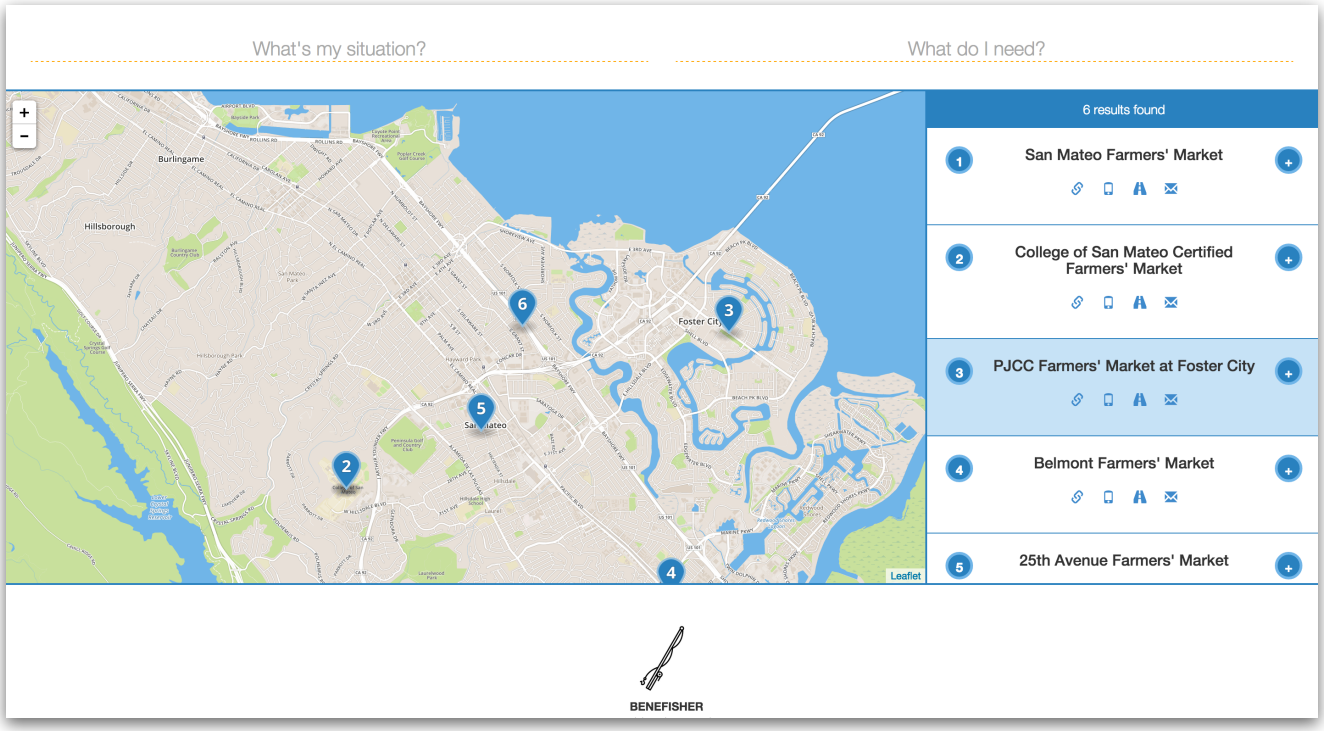

Figure 4.10 After Removing Result

Save Search Query (UC17)

Search queries are saved to the application's database for use by the application's neural network to deliver relevant results to users.

Automated Testing

Test Case ID:	STS-11
Test Name:	Save Search Query (Automated)
Description:	Search queries should be saved to the application database.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Coverage, White Box

Table 4.22 Test Case Description *STS-11*

Automated Test Description (STS-11)		
Test	Description	Expected Result
SearchController (Server Unit Tests)		
1	It should attempt to save query with results (all results found in DB path).	Search results are associated with the search query, and the method to save the query to the database is called.
2	It should attempt to save query with results (no results found in DB path)	Search results are associated with the search query, and the method to save the query to the database is called.

Table 4.23 Automated Test Description *STS-11*

Manual Testing

Test Case ID:	STS-12
Test Name:	Save Search Query (Manual)
Description:	Test that search queries are saved to the application database.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Conversion, Interface, Black Box

Table 4.24 Test Case Description *STS-12*

Manual Test Procedure (STS-12)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly. The initial search query is saved in the application's database with the bounds of the initial map position.
2	Enter a search term in the "What's my situation?" input.	A search query with the entered term, and the bounds of the initial map position, is saved in the application's database.
3	Enter a search term in the "What do I need?" input.	A search query with both entered terms, and the bounds of the initial map position, is saved in the application's database.
4	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.25 Manual Test Procedure *STS-12*

4.2 Map

The Map feature provides the main method for interacting with the application. As noted in the SRS [1], many of Benefisher's target users have limited access to transportation, so the location of services is very important to them. The Map feature is loosely coupled with the rest of the system, and provides a single unique testing challenge:

- The client-side MapController (*map-controller.js*) interacts with the user's browser's geolocation API to retrieve information about the user's physical location.This is achieved by creating a service that handles all interactions with the API, and then injecting a mock of the service into the controller during automated client unit testing.

Search for Service (UC1)

This Use Case is described in detail in the Search feature subsection (4.1).

Automated Testing

Test Case ID:	STS-13
Test Name:	Subscribe to Search Service (Automated)
Description:	Ensure that the map controller subscribes to the search service. The remaining functionality for this Use Case is covered by *STS-1*.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Interface, Coverage, Regression, White Box

Table 4.26 Test Case Description *STS-13*

Automated Test Description (STS-13)		
Test	Description	Expected Result
MapController (Client Unit Tests)		
1	It should subscribe to the search service.	Map is updated with results anytime a search is specified.

Table 4.27 Automated Test Description *STS-13*

Manual Testing

See test case STS-2 for manual testing of this Use Case, along with the description, in the Search section (4.1).

Browse Map (UC5)

This Use Case is described in detail in the Search section (4.1).

Automated Testing

Test Case ID:	STS-15
Test Name:	Update Map on Move (Automated)
Description:	Ensure that the map controller passes unit testing and performs basic operations as expected.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Interface, Coverage, Regression, White Box

Table 4.30 Test Case Description *STS-15*

Automated Test Description (STS-15)		
Test	Description	Expected Result
MapController (Client Unit Tests)		
1	It should create map defaults.	The generated map's configuration matches the configuration that we have sent it.
2	It should create map center.	The generated map's center matches the one that we've sent it.
3	It should initialize with no markers.	The map has 0 markers initially.
4	It should add markers on update.	Upon an update, the map gains *n* markers, where *n* is equal to the number of results sent in the update.
5	It should not add markers that have an ignored property of true.	If one of the markers is hidden (meaning that the user has hidden it), the marker is no longer be on the map
6	It should focus on the marker that was selected by the user.	When a marker is clicked, the marker gain focus from the application
7	It should call the search service's selected function when a marker has been clicked.	When a marker is clicked, the map triggers the *selected()* method in the search service.
8	It should subscribe to search service.	The map is subscribed to updated from the search service.
9	It should call search on map loaded event with correct parameters.	When the map is loaded, it is populated with results as per its request.
10	It should call search on map drag end event with correct parameters.	When the map is dragged, the results change accordingly.
11	It should call search on map zoom end event with correct parameters.	When the map is zoomed, the results change accordingly.

