In this final step, we will enhance our phonecat web application by attaching CSS and JavaScript animations on top of the template code we created before.

- Used the `ngAnimate` to enable animations throughout the application.
- Common `ng` directives automatically trigger hooks for animations to tap into.
- When an animation is found then the animation will run in between the standard DOM operation that is being issued on the element at the given time (e.g. inserting and removing nodes on `ngRepeat` or adding and removing classes on `ngClass`).

Workspace Reset Instructions ➤

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

Dependencies

The animation functionality is provided by Angular in the `ngAnimate` module, which is distributed separately from the core Angular framework. In addition we will use `jQuery` in this project to do extra JavaScript animations.

We are using Bower to install client side dependencies. This step updates the `bower.json` configuration file to include the new dependency:

```
{
"name": "angular-seed",
"description": "A starter project for AngularJS",
"version": "0.0.0",
"homepage": "https://github.com/angular/angular-seed",
"license": "MIT",
"private": true,
"dependencies": {
  "angular": "~1.3.0",
  "angular-mocks": "~1.3.0",
  "bootstrap": "~3.1.1",
  "angular-route": "~1.3.0",
  "angular-resource": "~1.3.0",
  "jquery": "~2.1.1",
  "angular-animate": "~1.3.0"
}
}
```

- `"angular-animate": "~1.3.0"` tells bower to install a version of the angular-animate component that is compatible with version 1.3.x.
- `"jquery": "2.1.1"` tells bower to install the 2.1.1 version of jQuery. Note that this is not an Angular library, it is the standard jQuery library. We can use bower to install a wide range of 3rd party libraries.

We must ask bower to download and install this dependency. We can do this by running:

```
npm install
```

Warning: If a new version of Angular has been released since you last ran `npm install`, then you may have a problem with the `bower install` due to a conflict between the versions of angular.js that need to be installed. If you get this then simply delete your `app/bower_components` folder before running `npm install`.

Note: If you have bower installed globally then you can run `bower install` but for this project we have preconfigured `npm install` to run bower for us.

How Animations work with `ngAnimate`

To get an idea of how animations work with AngularJS, please read the AngularJS Animation Guide first.

Template

The changes required within the HTML template code is to link the asset files which define the animations as well as the `angular-animate.js` file. The animation module, known as `ngAnimate`, is defined within `angular-animate.js` and contains the code necessary to make your application become animation aware.

Here's what needs to be changed in the index file:

app/index.html .

```
...
<!-- for CSS Transitions and/or Keyframe Animations -->
<link rel="stylesheet" href="css/animations.css">

...

<!-- jQuery is used for JavaScript animations (include this before angular.js) -->
<script src="bower_components/jquery/jquery.js"></script>

...

<!-- required module to enable animation support in AngularJS -->
<script src="bower_components/angular-animate/angular-animate.js"></script>

<!-- for JavaScript Animations -->
<script src="js/animations.js"></script>

...
```

> **Important:** Be sure to use jQuery version 2.1 or newer when using Angular 1.3; jQuery 1.x is not officially supported. Be sure to load jQuery before all AngularJS scripts, otherwise AngularJS won't detect jQuery and animations will not work as expected.

Animations can now be created within the CSS code (`animations.css`) as well as the JavaScript code (`animations.js`). But before we start, let's create a new module which uses the ngAnimate module as a dependency just like we did before with `ngResource` .

Module & Animations

app/js/animations.js .

```
angular.module('phonecatAnimations', ['ngAnimate']);
// ...
// this module will later be used to define animations
// ...
```

And now let's attach this module to our application module...

app/js/app.js .

```
// ...
angular.module('phonecatApp', [
  'ngRoute',

  'phonecatAnimations',
  'phonecatControllers',
  'phonecatFilters',
  'phonecatServices',
]);
// ...
```

Now, the phonecat module is animation aware. Let's make some animations!

Animating ngRepeat with CSS Transition Animations

We'll start off by adding CSS transition animations to our `ngRepeat` directive present on the `phone-list.html` page. First let's add an extra CSS class to our repeated element so that we can hook into it with our CSS animation code.

app/partials/phone-list.html .

```
<!--
Let's change the repeater HTML to include a new CSS class
which we will later use for animations:
-->
<ul class="phones">
<li ng-repeat="phone in phones | filter:query | orderBy:orderProp"
    class="thumbnail phone-listing">
  <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>
```

Notice how we added the `phone-listing` CSS class? This is all we need in our HTML code to get animations working.

