AngularJS IN ACTION

Lukas Ruebbelke

FOREWORD BY Martin Gontovnikas

SAMPLE CHAPTER

MANNING

AngularJS in Action

by Lukas Ruebbelke
with Brian Ford

Chapter 6

brief contents

6 Animations

This chapter covers

- How AngularJS handles animations
- Understanding the animation-naming convention
- The three types of animations
- Concrete examples of each type as it relates to Angello

6.1 Intro to animations

AngularJS was originally created as a framework to handle enterprise CRUD applications. With the introduction of the new animations API, AngularJS has broadened the possibilities to offer something for designers and developers alike.

The most powerful aspect of AngularJS is directives, and AngularJS animations are essentially class-based directives that have the power to harness complex animations with the addition of a single class to your markup.

The goal of this chapter is to show you the AngularJS animation events, the naming convention around those events, and the three types of animations you can do in AngularJS, with practical examples for each. We're not going to examine CSS3 animations or JavaScript animations in depth, but rather endeavor to lay a strong foundation that you can let your creativity run wild on.

6.1.1 How AngularJS handles animations

AngularJS animations can be distilled down to five events and a class-based naming convention. Once you've grasped the events at play and the naming convention, AngularJS animations fade into the background and the animations themselves take center stage.

There are three types of animations that you can create with AngularJS: CSS transitions, CSS animations, and JavaScript animations. Each type of animation is well suited for varying contexts, and we'll explore each of them later in the chapter.

AngularJS doesn't actually do any of the animations themselves, but simply provides the hooks for you to apply your own animations as you see fit. These hooks come in the form of *events*, and there are only five of them.

The five animation events are `enter`, `leave`, `move`, `addClass`, and `removeClass` (see table 6.1).

Table 6.1 The AngularJS animation furious five

Event	Function	Description
`enter`	`$animate.enter(element, parent, after, callback);`	Appends the `element` object after the `after` node or within the parent node and then runs the `enter` animation on the element
`leave`	`$animate.leave(element, callback);`	Runs the `leave` animation and then removes the element from the DOM
`move`	`$animate.move(element, parent, after, callback);`	Moves the `element` node either after the `after` node or inside of the `v` node and then runs the `move` animation on the element
`addClass`	`$animate.addClass(element, className, callback);`	Runs the `addClass` animation based on the `className` value and then adds the class to the element
`removeClass`	`$animate.removeClass(element, className, callback);`	Runs the `removeClass` animation based on the `className` value and then removes the class from the element

The `enter` and `leave` events are fired when a DOM element is added or removed from the DOM tree, respectively. The `move` event is fired when a DOM element changes position within the DOM tree. Last but not least, the `addClass` and `removeClass` events are fired when a class is added to or removed from an element, respectively.

6.1.2 The animation-naming convention

AngularJS animations are entirely class-based, which is a design decision that makes integration with third-party libraries easier. Even JavaScript animations follow a class-based naming convention for consistency.

The animation-naming convention follows a [class]-[event]-[state] pattern, as shown in figure 6.1. This figure indicates that we're dealing with a mute class that's being added and removed, as seen by `.mute-add` and `.mute-remove`. The animation defaults to the starting state and then progresses to the active state, as in "the class has been actively applied." The starting state is `.mute-add`, and `.mute-add-active` is the active or completed state.

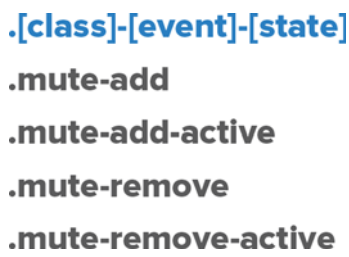

Figure 6.1 The animation-naming convention applied to directives

If your animations are defined within CSS and the events are triggered by an AngularJS directive such as `ng-if` or `ng-repeat`, then the class name will be prefixed with an *ng*, as in `ng-enter` and `ng-leave`.