Table 4.31 Automated Test Description *STS-15*

Manual Testing

Test Case ID:	STS-16
Test Name:	Update Map on Move (Manual)
Description:	Ensure that panning and/or zooming the map updates the map with relevant results.
Prerequisites:	Testing for UC18 should pass.
Test Environment:	Acceptance Server
Test Strategy	Interface, Black Box

Table 4.32 Test Case Description *STS-16*

Manual Test Procedure (STS-16)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Drag the map to the east/west and then release the mouse.	Markers that are in the current view should appear, while markers outside the view should disappear.
3	Drag the map north/south and then release the mouse.	Markers that are in the current view should appear, while markers outside the view should disappear.
4	Zoom the map out using the mouse scroll wheel or trackpad scroll gesture and then release.	Markers that are in the current view should appear, while markers outside the view should disappear.
5	Zoom the map out by clicking the '-' button on the map.	Markers that are in the current view should appear, while markers outside the view should disappear.
6	Zoom the map in using the mouse scroll whell or trackpad scroll gesture and then release the mouse.	Markers that are in the current view should appear, while markers outside the view should disappear.
7	Zoom the map in by clicking the '+' button on the map.	Markers that are in the current view should appear, while markers outside the view should disappear.
8	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.33 Manual Test Procedure *STS-16*