Now for the actual CSS transition animation code:

app/css/animations.css

```
.phone-listing.ng-enter,
.phone-listing.ng-leave,
.phone-listing.ng-move {
  -webkit-transition: 0.5s linear all;
  -moz-transition: 0.5s linear all;
  -o-transition: 0.5s linear all;
  transition: 0.5s linear all;
}

.phone-listing.ng-enter,
.phone-listing.ng-move {
  opacity: 0;
  height: 0;
  overflow: hidden;
}

.phone-listing.ng-move.ng-move-active,
.phone-listing.ng-enter.ng-enter-active {
  opacity: 1;
  height: 120px;
}

.phone-listing.ng-leave {
  opacity: 1;
  overflow: hidden;
}

.phone-listing.ng-leave.ng-leave-active {
  opacity: 0;
  height: 0;
  padding-top: 0;
  padding-bottom: 0;
}
```

As you can see our `phone-listing` CSS class is combined together with the animation hooks that occur when items are inserted into and removed from the list:

- The `ng-enter` class is applied to the element when a new phone is added to the list and rendered on the page.
- The `ng-move` class is applied when items are moved around in the list.
- The `ng-leave` class is applied when they're removed from the list.

The phone listing items are added and removed depending on the data passed to the `ng-repeat` attribute. For example, if the filter data changes the items will be animated in and out of the repeat list.

Something important to note is that when an animation occurs, two sets of CSS classes are added to the element:

1. a "starting" class that represents the style at the beginning of the animation
2. an "active" class that represents the style at the end of the animation

The name of the starting class is the name of event that is fired (like `enter`, `move` or `leave`) prefixed with `ng-`. So an `enter` event will result in a class called `ng-enter`.

The active class name is the same as the starting class's but with an `-active` suffix. This two-class CSS naming convention allows the developer to craft an animation, beginning to end.

In our example above, elements expand from a height of **0** to **120 pixels** when items are added or moved, around and collapsing the items before removing them from the list. There's also a nice fade-in and fade-out effect that also occurs at the same time. All of this is handled by the CSS transition declarations at the top of the example code above.

Although most modern browsers have good support for CSS transitions and CSS animations, IE9 and earlier do not. If you want animations that are backwards-compatible with older browsers, consider using JavaScript-based animations, which are described in detail below.

Animating `ngView` with CSS Keyframe Animations

Next let's add an animation for transitions between route changes in `ngView`.

To start, let's add a new CSS class to our HTML like we did in the example above. This time, instead of the `ng-repeat` element, let's add it to the element containing the `ng-view` directive. In order to do this, we'll have to make some small changes to the HTML code so that we can have more control over our animations between view changes.

`app/index.html`.

```
<div class="view-container">
<div ng-view class="view-frame"></div>
</div>
```

With this change, the `ng-view` directive is nested inside a parent element with a `view-container` CSS class. This class adds a `position: relative` style so that the positioning of the `ng-view` is relative to this parent as it animates transitions.

With this in place, let's add the CSS for this transition animation to our `animations.css` file:

`app/css/animations.css`.

```
.view-container {
position: relative;
}

.view-frame.ng-enter, .view-frame.ng-leave {
background: white;
position: absolute;
top: 0;
left: 0;
right: 0;
}

.view-frame.ng-enter {
-webkit-animation: 0.5s fade-in;
-moz-animation: 0.5s fade-in;
-o-animation: 0.5s fade-in;
animation: 0.5s fade-in;
z-index: 100;
}

.view-frame.ng-leave {
-webkit-animation: 0.5s fade-out;
-moz-animation: 0.5s fade-out;
-o-animation: 0.5s fade-out;
animation: 0.5s fade-out;
z-index:99;
}

@keyframes fade-in {
from { opacity: 0; }
to { opacity: 1; }
}
@-moz-keyframes fade-in {
from { opacity: 0; }
to { opacity: 1; }
}
@-webkit-keyframes fade-in {
from { opacity: 0; }
to { opacity: 1; }
}

@keyframes fade-out {
from { opacity: 1; }
to { opacity: 0; }
}
@-moz-keyframes fade-out {
from { opacity: 1; }
to { opacity: 0; }
}
@-webkit-keyframes fade-out {
from { opacity: 1; }
to { opacity: 0; }
}

/* don't forget about the vendor-prefixes! */
```

Nothing crazy here! Just a simple fade in and fade out effect between pages. The only out of the ordinary thing here is that we're using absolute positioning to position the next page (identified via `ng-enter`) on top of the previous page (the one that has the `ng-leave` class) while performing a cross fade animation in between. So as the previous page is just about to be removed, it fades out while the new page fades in right on top of it.

Once the leave animation is over then element is removed and once the enter animation is complete then the `ng-enter` and `ng-enter-active` CSS classes are removed from the element, causing it to rerender and reposition itself with its default CSS code (so no more absolute positioning once the animation is over). This works fluidly so that pages flow naturally between route changes without anything jumping around.

The CSS classes applied (the start and end classes) are much the same as with `ng-repeat`. Each time a new page is loaded the `ng-view` directive will create a copy of itself, download the template and append the contents. This ensures that all views are contained within a single HTML element which allows for easy animation control.

For more on CSS animations, see the Web Platform documentation.