6.1.3 Animations enable!

The most logical place to start from a pragmatic sense is with how you enable animations within your AngularJS application. AngularJS animations aren't part of the AngularJS core, and so you have to include that as a separate file. We'll use GreenSock Animation Platform (GSAP), which is a JavaScript animation framework. We want the TweenMax library, which contains everything GreenSock has to offer.

```
// client/assets/js/boot.js
{ file:
    '//cdnjs.cloudflare.com/ajax/libs/
    ➥  angular.js/1.3.3/angular-animate.min.js'
},
{ file:
    '//cdnjs.cloudflare.com/ajax/libs/gsap/latest/TweenMax.min.js'
},
```

GREENSOCK You can read more about GreenSock at http://www.greensock.com/gsap-js/.

Now that angular-animate.min.js has been included, we need to inject it as a sub-module into our application:

```
// client/src/angello/Angello.js
var myModule = angular.module('Angello', [
    //...
    'ngAnimate',
    //...
]);
```

With those two steps completed, we're ready to start adding animations to our application.

6.2 CSS transitions

The easiest animations to implement are CSS transitions. The ease of implementation comes from the fact that they're entirely CSS-based and much more concise to express than CSS animations.

We'll create a my-fade animation and apply it to a div that will trigger the animation when the div is added or removed from the DOM via `ng-if`. This animation will toggle the visibility of the story details in the right column when the Angello application is running in storyboard mode (see table 6.2).

Table 6.2 The animation-naming convention

Event	Starting CSS class	Ending CSS class	Directives that fire it
enter	`.ng-enter`	`.ng-enter-active`	`ngRepeat, ngInclude, ngIf, ngView`
leave	`.ng-leave`	`.ng-leave-active`	`ngRepeat, ngInclude, ngIf, ngView`
move	`.ng-move`	`.ng-move-active`	`ngRepeat`

6.2.1 Define the base transition

The first thing you need to do when constructing a CSS transition within AngularJS is set up the base transition. Because we're using `ng-if` to trigger the animation and the event is caused by an AngularJS directive, we need to define the classes for `ng-enter` and `ng-leave`:

```
/* client/assets/css/animations.css */
.my-fade-animation.ng-enter, .my-fade-animation.ng-leave {
    -webkit-transition: 0.5s linear all;
    -moz-transition: 0.5s linear all;
    -o-transition: 0.5s linear all;
    transition: 0.5s linear all;
}
```

In this code we define the transition for `ng-enter` and `ng-leave` on the my-fade animation to use linear easing that lasts for 0.5 seconds and applies to all properties.

6.2.2 Define the ng-enter transitions

The next step is to define the starting and stopping states for `ng-enter`. We'll start with an opacity of 0 and finish with an opacity of 1. This means that when the element is added, it'll start completely transparent and then fade in to full opacity.

```
/* client/assets/css/animations.css */
.my-fade-animation.ng-enter {
    opacity: 0;
}

.my-fade-animation.ng-enter.ng-enter-active {
    opacity: 1;
}
```

6.2.3 Define the ng-leave transitions

We'll now define the transition for `ng-leave`, which is usually the reverse of what you did for `ng-enter`. We'll start with an opacity of 1 and end with an opacity of 0:

```
.my-fade-animation.ng-leave {
    opacity: 1;
}

.my-fade-animation.ng-leave.ng-leave-active {
    opacity: 0;
}
```

For the sake of illustration, we've separated the `ng-enter` and `ng-leave` classes, but you could easily combine them for conciseness:

```
.my-fade-animation.ng-enter,
.my-fade-animation.ng-leave.ng-leave-active {
    opacity: 0;
}

.my-fade-animation.ng-leave,
.my-fade-animation.ng-enter.ng-enter-active {
    opacity: 1;
}
```

6.2.4 Making it move

Now that the CSS classes have been defined, it's a matter of attaching them to the DOM for use. Now you'll see what we mean when we say AngularJS transitions are essentially class-based directives that encapsulate animation functionality.