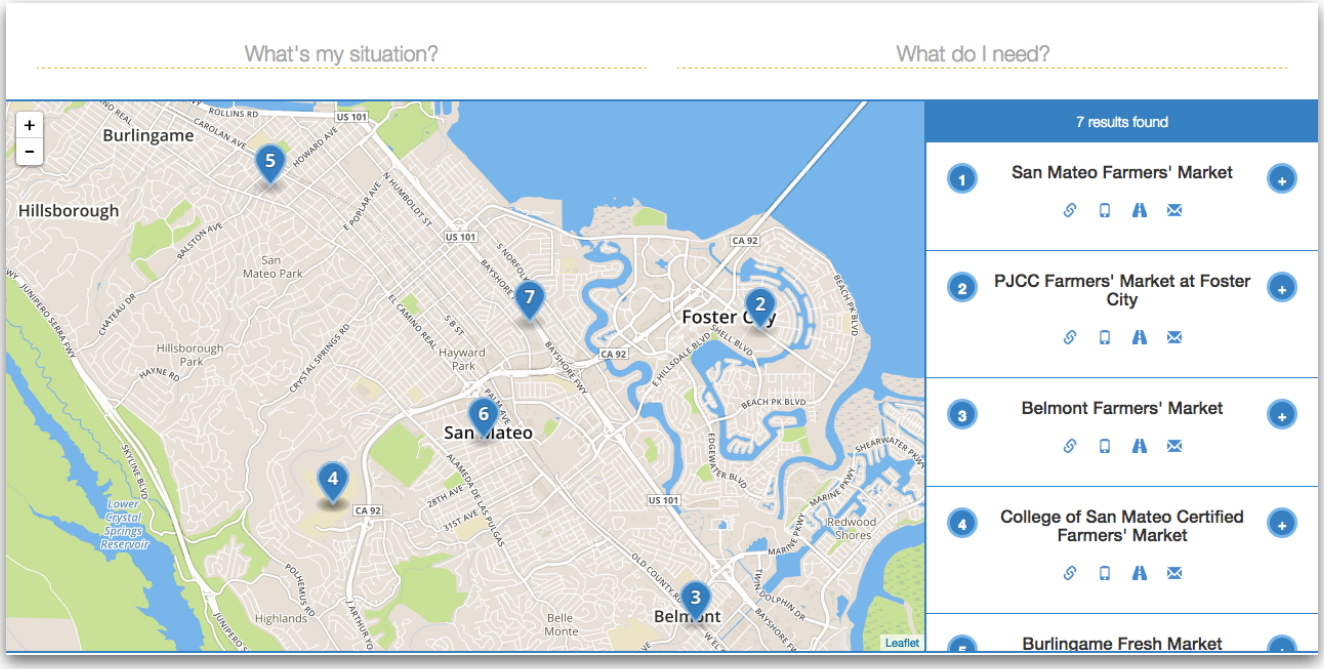

Figure 4.11 After opening the application

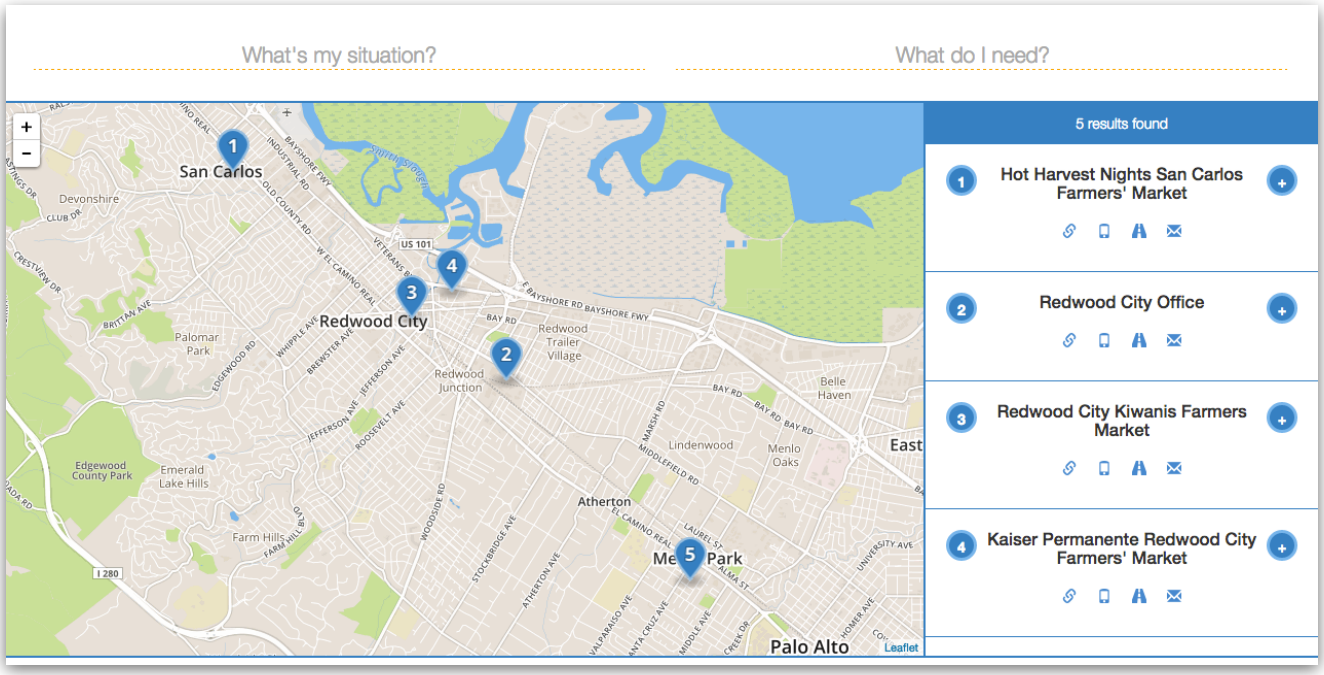

Figure 4.12 After Dragging map

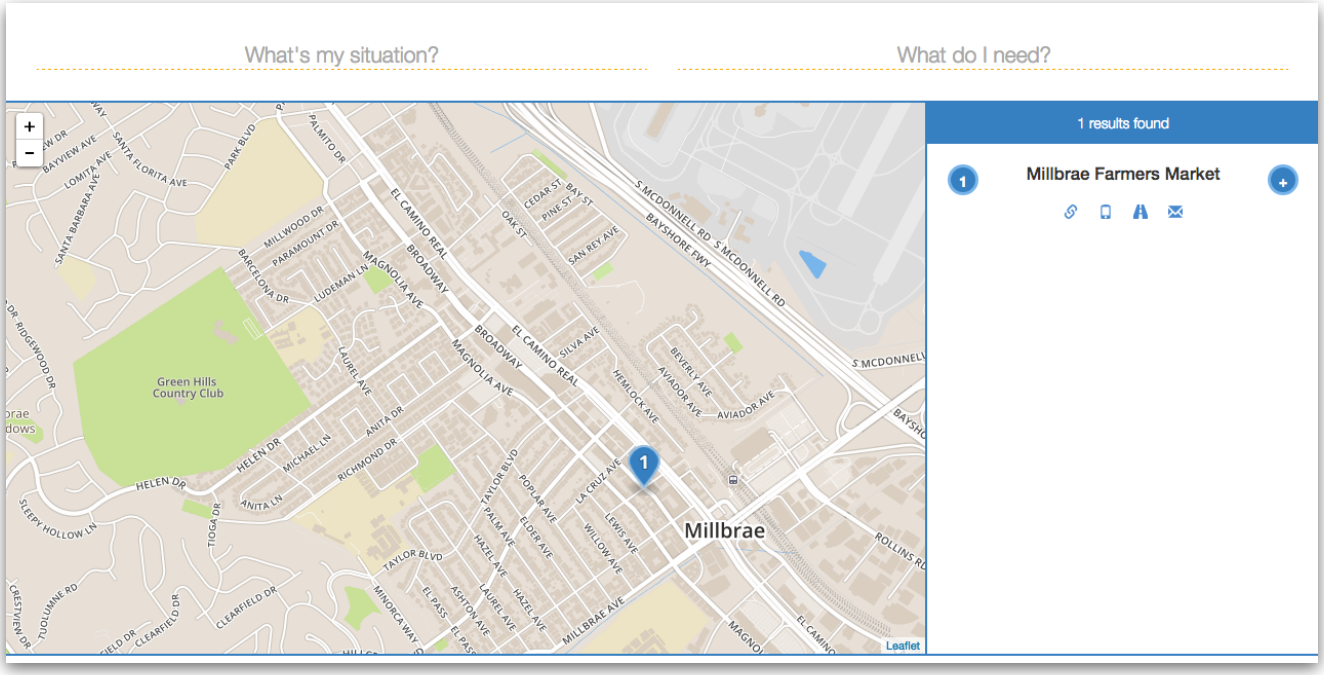

Figure 4.13 After zooming in

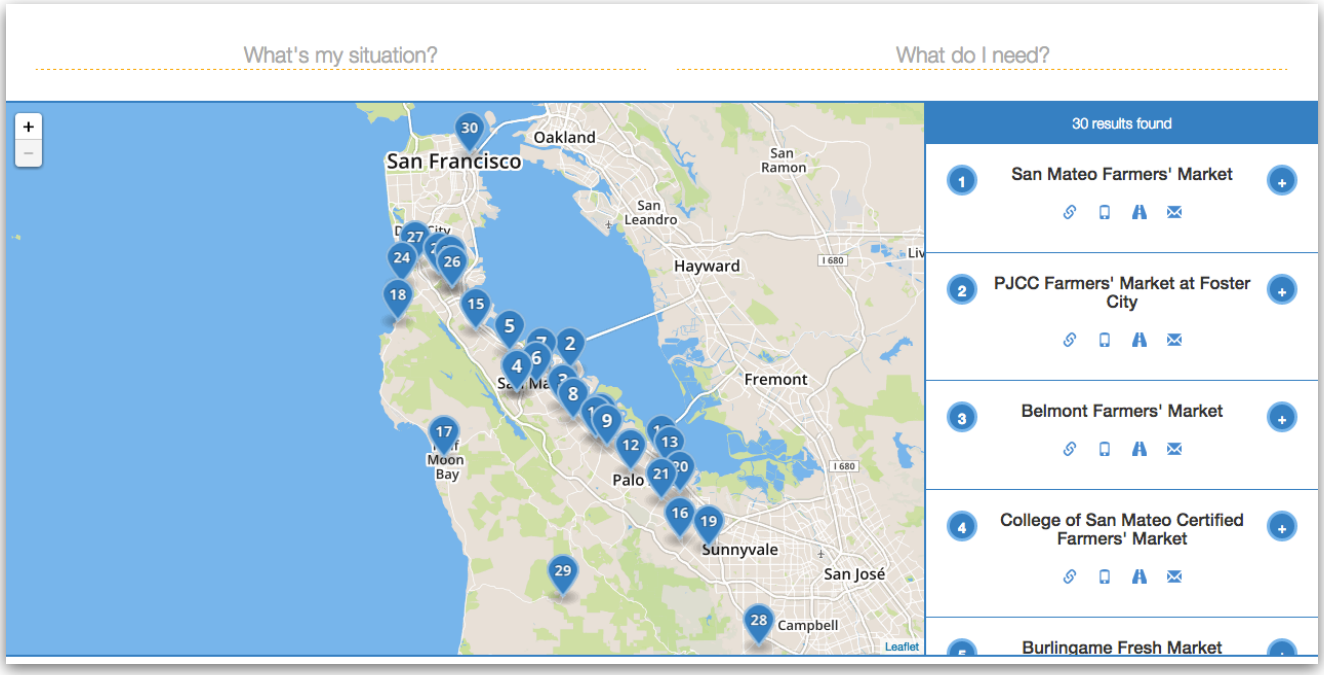

Figure 4.14 After Zooming Out

Interact with Results (UC9)

This Use Case is covered by tests STS-16, STS-17, and STS-20 thru STS-32.

Remove from Screen (UC15)

This Use Case is described in detail in the Search subsection (4.1).

