We did a lot of work in laying a foundation for the app in the last step, so now we'll do something simple; we will add full text search (yes, it will be simple!). We will also write an end-to-end test, because a good end-to-end test is a good friend. It stays with your app, keeps an eye on it, and quickly detects regressions.

- The app now has a search box. Notice that the phone list on the page changes depending on what a user types into the search box.

Workspace Reset Instructions ➤

The most important changes are listed below. You can see the full diff on GitHub

Controller

We made no changes to the controller.

Template

app/index.html :

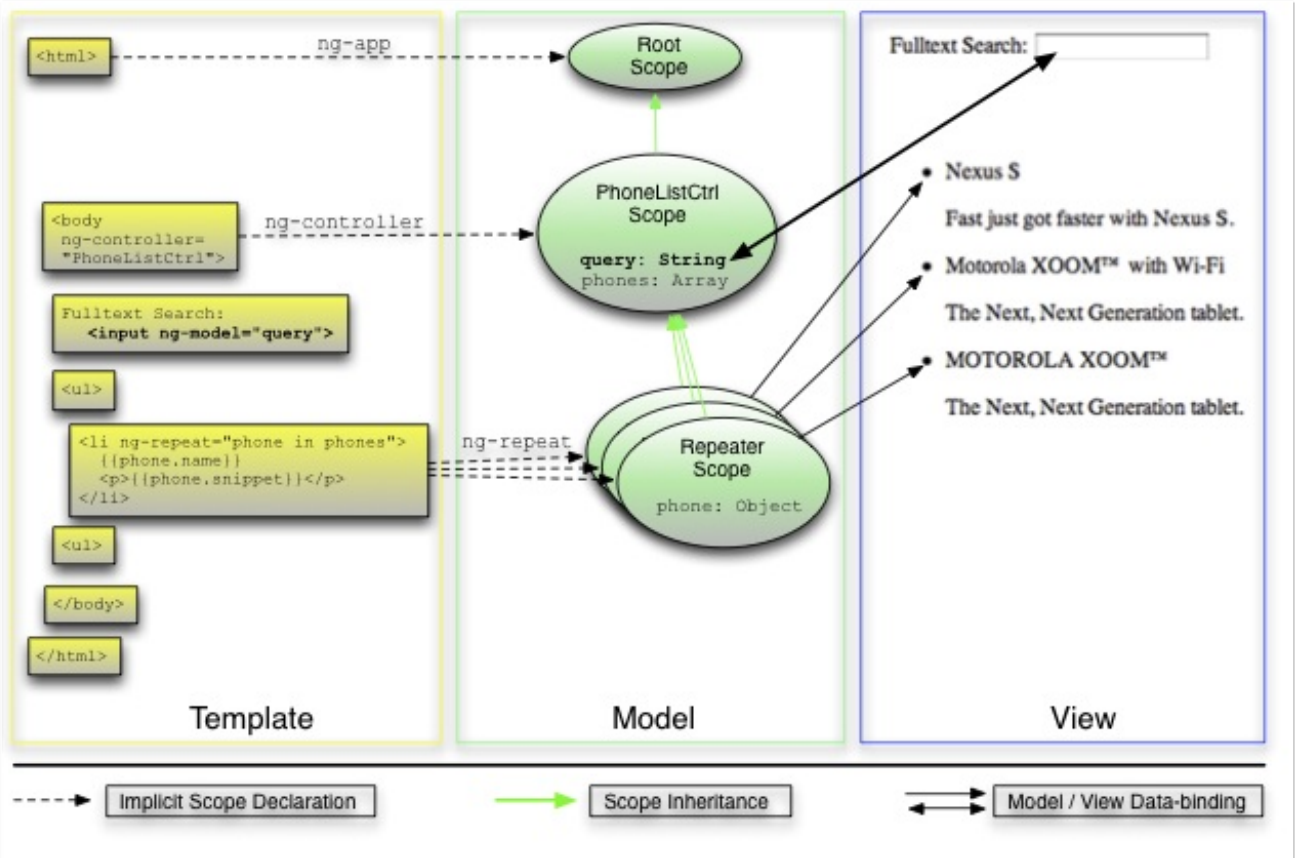
```
<div class="container-fluid">
  <div class="row">
    <div class="col-md-2">
      <!--Sidebar content-->

      Search: <input ng-model="query">

    </div>
    <div class="col-md-10">
      <!--Body content-->

      <ul class="phones">
        <li ng-repeat="phone in phones | filter:query">
          {{phone.name}}
          <p>{{phone.snippet}}</p>
        </li>
      </ul>

    </div>
  </div>
</div>
```

We added a standard HTML `<input>` tag and used Angular's filter function to process the input for the ngRepeat directive.

This lets a user enter search criteria and immediately see the effects of their search on the phone list. This new code demonstrates the following:

- Data-binding: This is one of the core features in Angular. When the page loads, Angular binds the name of the input box to a variable of the same name in the data model and keeps the two in sync.