This is the HTML without the animation:

```
<!-- client/src/angello/storyboard/tmpl/storyboard.html -->
<div class="details">
    <!-- ... -->

    <div ng-if="storyboard.detailsVisible">
        <!-- ... -->
    </div>
</div>
```

This is the HTML with the animation:

```
<!-- client/src/angello/storyboard/tmpl/storyboard.html -->
<div class="details">
    <!-- ... -->

    <div ng-if="storyboard.detailsVisible" class="my-fade-animation">
        <!-- ... -->
    </div>
</div>
```

And so the only part left in this section is to actually toggle `ng-if`:

```
// client/src/angello/storyboard/controllers/StoryboardController.js
angular.module('Angello.Storyboard')
    .controller('StoryboardCtrl',
        function ($scope, $log, StoriesModel, UsersModel,
                  STORY_STATUSES, STORY_TYPES) {
        //...
        storyboard.detailsVisible = true;
        //...
        storyboard.setDetailsVisible = function (visible) {
            storyboard.detailsVisible = visible;
        };
});
```

In the `StoryboardCtrl`, we create a property on our `$scope` reference, called `detailsVisible`, that we'll use to bind `ng-if` to. We also create a method called `setDetailsVisible` that we use to set `detailsVisible` to true or false based on the value of the visible parameter.

In the HTML, we bind to `detailsVisible` via `ng-if="storyboard.detailsVisible"`:

```
<!-- client/src/angello/storyboard/tmpl/storyboard.html -->
<div class="details">
    <div class="details-nav">
        <div ng-if="!storyboard.detailsVisible">
            <button class="btn pull-left btn-default"
                    ng-click="storyboard.setDetailsVisible(true)">
                <span class="glyphicon glyphicon-arrow-left"></span>
            </button>
        </div>
        <div ng-if="storyboard.detailsVisible">
            <button class="btn pull-right btn-default"
                    ng-click="storyboard.setDetailsVisible(false)">
                <span class="glyphicon glyphicon-arrow-right"></span>
            </button>
        </div>
    </div>

    <div ng-if="storyboard.detailsVisible"
 class="my-fade-animation">
        <!-- ... -->
    </div>
</div>
```

Note that we also have two other divs that are being toggled based on the property of `detailsVisible`. If `detailsVisible` is true, then the button to set `detailsVisible` to false is shown, and vice versa.

We've now completed the functionality for attaching a CSS transition to our application. In the next section we'll cover another animation, but this time we'll do it with a CSS animation.

6.3 CSS animations

Now that you've seen AngularJS animations using CSS transitions, let's build on that with another animation using CSS animations. CSS animations tend to be more verbose than CSS transitions, but they're also significantly more powerful.

For this example, we'll do another fade animation, but this time with `ng-repeat`. If you recall, in table 6.2 `ng-repeat` has three events that we need to style for. These three events are `ng-enter`, `ng-leave`, and `ng-move`.

6.3.1 Define the base animation classes

The first thing we need to do is to define the base animation classes:

```
/* client/assets/css/animations.css */
.my-repeat-animation.ng-enter {
  -webkit-animation: 0.5s repeat-animation-enter;
  -moz-animation: 0.5s repeat-animation-enter;
  -o-animation: 0.5s repeat-animation-enter;
  animation: 0.5s repeat-animation-enter;
}

.my-repeat-animation.ng-leave {
  -webkit-animation: 0.5s repeat-animation-leave;
  -moz-animation: 0.5s repeat-animation-leave;
  -o-animation: 0.5s repeat-animation-leave;
  animation: 0.5s repeat-animation-leave;
}

.my-repeat-animation.ng-move {
  -webkit-animation: 0.5s repeat-animation-move;
  -moz-animation: 0.5s repeat-animation-move;
  -o-animation: 0.5s repeat-animation-move;
  animation: 0.5s repeat-animation-move;
}
```

We define our base CSS class as `my-repeat-animation` and then define animations for `ng-enter`, `ng-leave`, and `ng-move`. We then define the animation property with a 0.5-second duration and the appropriate keyframe for the animation.

> **VENDOR PREFIXES** The reason why CSS animations are so verbose is because you have to define the animation for every vendor prefix. Using a CSS preprocessor such as Sass or Less eliminates the need to type all of this out by hand.

6.3.2 Define the animation keyframes

Now that the base animation classes are defined, it's just a matter of defining the keyframes with the from and to states defined. Also, with CSS animations, it's not necessary to use the active convention that CSS transitions use.