Automated Testing

Test Case ID:	STS-17
Test Name:	Remove Marker from Map (Automated)
Description:	Ensures that the map controller does not add markers that are marked as ignored by the user.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Interface, Regression, Coverage, Black Box

Table 4.34 Test Case Description *STS-17*

Automated Test Description (STS-17)		
Test	Description	Expected Result
MapController (Client Unit Tests)		
1	It should not add markers that have an ignored property of true.	Map markers should not be added to the map if its result was downvoted.

Table 4.35 Automated Test Procedure *STS-17*

Manual Testing

Test Case ID:	STS-18
Test Name:	Remove Marker from Map (Manual)
Description:	Ensures that the map controller receives the event that a user chose to ignore a result and hides the corresponding marker.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Interface, Regression, Black Box

Table 4.36 Test Case Description *STS-18*

Manual Test Procedure (STS-18)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Click a marker on the map.	Its corresponding result should become selected in the results section. The result should also be brought on screen if it was previously off screen.
3	Click the thumbs down button within the marker's result section.	The marker related to the result should be removed from the map. The numbers of each marker should also be updated.
4	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.37 Manual Test Procedure *STS-18*

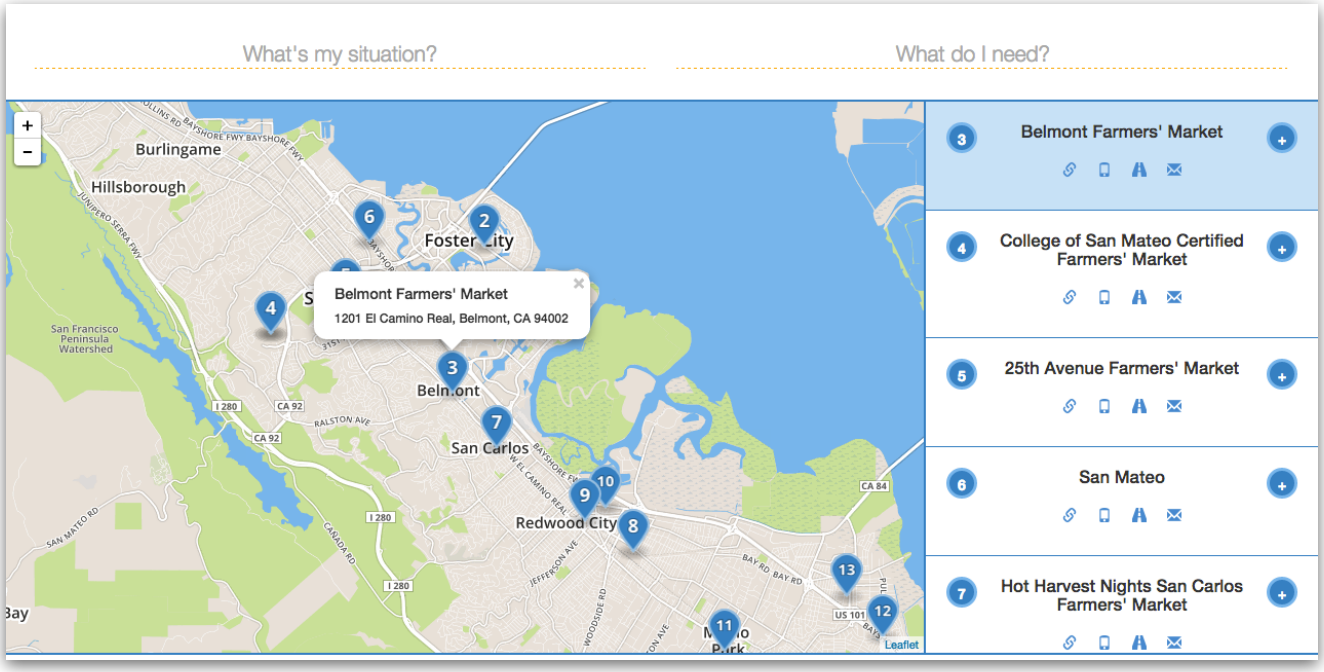

Figure 4.14 After Clicking a marker on the map

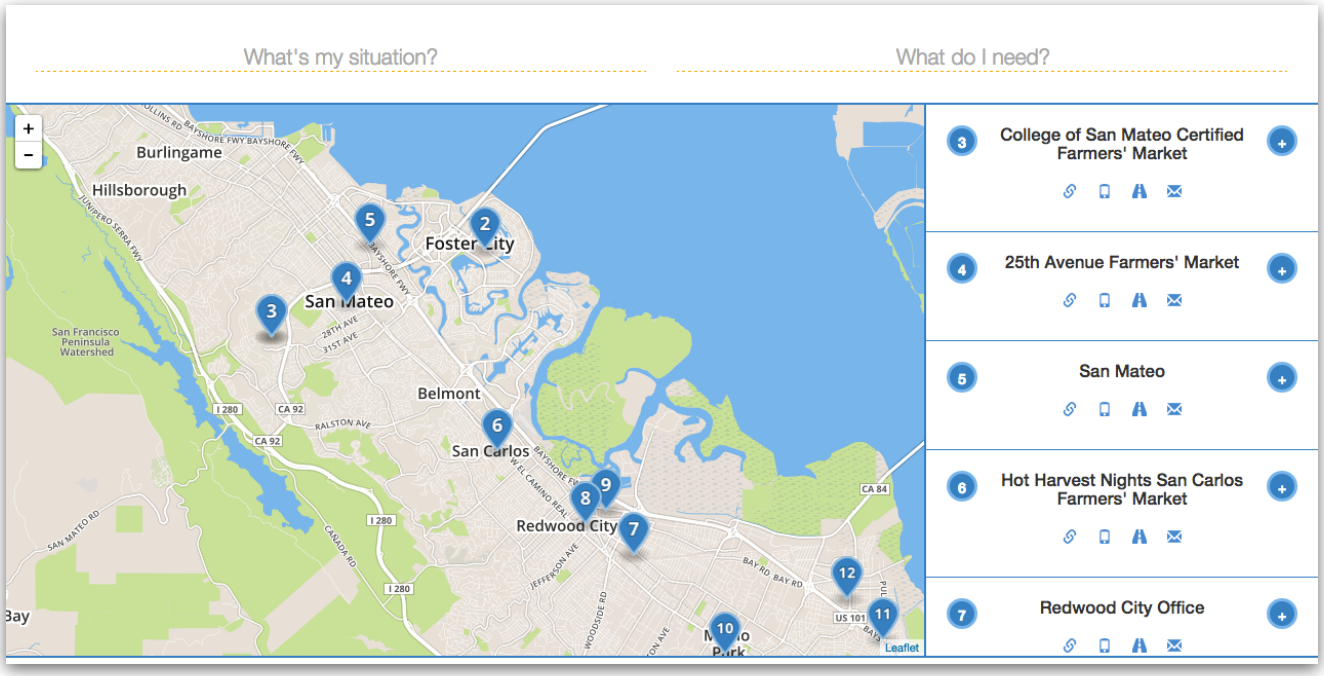

Figure 4.15 After Removing the Result

Click Result on Map (UC16)

When map results are clicked, the corresponding result should be highlighted in the results pane.

