Final Project - Persistent and Secure Contact Lists

**CMPS 369 - Web Application Development - Final Project**

For all parts of this assignments, you are expected to use Node.js on the server and standard HTML/CSS/JS on the client.  While it’s not required, you may find it easier to use the Express web framework to do your server-side code.  In addition, while you are free to use EJS for template rendering, you might also consider Jade.  Please see the course syllabus for when we will be covering these topics.  It is completely your choice (and your responsibility) to make these types of choices.

**Summary:**

There is a total of 125 points available - which includes 25 points of extra credit.  The details of each part are explained on the next page.

|  |  |  |
| --- | --- | --- |
| Part 1 | Persistent Contact List | 20 Points |
| Part 2 | Location (Lat/Long) for Contacts | 10 Points |
| Part 3 | Mapping Contacts | 10 Points |
|  | Contact Info Map Tooltips | * 5 Extra Credit |
| Part 4 | Contact CRUD | 30 Points |
|  | Single Page App - CRUD | * 10 Extra Credit |
| Part 5 | Securing the Contacts page | 10 Points |
| Part 6 | Make it look nice with CSS | 20 Points |
| Part 7 | Name Search Fields | * 5 Extra Credit |
| Part 8 | Location-Based Search | * 5 Extra Credit |

Note that the assignment is designed to allow you to choose where to focus.  For example, if you avoid the geocoding / map parts (Part 2 and 3) entirely, you can still earn 95 points by doing the CRUD SPA (+10) and Contact Search (+5).  Likewise, the points from Part 5 can be made up through extra credit as well.

**Project Details**

**Part 1:   Persistent Contact List**

In Project 1 you created a contact form and saved it to an array in memory.  You also created a /contacts page which served up a list of all the contacts in your array.  Now you must store each contact into a MongoDB collection called “contacts”.  Each document in the contacts collection should have all the information you collected about contacts in Project 1:

First Name                 (String)

Last Name                 (String)

Street                         (String)

City                         (String)

State                         (String)

Zip                         (String)

Phone                         (String)

Email                         (String)

Mr./Mrs./Ms./Dr.         (String)

Contact By Mail        (Boolean)

Contact By Phone        (Boolean)

Contact BY Email         (Email)

Users should be able to visit /mailer to get a standard HTML form where they can fill out their information.

The form should submit (standard POST - not AJAX) to server, where the contact information will be inserted into the database.  The user should then see a thank you page.

GET Requests to /contacts shows a table of all contact information found in the database.

**Part 2:        Adding Location (Latitude and Longitude) to Contacts**

In Project 2 you learned how to use Google’s geocoding API to convert an address into latitude longitude coordinates.  We will now take each new contact’s address and use the same API to find their location, and store latitude and longitude to the database as well (as part of their contact information).

You have two choices here regarding when to do the geocoding - both can work, and both have their advantages/disadvantages.

Option 1 is to perform the geocoding from the web browser itself.  This would require you to capture the “submit” event on the /mail form submission, and do the geocoding instead.  Then, once the geocoding is complete, you can programmatically submit the form yourself (jQuery has a simple $(form).submit() function!).

Option 2 is to perform the geocoding on the server.  You can easily make web requests from Node - and there are some libraries to help you.  A quick web search turned this one up - [https://github.com/nchaulet/node-geocoder](https://www.google.com/url?q=https://github.com/nchaulet/node-geocoder&sa=D&ust=1551461498058000)

Choice is yours.

**Part 3:  Add a Map to the /contacts page**

When the /contacts page is visited, the top half of the page should be the familiar table of contact information.  However, below the table should now include a map, with each contact plotted as a marker on the map.  The map should be interactive - in that when a user clicks on a contact in the contact table, the map should automatically move to the contact’s location.

+5 Extra credit opportunity:  Display contact info as a tooltip when the user hovers/clicks on the marker on the map.

**Part 4:  Contacts CRUD**

Until now, all you can do is Read all the contacts on the /contact page.  Add editability to the contact list.