The following is a fairly lengthy piece of code, but the pattern is easy to identify. The `ng-enter` animations go from 0 opacity to an opacity of 1, while `ng-leave` does the exact opposite, and `ng-move` goes from an opacity of 0.5 to an opacity of 1:

```
/* client/assets/css/animations.css */
@keyframes repeat-animation-enter {
  from {
    opacity:0;
  }
  to {
    opacity:1;
  }
}

@-webkit-keyframes repeat-animation-enter {
  from {
    opacity:0;
  }
  to {
    opacity:1;
  }
}

@-moz-keyframes repeat-animation-enter {
  from {
    opacity:0;
  }
  to {
    opacity:1;
  }
}

@-o-keyframes repeat-animation-enter {
  from {
    opacity:0;
  }
  to {
    opacity:1;
  }
}

@keyframes repeat-animation-leave {
  from {
    opacity:1;
  }
  to {
    opacity:0;
  }
}

@-webkit-keyframes repeat-animation-leave {
  from {
    opacity:1;
  }
  to {
    opacity:0;
  }
}
```

```
@-moz-keyframes repeat-animation-leave {
  from {
    opacity:1;
  }
  to {
    opacity:0;
  }
}

@-o-keyframes repeat-animation-leave {
  from {
    opacity:1;
  }
  to {
    opacity:0;
  }
}

@keyframes repeat-animation-move {
  from {
    opacity:0.5;
  }
  to {
    opacity:1;
  }
}

@-webkit-keyframes repeat-animation-move {
  from {
    opacity:0.5;
  }
  to {
    opacity:1;
  }
}

@-moz-keyframes repeat-animation-move {
  from {
    opacity:0.5;
  }
  to {
    opacity:1;
  }
}

@-o-keyframes repeat-animation-move {
  from {
    opacity:0.5;
  }
  to {
    opacity:1;
  }
}
```

6.3.3 Make it move

To show the portability of AngularJS animations, we can actually attach the same animation to two different `ng-repeat` instances with little fanfare:

```
<!-- client/src/angello/storyboard/tmpl/storyboard.html -->
<div class="list-area-animation"
     ng-class="{'list-area-expanded':!storyboard.detailsVisible}">
    <div class="list-wrapper">
        <ul class="list my-repeat-animation"
            ng-repeat="status in storyboard.statuses">
            <h3 class="status">{{status.name}}</h3>
            <hr/>
            <li userstory
                ng-repeat="story in storyboard.stories
                ➥  | filter:{status:status.name}"
                drag-container="story"
                ➥  mime-type="application/x-angello-status"
                drop-container=""
                ➥  accepts="['application/x-angello-status']"
                class="story my-repeat-animation"
                ng-click="storyboard.setCurrentStory(story)">

                <!-- ... -->
            </li>
        </ul>
    </div>
</div>
```

We attach it to the `ul` items, which render the status columns that the user stories are organized into, and to the `li` items that represent the user stories themselves.

We asserted at the beginning of the chapter that AngularJS animations are just a matter of a handful of events and a naming convention. We believe that this section really proved it in the sense that we haven't introduced any new concepts other than the CSS animation syntax itself. It was to a point anticlimactic, because by now some of these elements should start to feel familiar.

6.4 JavaScript animations

The final type in the AngularJS animations triad is JavaScript animations. For this example we'll toggle the position of the details section by animating it on and off the screen. We'll accomplish this by dynamically attaching a `details-visible` class using `ng-class`.

You can see the details section shown in figure 6.2, and in figure 6.3 it's in its hidden state.

You can use any JavaScript animation library, but for our example we'll use TweenMax, which is a part of the GreenSock Animation Platform. TweenMax is an incredibly powerful and feature-rich animation library that performs well on desktop and mobile browsers.