Automated Testing

Test Case ID:	STS-19
Test Name:	Click Result on Map (Automated)
Description:	Ensures that the map controller is notified when a marker is clicked.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Coverage, White Box

Table 4.38 Test Case Description *STS-19*

Automated Test Description (STS-19)		
Test	Description	Expected Result
ResultsController (Client Unit Tests)		
1	It should call the search service's selected function when a marker has been clicked	The search service's "selected" function should be called.

Table 4.39 Automated Test Description *STS-19*

Manual Testing

Test Case ID:	STS-20
Test Name:	Click Result on Map (Manual)
Description:	Ensures that the map controller is notified when a marker is clicked.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Interface, Black Box

Table 4.40 Test Case Description (STS-20)

Manual Test Procedure (STS-20)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Click a marker on the map.	The marker's popup should open, also the marker's result should be selected in the results section.
3	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.41 Manual Test Procedure *STS-20*

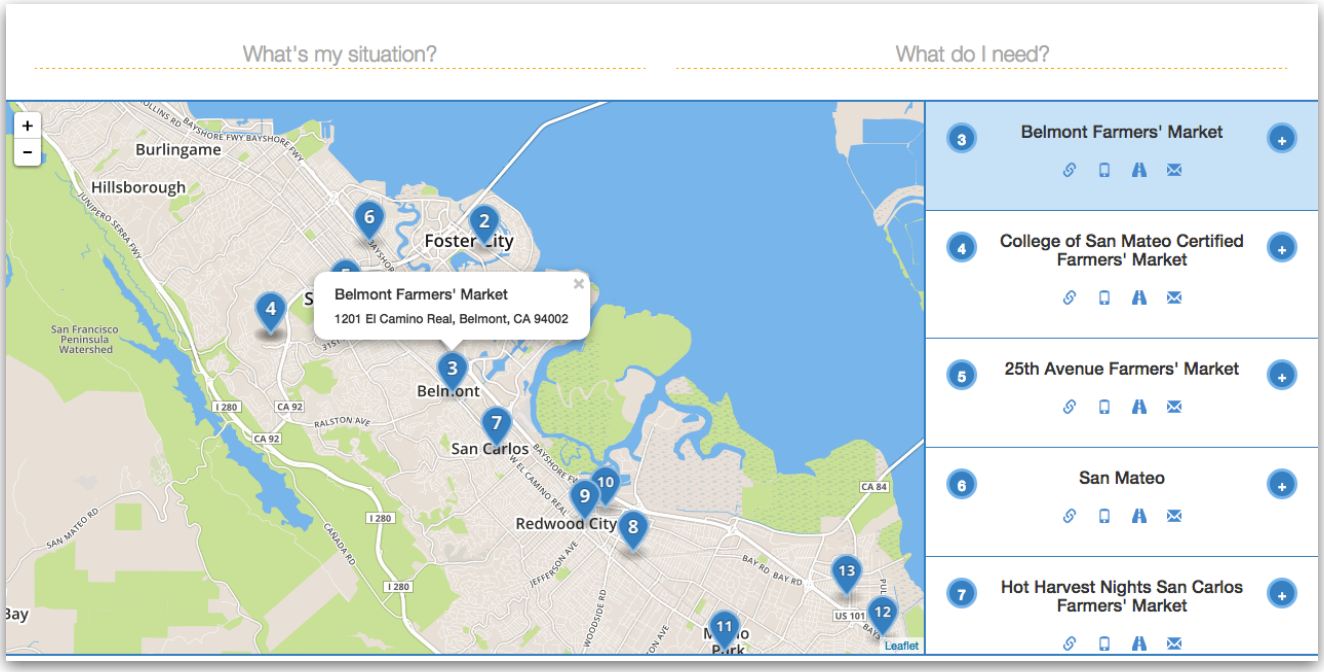

Figure 4.16 After Clicking a Result

4.3 Results

Search for Service (UC1)

See test cases STS-1 and STS-2 for this Use Case, along with the description, in the Search section (4.1).

Browse Map (UC5)

See the description and test cases for this Use Case in the Map section (4.2).

Save interaction data (UC8)

The interactions described in UC10 - UC14 are saved to the application database for use in the application's neural network to improve the relevance of search results. This Use Case is covered by Use Cases UC10 - UC14 in this section.

Interact with Results (UC9)

This Use Case allows a number of interactions with search results, described in detail in UC10 - UC14. It is covered by the test cases for those Use Cases.

Expand Results (UC10)

To avoid overwhelming the user with information, results are initially displayed in a compact state, and the user can choose to expand results they are interested in.

Automated Testing

Test Case ID:	STS-21
Test Name:	Expand Results (Automated)
Description:	Ensures that a result's "expanded" property is set to true when selected to expand. Also make's sure that only one result is expanded at a time.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Interface, Coverage, White Box

Table 4.42 Test Case Description *STS-21*

Automated Test Description (STS-21)		
Test	Description	Expected Result
ResultsController (Client Unit Tests)		
1	It should change result's expand property to true when expandResult function is called.	The result's expand property should be set to true, which lets the jade file know to expand the result.
2	It should collapse a result when a different result is expanded	A result that is already expanded should be collapsed when a different result is expanded.

Table 4.43 Automated Test Description *STS-21*

Manual Testing

Test Case ID:	STS-22
Test Name:	Expand Result (Manual)
Description:	Ensures that a result expands to show information about it when the '+' button is clicked, and that it collapses to hide the information when the '-' button is clicked.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Interface, Black Box

Table 4.44 Test Case Description *STS-22*

Manual Test Procedure (STS-22)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Click the '+' button in the top right corner of a result.	The result should expand to show the thumbs up/down buttons, its address, and hours of operation. The '+' should change to a '-'.
3	Click the '-' button in the top right corner of a result.	The result should collapse to hide the thumbs up/down buttons, its address, and its hours of operation. The '-' should change to a '+'.
4	Click the '+' button in the top right corner of a result. Then click the '+' button of a different result.	The first result should expand when its '+' button is clicked. The first result should then collapse when the second result's '+' button is clicked. The second result should then be the only one expanded.
5	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.45 Manual Test Procedure *STS-22*

Get Directions (UC11)

This Use Case provides users with an easy way to get directions to a particular service. This interaction is saved to the application database for future use by the neural network.

