In this step, you will add thumbnail images for the phones in the phone list, and links that, for now, will go nowhere. In subsequent steps you will use the links to display additional information about the phones in the catalog.

- There are now links and images of the phones in the list.

Workspace Reset Instructions ➤

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

---

Data

Note that the `phones.json` file contains unique ids and image urls for each of the phones. The urls point to the `app/img/phones/` directory.

`app/phones/phones.json` (sample snippet):

```
[
{
  ...
  "id": "motorola-defy-with-motoblur",
  "imageUrl": "img/phones/motorola-defy-with-motoblur.0.jpg",
  "name": "Motorola DEFY\u2122 with MOTOBLUR\u2122",
  ...
},
...
]
```

Template

app/index.html :

```
...
<ul class="phones">
  <li ng-repeat="phone in phones | filter:query | orderBy:orderProp" class="thumbnail">
    <a href="#/phones/{{phone.id}}" class="thumb"><img ng-src="{{phone.imageUrl}}"></a>
    <a href="#/phones/{{phone.id}}">{{phone.name}}</a>
    <p>{{phone.snippet}}</p>
  </li>
</ul>
...
```

To dynamically generate links that will in the future lead to phone detail pages, we used the now-familiar double-curly brace binding in the `href` attribute values. In step 2, we added the `{{phone.name}}` binding as the element content. In this step the `{{phone.id}}` binding is used in the element attribute.

We also added phone images next to each record using an image tag with the ngSrc directive. That directive prevents the browser from treating the Angular `{{ expression }}` markup literally, and initiating a request to invalid url `http://localhost:8000/app/{{phone.imageUrl}}`, which it would have done if we had only specified an attribute binding in a regular `src` attribute (`<img src="{{phone.imageUrl}}">`). Using the `ngSrc` directive prevents the browser from making an http request to an invalid location.

Test

test/e2e/scenarios.js :

```
...
it('should render phone specific links', function() {
  var query = element(by.model('query'));
  query.sendKeys('nexus');
  element.all(by.css('.phones li a')).first().click();
  browser.getLocationAbsUrl().then(function(url) {
    expect(url.split('#')[1]).toBe('/phones/nexus-s');
  });
});
...
```

We added a new end-to-end test to verify that the app is generating correct links to the phone views that we will implement in the upcoming steps.

You can now rerun `npm run protractor` to see the tests run.

Experiments

Replace the `ng-src` directive with a plain old `src` attribute. Using tools such as Firebug, or Chrome's Web Inspector, or inspecting the webserver access logs, confirm that the app is indeed making an extraneous request to `/app/%7B%7Bphone.imageUrl%7D%7D` (or `/app/{{phone.imageUrl}}`).

The issue here is that the browser will fire a request for that invalid image address as soon as it hits the `img` tag, which is before Angular has a chance to evaluate the expression and inject the valid address.

Summary

Now that you have added phone images and links, go to step 7 to learn about Angular layout templates and how Angular makes it easy to create applications that have multiple views.

Previous | Live Demo | Code Diff | Next