On the /contact page, add a button to Create a new contact.  This will take you to /mailer (or similar) page.

Next to each contact on the contact table, you should have a link to Update the Contact.  This should take the user to a new page (which looks like the initial /mailer form), but with the form elements already pre-populated. The user can edit the values, and click save.  Be sure that the contact is simply Updated - not that a new one is added.

Hint:  Include the MongoDB \_id value as a hidden form field on your edit page - so when the user submits you can look that contact up in the DB and update the contact.

Finally, add a Delete button next to each contact in the contact table as well.    Clicking on it will delete the user, and re-display the /contacts page.

+10 Extra credit opportunity:  Implement the CRUD application as a Single Page Application using AJAX.  For this, Create and Update should NOT navigate the user to a new page - rather, those buttons should display a (previously hidden) form on the /contact page itself.  After clicking “Save”, use AJAX to save the new/edited contact and have the server send back the updated list of contacts.  Your table (and Map) should then be updated dynamically.

**Part 5:  Secure the /contacts page**

When you sign up for a service, you don’t expect to get access to information about all other users!  Lets now secure the /contacts page so only “admins” can see this stuff.  All requests made to /contacts, along with the ability to do the CRUD operations in Part 4, should require a user to already be logged in.

Create a login page - /login that allows the user to enter a username and password.  We’re going to keep things really simple here - the username is “cmps369” and the password is “finalproject”.  This is exceptionally poor practice - but I’m trying to keep the scope of this project limited!

Anytime a user access the contacts pages (not the /mailer form - anyone can enter their information, not authenticated), they must be logged in.  If they are not, redirect them to /login.

For authentication, make use of passport - the de-facto module for doing authorization in express.  We’ll cover this in [Module 20](https://www.google.com/url?q=https://docs.google.com/presentation/d/1brqevzLcihmPsQNe-GJWu8wF75WV6q7eiNnkz73IMZo/pub?start%3Dfalse%26loop%3Dfalse%26delayms%3D3000%26slide%3Did.p&sa=D&ust=1551461498063000).

**Part 6:  Make your web application look nice**

All of your pages throughout your site should now look professional, and consistent.  I’m not giving you lots of specifics here - beauty is often in the eyes of the beholder!  Here’s my grading guidelines:

* All pages on the site should have consistent font styles, color themes, spacing.  You’d be wise to use externally (and shared) CSS style sheets to ensure this.  You’d be even wiser to make use of Bootstrap (we’re covering this late in the semester).
* All pages should be reasonably responsive - meaning resizing the web page should result in a professional and usable looking page.  Again - you’d be smart to use Bootstrap!

**Part 7:  +5 Extra Credit Opportunity  - Searching the Contact Page**

At the top of the /contact page, add search fields for First Name and Last Name.  When the user types values, the contact table should be automatically filtered to display only contacts whose first name/last name includes the text written in the search field.  They do not need to be perfect matches - if there is a contact named “Abigail”, then typing “Ab” into the First name search box would allow her to be shown in the results.

**Part 8:  +5 Extra Credit Opportunity - Searching By Address**

Add a search field to the /contacts page where a user can type in an address.  Add drop down so the user can select the search radius - 5 miles, 10 miles, 25 miles should be the three choices.

When the user sets the address in the search field - you need to geocode the address to obtain its latitude and longitude.  Then, update the contact page to only show contacts who are within the given search radius.  Also re-center the map to the search address.

You can convert distances between latitude and longitude using the following reference - or you might be able to find other ways - it's up to you!

[https://www.geodatasource.com/developers/javascript](https://www.google.com/url?q=https://www.geodatasource.com/developers/javascript&sa=D&ust=1551461498065000)

Published by [Google Drive](https://docs.google.com/)–[Report Abuse](https://docs.google.com/abuse?id=1lfnxI3C_44m9MwZaBxAtfBb9w6jciJvoIsSQw40mXvg)–Updated automatically every 5 minutes