Manual Testing

Test Case ID:	STS-23
Test Name:	Get Directions (Manual)
Description:	Ensures that clicking the directions button on a result redirects to Google maps.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Interface, Black Box

Table 4.46 Test Case Description *STS-23*

Manual Test Procedure (STS-23)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly.
2	Click on the road map icon of one of the results.	Redirects user to Google Maps for directions. User's current location is set as starting location, and result address is set as destination location. The interaction is saved in database.
3	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Android Browser, iPhone Safari	

Table 4.47 Manual Test Procedure *STS-23*

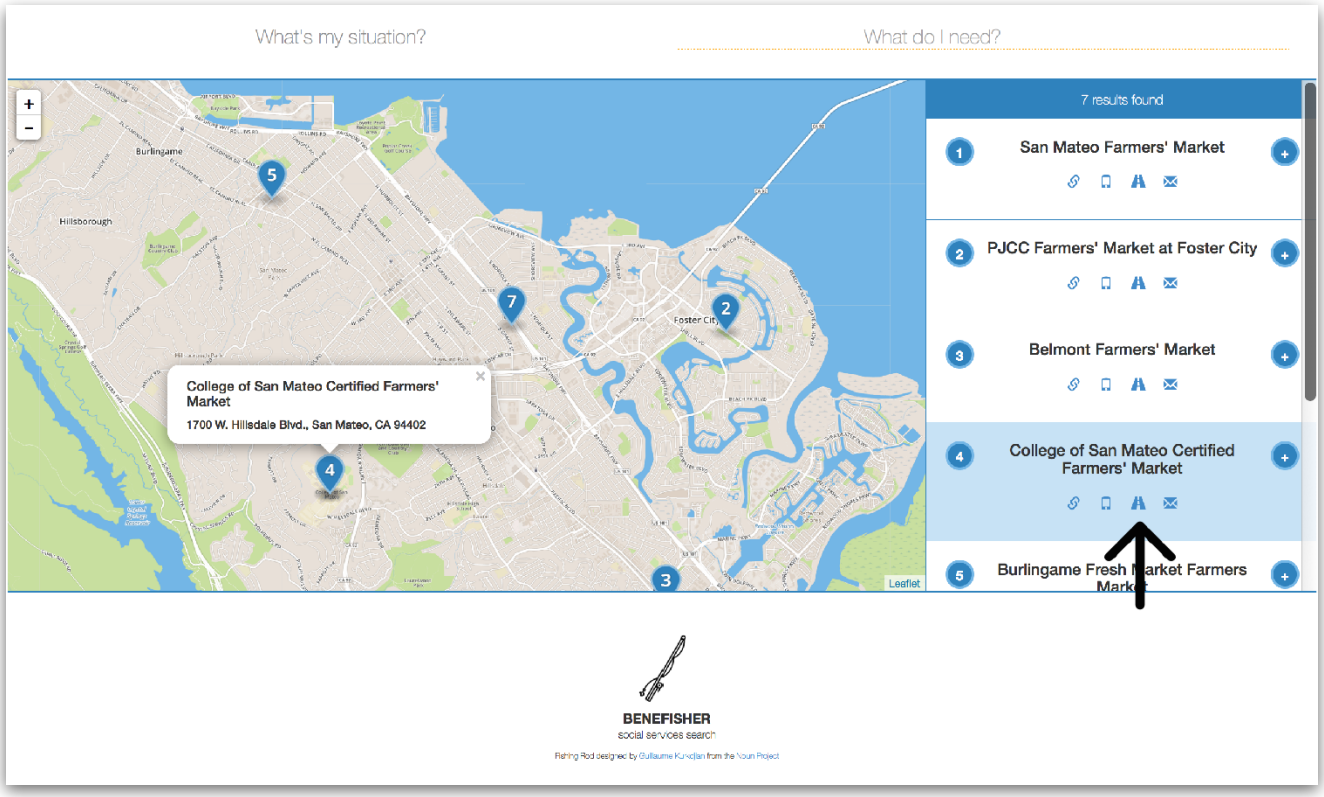

Figure 4.17 Navigate to Service Location Link

Call Service (UC12)

This Use Case allows mobile users to easily call a particular service. This interaction is saved to the application database for future use by the neural network.

Manual Testing

Test Case ID:	STS-24
Test Name:	Call Service (Manual)
Description:	Ensures that clicking the phone button from a mobile device attempts a call to the service.
Prerequisites:	Access application from a mobile device
Test Environment:	Acceptance Server
Test Strategy	Interface, Black Box

Table 4.48 Test Case Description *STS-24*

Manual Test Procedure (STS-24)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly.
2	Click on the cellphone icon of one of the results.	Mobile device should attempt to call the number of the service, and the interaction is saved in database.
3	Repeat in each of the following browsers: Mobile: Andriod Browser, iPhone Safari	

Table 4.49 Manual Test Procedure *STS-24*

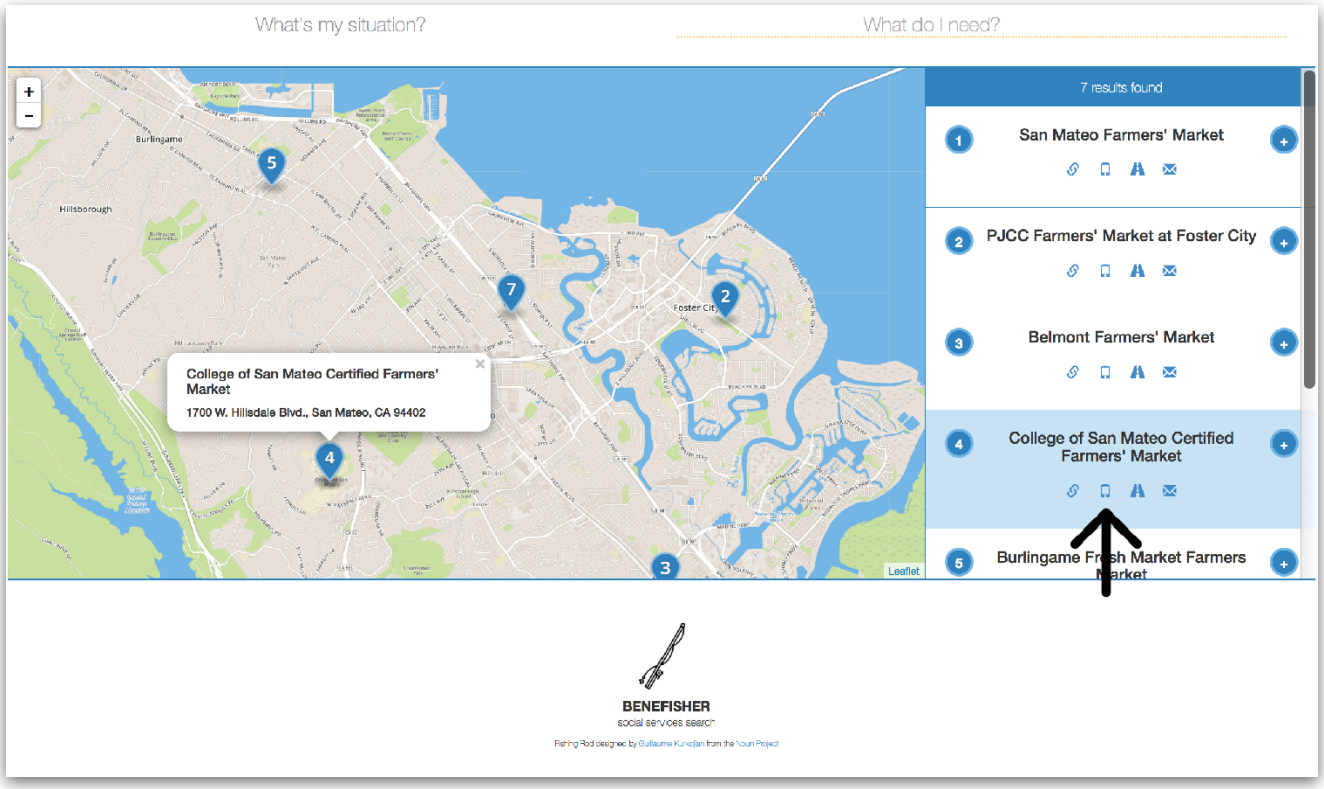

Figure 4.18 Call Service Link

Navigate to Site (UC13)

The user can visit a particular service's website. This interaction is saved to the application database for future use by the neural network.