Animating `ngClass` with JavaScript

Let's add another animation to our application. Switching to our `phone-detail.html` page, we see that we have a nice thumbnail swapper. By clicking on the thumbnails listed on the page, the profile phone image changes. But how can we change this around to add animations?

Let's think about it first. Basically, when you click on a thumbnail image, you're changing the state of the profile image to reflect the newly selected thumbnail image. The best way to specify state changes within HTML is to use classes. Much like before, how we used a CSS class to specify an animation, this time the animation will occur whenever the CSS class itself changes.

Whenever a new phone thumbnail is selected, the state changes and the `.active` CSS class is added to the matching profile image and the animation plays.

Let's get started and tweak our HTML code on the `phone-detail.html` page first:

`app/partials/phone-detail.html` .

```
<!-- We're only changing the top of the file -->
<div class="phone-images">
  <img ng-src="{{img}}"
       class="phone"
       ng-repeat="img in phone.images"
       ng-class="{active:mainImageUrl==img}">
</div>

<h1>{{phone.name}}</h1>

<p>{{phone.description}}</p>

<ul class="phone-thumbs">
  <li ng-repeat="img in phone.images">
    <img ng-src="{{img}}" ng-mouseenter="setImage(img)">
  </li>
</ul>
```

Just like with the thumbnails, we're using a repeater to display **all** the profile images as a list, however we're not animating any repeat-related animations. Instead, we're keeping our eye on the ng-class directive since whenever the `active` class is true then it will be applied to the element and will render as visible. Otherwise, the profile image is hidden. In our case, there is always one element that has the active class, and, therefore, there will always be one phone profile image visible on screen at all times.

When the active class is added to the element, the `active-add` and the `active-add-active` classes are added just before to signal AngularJS to fire off an animation. When removed, the `active-remove` and the `active-remove-active` classes are applied to the element which in turn trigger another animation.

To ensure that the phone images are displayed correctly when the page is first loaded we also tweak the detail page CSS styles:

app/css/app.css

```
.phone-images {
background-color: white;
width: 450px;
height: 450px;
overflow: hidden;
position: relative;
float: left;
}

...

img.phone {
float: left;
margin-right: 3em;
margin-bottom: 2em;
background-color: white;
padding: 2em;
height: 400px;
width: 400px;
display: none;
}

img.phone:first-child {
display: block;
}
```

You may be thinking that we're just going to create another CSS-enabled animation. Although we could do that, let's take the opportunity to learn how to create JavaScript-enabled animations with the `animation()` module method.

app/js/animations.js .

```
var phonecatAnimations = angular.module('phonecatAnimations', ['ngAnimate']);

phonecatAnimations.animation('.phone', function() {

  var animateUp = function(element, className, done) {
    if(className != 'active') {
      return;
    }
    element.css({
      position: 'absolute',
      top: 500,
      left: 0,
      display: 'block'
    });

    jQuery(element).animate({
      top: 0
    }, done);

    return function(cancel) {
      if(cancel) {
        element.stop();
      }
    };
  }

  var animateDown = function(element, className, done) {
    if(className != 'active') {
      return;
    }
    element.css({
      position: 'absolute',
      left: 0,
      top: 0
    });

    jQuery(element).animate({
      top: -500
    }, done);

    return function(cancel) {
      if(cancel) {
        element.stop();
      }
    };
  }

  return {
    addClass: animateUp,
    removeClass: animateDown
  };
});
```

Note that we're using jQuery to implement the animation. jQuery isn't required to do JavaScript animations with AngularJS, but we're going to use it because writing your own JavaScript animation library is beyond the scope of this tutorial. For more on `jQuery.animate`, see the jQuery documentation.

The `addClass` and `removeClass` callback functions are called whenever a class is added or removed on the element that contains the class we registered, which is in this case `.phone`. When the `.active` class is added to the element (via the `ng-class` directive) the `addClass` JavaScript callback will be fired with `element` passed in as a parameter to that callback. The last parameter passed in is the `done` callback function. The purpose of `done` is so you can let Angular know when the JavaScript animation has ended by calling it.

The `removeClass` callback works the same way, but instead gets triggered when a class is removed from the element.

Within your JavaScript callback, you create the animation by manipulating the DOM. In the code above, that's what the `element.css()` and the `element.animate()` are doing. The callback positions the next element with an offset of `500 pixels` and animates both the previous and the new items together by shifting each item up `500 pixels`. This results in a conveyor-belt like animation. After the `animate` function does its business, it calls `done`.

Notice that `addClass` and `removeClass` each return a function. This is an **optional** function that's called when the animation is cancelled (when another animation takes place on the same element) as well as when the animation has completed. A boolean parameter is passed into the function which lets the developer know if the animation was cancelled or not. This function can be used to do any cleanup necessary for when the animation finishes.

Summary

There you have it! We have created a web app in a relatively short amount of time. In the closing notes we'll cover where to go from here.

Previous | Live Demo | Code Diff | Next