This is a shorter lab. We’ll finish up the client side only behavior by implementing Delete.

In the index.html file, add a delete button after the edit button:

<div class="col-sm-3">

    <button class="btn btn-default" ng-click="deleteContact(contact)">

        <span>Delete</span>

    </button>

</div>

Next, let’s add the controller handler to delete the contact:

$scope.deleteContact = function (contact) {

    var index = $scope.contacts.indexOf(contact);

    $scope.contacts.splice(index, 1);

};

Well, we should write some tests for this:

describe("delete Button", function () {

    var scope, controller;

    beforeEach(inject(function ($rootScope, $controller) {

        scope = $rootScope.$new();

        controller = $controller;

    }));

it("delete removes a contact", function () {

    controller("contactsController", { $scope: scope });

    var length = scope.contacts.length;

    var contactToDelete = scope.contacts[length - 2];

    scope.deleteContact(contactToDelete);

    expect(scope.contacts.length).toBe(length - 1);

    expect(scope.contacts.indexOf(contactToDelete)).toBe(-1);

});

});

Run the tests and what happens? Did all your tests fail? That’s because in the last lab, we didn’t add components.js to the include list in the specrunner.html file. Add that, and then our tests work.

Let’s add a confirmation box to our delete method:

$scope.deleteContact = function (contact) {

    var index = $scope.contacts.indexOf(contact);

    if (confirm("Are you sure you want to delete?"))

        $scope.contacts.splice(index, 1);

};

Run the app. See that it works.

Now, run your tests. What happens?

Well, you can make your test pass or fail at will. If you say ‘yes’ to the confirmation, it passes. If you cancel, it fails.

And, think about how this would work running without a browser as part of an automatic build. It wouldn’t.

There are several ways to fix this. The simplest is to move the confirmation into the html page, but we’re not going to do that because it doesn’t allow testing. Instead, we’ll replace the confirm call with a call to $window.confirm() so that we can mock it in tests.

First, add the $window parameter to the controller function:

contactsApp.controller("contactsController",

    function ($scope, $window) {

Then, replace confirm() with $window.confirm():

$scope.deleteContact = function (contact) {

    var index = $scope.contacts.indexOf(contact);

    if ($window.confirm("Are you sure you want to delete?"))

        $scope.contacts.splice(index, 1);

};

Run the HTML page, and esure it still works.

Run your tests. The behavior is the same. Now, let’s mock the window object in our tests.

In the delete button tests, modify the beforeEach to include the $window parameter. Cache that in a window variable scoped to this section:

describe("delete Button", function () {

    var scope, controller, window;

    beforeEach(inject(function ($rootScope, $controller, $window) {

        scope = $rootScope.$new();

        controller = $controller;

        window = $window;

    }));

Now, modify the first test so that the call to controller includes the $window parameter:

it("delete removes a contact", function () {

    controller("contactsController", { $scope: scope, $window: window });

Finally, define the behavior for the window.confirm method. For this test to pass, we would expect the user to press “yes”, and confirm to return true:

window.confirm = function () {

    return true;

};

And now, we can write one more useful test to ensure we don’t delete a contact when the user presses cancel:

it("delete and cancel does nothing contact", function () {

    controller("contactsController", { $scope: scope, $window: window });

    window.confirm = function () {

        return false;

    };

    var length = scope.contacts.length;

    var contactToDelete = scope.contacts[length - 2];

    scope.deleteContact(contactToDelete);

    expect(scope.contacts.length).toBe(length);

    expect(scope.contacts.indexOf(contactToDelete)).toBe(length - 2);

});

Run the tests again, and verify the behavior is correct.

Tomorrow, we’ll start writing server code.