Manual Testing

Test Case ID:	STS-25
Test Name:	Navigation to site (Manual)
Description:	Ensures that clicking a result's web address redirects the user to the service's website.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Interface, Coverage, White Box

Table 4.50 Test Case Description *STS-25*

Manual Test Procedure (STS-25)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly.
2	Click on the link icon of one of the results.	Browser should redirect to the service's website, and the interaction should be successfully saved in database.
3	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.51 Manual Test Procedure *STS-25*

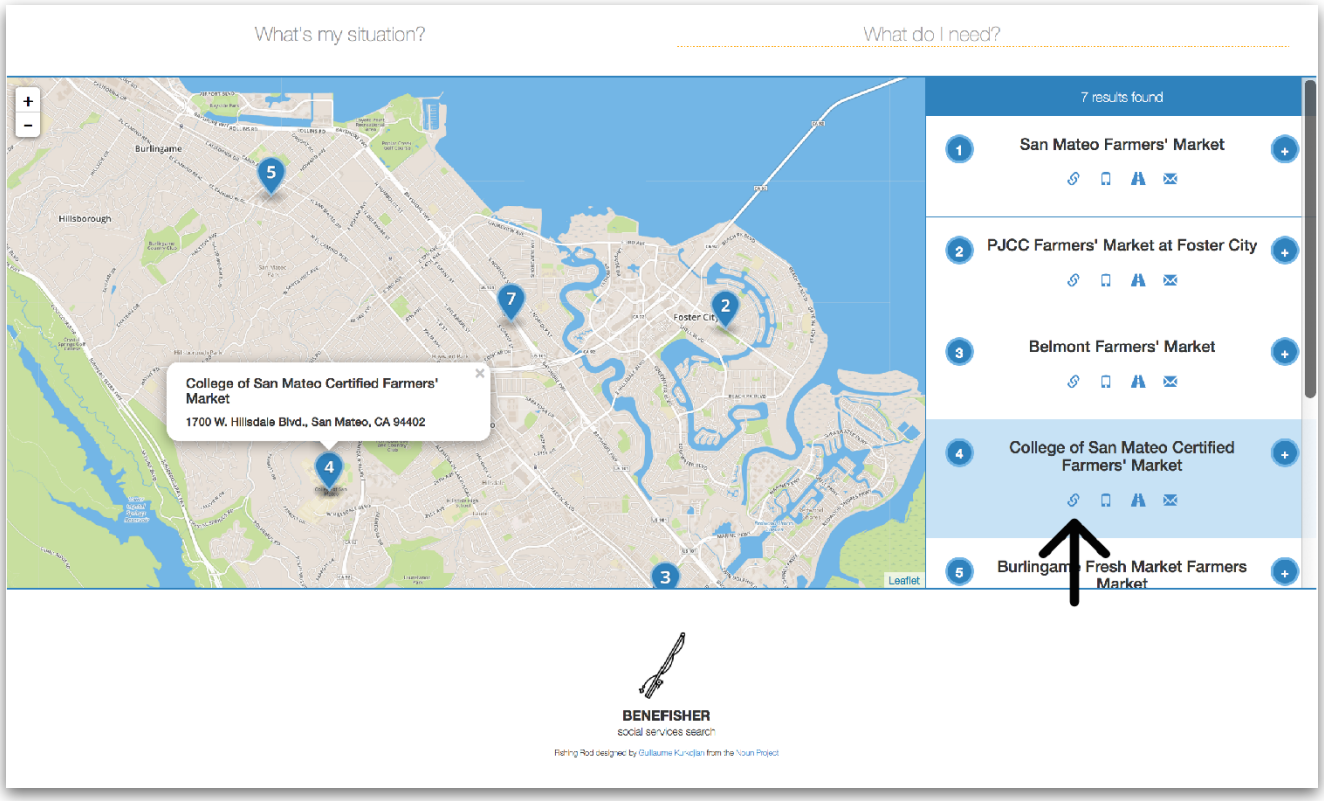

Figure 4.19 Navigation to Site Link

Email Service (UC14)

This use case opens the user's default email application, and begins composition of an email to a particular service. This interaction is saved to the application database for future use by the neural network.

Manual Testing

Test Case ID:	STS-26
Test Name:	Email Service (Manual)
Description:	Ensures that a clicking a result's email address allows the user to send an email to that address.
Prerequisites:	N/A
Test Environment:	Acceptance Server
Test Strategy	Interface, Black Box

Table 4.52 Test Case Description *STS-26*

Manual Test Procedure (STS-26)		
Step	Description	Expected Results
1	Open the Benefisher Application in a browser.	The application is displayed correctly
2	Click on the email icon of one of the results	Default email client opens, and creates a new email with the service's email address set in the "to" field. The interaction is saved in database
3	Repeat in each of the following browsers: Desktop: Google Chrome, Mozilla Firefox, Internet Explorer Mobile: Andriod Browser, iPhone Safari	

