# Spring MVC Tutorial – Contact List Application

Step 12: Wiring View to Search REST Controller Endpoints

# Step 12: Wiring View to Search REST Controller Endpoints

## Overview

In this step, we'll wire the Search View components created in Step 04 to the Search REST endpoint created in Step 10. In order to do this, we'll have to create and register an onClick handler for the Search form like the ones we have for the Add and Edit forms. The new onClick handler will submit an Ajax call to our Search REST endpoint and then use the returned Contact data to populate the Contact Table with the search results. As we implement this new feature, we will also take the opportunity to refactor the JavaScript code that renders an array of Contact objects to the Contacts Table into a method so that it can be reused by this new feature.

## Refactoring Code

Our new Search feature will issue an Ajax call to the server and will render the returned array of Contacts to the Contact Table. This is similar to the loadContacts method — the only difference is which REST endpoint is called. To prevent having to copy the code that renders the Contact Table, we'll refactor that code into a new JavaScript function. Create the following function in the contactList.js file (this code was taken from the loadContacts function):

```
// fills the Contact Table with results from an Ajax call - used in conjunction
// with the Search button and loadContact function
function fillContactTable(contactList, status) {
    // clear the previous list
    clearContactTable();
    // grab the tbody element that will hold the new list of contacts
    var cTable = $('#contentRows');

    // render the new contact data to the table
    $.each(contactList, function (index, contact) {
        cTable.append($('<tr>')
                .append($('<td>')
                        .append($('<a>')
                                .attr({
                                    'data-contact-id': contact.contactId,
                                    'data-toggle': 'modal',
                                    'data-target': '#detailsModal'
                                })
                                .text(contact.firstName + ' ' + contact.lastName)
                                ) // ends the <a> tag
                        ) // ends the <td> tag for the contact name
                .append($('<td>').text(contact.company))
                .append($('<td>')
                        .append($('<a>')
                                .attr({
                                    'data-contact-id': contact.contactId,
                                    'data-toggle': 'modal',
                                    'data-target': '#editModal'
                                })
                                .text('Edit')
                                ) // ends the <a> tag
                        ) // ends the <td> tag for Edit
                .append($('<td>')
                        .append($('<a>')
                                .attr({
                                    'onClick': 'deleteContact(' + contact.contactId + ')'
                                })
                                .text('Delete')
                                ) // ends the <a> tag
                        ) // ends the <td> tag for Delete
                );  // ends the <tr> for this Contact
    });  // ends the 'each' function
}
```

Now, modify your loadContacts function to that it looks like this (we're simply removing the code we just put in the fillContactTable function):

```
// Load contacts into the summary table
function loadContacts() {
    // Make an Ajax GET call to the 'contacts' endpoint. Iterate through
    // each of the JSON objects that are returned and render them to the
    // summary table.
    $.ajax({
        url: "contacts"
    }).success(function(data, status){
        fillContactTable(data, status);
    });
}
```

## Search Button Click Handler

Next, we will create the click handler for the Search button.  This handler will be similar to the handler for the Add button.  Add the following code right below the Edit button click handler:

Notes:

1. The click handler issues a POST to search/contacts (we created this endpoint in the previous step).
2. It passes the data from the Search form to the server.
3. On success, it clears the Search form and then calls fillContactTable, passing the data returned from the Ajax call.

```
// on click for our search button
    $('#search-button').click(function (event) {
        // we don’t want the button to actually submit
        // we'll handle data submission via ajax
        event.preventDefault();
        $.ajax({
            type: 'POST',
            url: 'search/contacts',
            data: JSON.stringify({
                firstName: $('#search-first-name').val(),
                lastName: $('#search-last-name').val(),
                company: $('#search-company').val(),
                phone: $('#search-phone').val(),
                email: $('#search-email').val()
            }),
            headers: {
                'Accept': 'application/json',
                'Content-Type': 'application/json'
            },
            'dataType': 'json'
        }).success(function (data, status) {
            $('#search-first-name').val('');
            $('#search-last-name').val('');
            $('#search-company').val('');
            $('#search-phone').val('');
            $('#search-email').val('');

            fillContactTable(data, status);
        });
    });
```

## Wrap-up

In this step, we wired our Search form to the Search REST endpoint.  Along the way, we were able to refactor our JavaScript code to avoid repeated code.