  In this code, the data that a user types into the input box (named **query**) is immediately available as a filter input in the list repeater (`phone in phones | filter:` **query**). When changes to the data model cause the repeater's input to change, the repeater efficiently updates the DOM to reflect the current state of the model.



- Use of the `filter` filter: The filter function uses the `query` value to create a new array that contains only those records that match the `query`.

  `ngRepeat` automatically updates the view in response to the changing number of phones returned by the `filter` filter. The process is completely transparent to the developer.

Test

In Step 2, we learned how to write and run unit tests. Unit tests are perfect for testing controllers and other components of our application written in JavaScript, but they can't easily test DOM manipulation or the wiring of our application. For these, an end-to-end test is a much better choice.

The search feature was fully implemented via templates and data-binding, so we'll write our first end-to-end test, to verify that the feature works.

`test/e2e/scenarios.js` :

```
describe('PhoneCat App', function() {

describe('Phone list view', function() {

  beforeEach(function() {
    browser.get('app/index.html');
  });


  it('should filter the phone list as a user types into the search box', function() {

    var phoneList = element.all(by.repeater('phone in phones'));
    var query = element(by.model('query'));

    expect(phoneList.count()).toBe(3);

    query.sendKeys('nexus');
    expect(phoneList.count()).toBe(1);

    query.clear();
    query.sendKeys('motorola');
    expect(phoneList.count()).toBe(2);
  });
});
});
```

This test verifies that the search box and the repeater are correctly wired together. Notice how easy it is to write end-to-end tests in Angular. Although this example is for a simple test, it really is that easy to set up any functional, readable, end-to-end test.

Running End to End Tests with Protractor

Even though the syntax of this test looks very much like our controller unit test written with Jasmine, the end-to-end test uses APIs of Protractor. Read about the Protractor APIs at https://github.com/angular/protractor/blob/master/docs/api.md.

Much like Karma is the test runner for unit tests, we use Protractor to run end-to-end tests. Try it with `npm run protractor`. End-to-end tests are slow, so unlike with unit tests, Protractor will exit after the test run and will not automatically rerun the test suite on every file change. To rerun the test suite, execute `npm run protractor` again.

Note: You must ensure your application is being served via a web-server to test with protractor. You can do this using `npm start`. You also need to ensure you've installed the protractor and updated webdriver prior to running the `npm run protractor`. You can do this by issuing `npm install` and `npm run update-webdriver` into your terminal.

Experiments

Display Current Query

Display the current value of the `query` model by adding a `{{query}}` binding into the `index.html` template,

and see how it changes when you type in the input box.

Display Query in Title

Let's see how we can get the current value of the `query` model to appear in the HTML page title.

- Add an end-to-end test into the `describe` block, `test/e2e/scenarios.js` should look like this:

```
describe('PhoneCat App', function() {

  describe('Phone list view', function() {

    beforeEach(function() {
      browser.get('app/index.html');
    });

    var phoneList = element.all(by.repeater('phone in phones'));
    var query = element(by.model('query'));

    it('should filter the phone list as a user types into the search box', function() {
      expect(phoneList.count()).toBe(3);

      query.sendKeys('nexus');
      expect(phoneList.count()).toBe(1);

      query.clear();
      query.sendKeys('motorola');
      expect(phoneList.count()).toBe(2);
    });

    it('should display the current filter value in the title bar', function() {
      query.clear();
      expect(browser.getTitle()).toMatch(/Google Phone Gallery:\s*$/);

      query.sendKeys('nexus');
      expect(browser.getTitle()).toMatch(/Google Phone Gallery: nexus$/);
    });
  });
});
```

  Run protractor (`npm run protractor`) to see this test fail.

- You might think you could just add the `{{query}}` to the title tag element as follows:

```
<title>Google Phone Gallery: {{query}}</title>
```

  However, when you reload the page, you won't see the expected result. This is because the "query" model lives in the scope, defined by the `ng-controller="PhoneListCtrl"` directive, on the body element:

```
<body ng-controller="PhoneListCtrl">
```

  If you want to bind to the query model from the `<title>` element, you must **move** the `ngController` declaration to the HTML element because it is the common parent of both the body and title elements:

```
<html ng-app="phonecatApp" ng-controller="PhoneListCtrl">
```

Be sure to **remove** the `ng-controller` declaration from the body element.

- Re-run `npm run protractor` to see the test now pass.
- While using double curlies works fine within the title element, you might have noticed that for a split second they are actually displayed to the user while the page is loading. A better solution would be to use the ngBind or ngBindTemplate directives, which are invisible to the user while the page is loading:

```
<title ng-bind-template="Google Phone Gallery: {{query}}">Google Phone Gallery</title>
```

Summary

We have now added full text search and included a test to verify that search works! Now let's go on to step 4 to learn how to add sorting capability to the phone app.

Previous | Live Demo | Code Diff | Next