Table 4.53 Manual Test Procedure *STS-26*

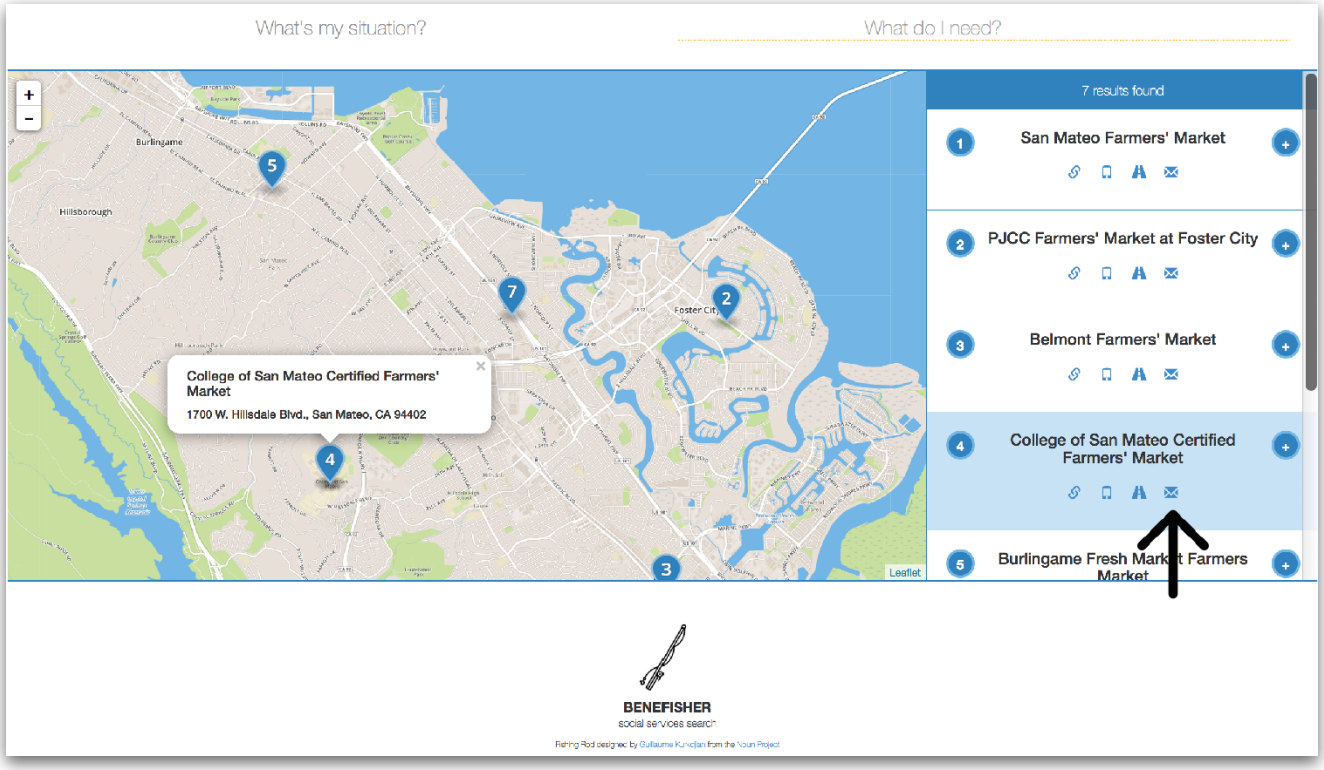

Figure 4.20 Email Link

Remove from Screen (UC15)

See the description and test cases for this Use Case in the Search (4.1) and Map (4.2) sections.

Click Result on Map (UC16)

See the description and test cases for this Use Case in the Map section (4.2).

5. REQUIREMENTS TRACEABILITY

Use Case	System Test ID	Design Components
Search for Services (UC-1)	STS-1 - Search for Service (Automated) STS-2 - Search for Service (Manual) STS-19 - Subscribe to Search Service (Automated) STS-20 - Recieve Search Data (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade search.jade notifications.jade app.js (client) services.js search-controller.js search-service.js notification-service.js
Search by Need (UC-2)	STS-3 - Search by Need (Automated) STS-4 - Search by Need (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade search.jade app.js (client) services.js search-controller.js search-service.js
Search by Situation (UC-3)	STS-5 - Search by Situation (Automated) STS-6 - Search by Situation (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade search.jade app.js (client) services.js search-controller.js search-service.js
Search by Need & Situation (UC-4)	STS-7 - Search by Need & Situation (Automated) STS-8 - Search by Need & Situation (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade search.jade app.js (client) services.js search-controller.js search-service.js

Use Case	System Test ID	Design Components
Browse Map (UC-5)	STS-9 - Browse Map (Automated) STS-10 - Browse Map (Manual) STS-21 - Update Map on Move (Automated) STS-22 - Update Map on Move (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade map.jade app.js (client) search-service.js map-controller.js
Select Pre-Defined Need (UC-6)	STS-11 - Select Pre-Defined Need (Automated) STS-12 - Select Pre-Defined Need (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade search.jade app.js (client) services.js search-controller.js search-service.js
Select Pre-Defined Situation (UC-7)	STS-13 - Select Pre-Defined Situation (Automated) STS-14 - Select Pre-Defined Situation (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade search.jade app.js (client) services.js search-controller.js search-service.js
Save Interaction Data (UC-8)	STS-27 - Expand Results (Automated) STS-28 - Expand Results (Manual) STS-29 - Get Directions (Manual) STS-30 - Call Service (Manual) STS-31 - Navigate to Site (Manual) STS-32 - Email Serivce (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade results.jade app.js (client) services.js search-service.js result-controller.js interactions.js

Use Case	System Test ID	Design Components
Interact with Results (UC-9)	STS-15 - Remove From Screen (Automated) STS-16 - Remove From Screen (Manual) STS-19 - Subscribe to Search Service (Automated) STS-20 - Recieve Search Data (Manual) STS-21 - Updated Map on Move (Automated) STS-22 - Update Map on Move (Manual) STS-23 - Remove Marker from Map (Automated) STS-24 - Remove Marker from Map (Manual) STS-25 - Click Result on Map (Automated) STS-26 - Click Result on Map (Manual) STS-27 - Expand Results (Automated) STS-28 - Expand Results (Manual) STS-29 - Get Directions (Manual) STS-30 - Call Service (Manual) STS-31 - Navigate to Site (Manual) STS-32 - Email Service (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade results.jade map.jade app.js (client) services.js search-service.js map-controller.js result-controller.js interactions.js
Expand Results (UC-10)	STS-27- Expand Results (Automated) STS-28 - Expand Results (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade results.jade app.js (client) services.js search-service.js result-controller.js
Get Directions (UC-11)	STS-29 - Get Directions (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade results.jade app.js (client) services.js search-service.js result-controller.js
Call Service (UC-12)	STS-30 - Call Service (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade results.jade app.js (client) services.js search-service.js result-controller.js

Use Case	System Test ID	Design Components
Navigate to Site (UC-13)	STS-31 - Navigate to Site (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade results.jade app.js (client) services.js search-service.js result-controller.js
E-mail Service (UC-14)	STS-32 - Email Serivce (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade results.jade app.js (client) services.js search-service.js result-controller.js
Remove from Screen (UC-15)	STS-15 - Remove From Screen (Automated) STS-16 - Remove From Screen (Manual) STS-23 - Remove Marker From Map (Automated) STS-24 - Remove Marker From Map (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade app.js (client) services.js search-service.js result-controller.js
Click Result on Map (UC-16)	STS-25 - Click Result on Map (Automated) STS-26 - Click Result on Map (Manual)	app.js (server) index.js layout.jade index.jade scripts.jade map.jade app.js (client) services.js search-service.js map-controller.js
Save Search Query (UC-17)	STS-17 - Save Search Query (Automated) STS-18 - Save Search Query (Manual)	

Table 5.1 Use Case-Test Case-Component Matrix

9. APPROVALS

The signatures here indicate that the relevant stakeholders understand the contents of this document, and approve of the outlined software design.

Name	Signature	Date
Adrian Chambers		
Anthony Cristiano		
Daniel Green		
James Doan		
Jesse Rosato		
Sponsor Representative		
Meiliu Lu (Advisor)		

Table 6.1 Project Team, Advisor, and Sponsor Signatures