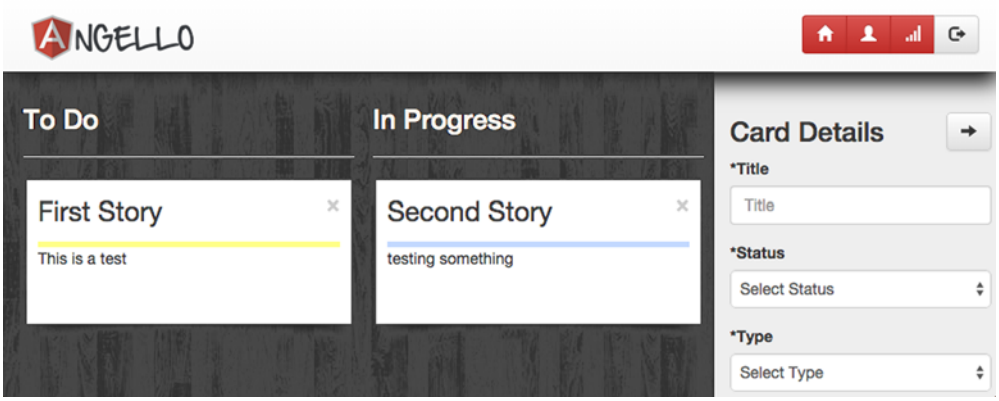

Figure 6.2 **Details shown**

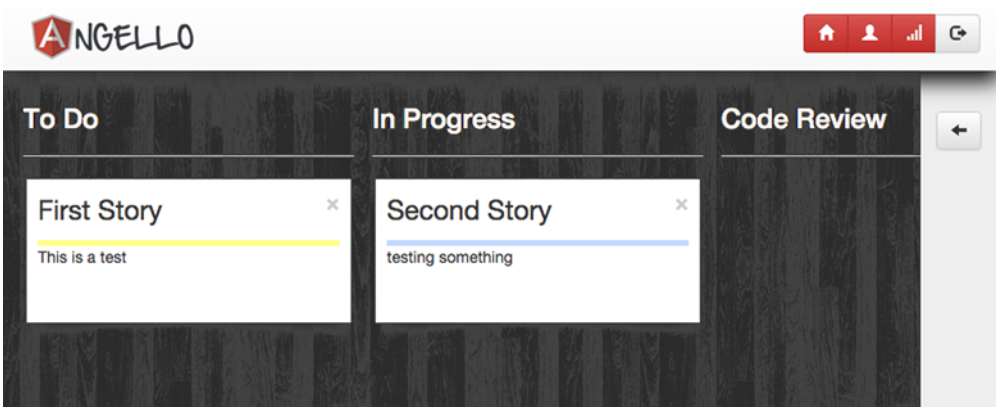

Figure 6.3 **Details hidden**

6.4.1 Defining the JavaScript animation

JavaScript animations are defined using the `animation` service:

```
// client/src/angello/app/animations/DetailsAnimation.js
angular.module('Angello.Common')
    .animation('.details-animation',
        function () {
            //...
        });
```

Defining the animation is similar to defining an AngularJS service or controller. The only difference is that the animation name is class-based, so instead of `details-animation`, it's `.details-animation`.

6.4.2 The JavaScript animation events

Now that the animation has been defined, we need to actually configure it to handle the animation events. Because we trigger the animation with `ng-class`, the two events we want to listen to are `addClass` and `removeClass`:

```
// client/src/angello/app/animations/DetailsAnimation.js
angular.module('Angello.Common')
    .animation('.details-animation',
        function () {
            return {
                addClass: function (element, className, done) {
                    //...
                },
                removeClass: function (element, className, done) {
                    //...
                }
            };
        });
```

The event handlers are defined inline to the return object. The three parameters that each handler receives are `element`, `className`, and `done`. The `element` is the DOM element that the event was triggered on, `className` is the name of the class that triggered the event, and `done` is the callback function that needs to be called when the animation is complete.

6.4.3 The JavaScript animation class

It's possible to have more than one animation defined on an element, and so it's necessary to perform some logic to only act if the class that triggered the event is the one you've defined:

```
// client/src/angello/app/animations/DetailsAnimation.js
angular.module('Angello.Common')
    .animation('.details-animation',
        function () {
            return {
                addClass: function (element, className, done) {
                    if (className == 'details-visible') {
                        //...
                    }
                    else {
                        done();
                    }
                },
                removeClass: function (element, className, done) {
                    if (className == 'details-visible') {
                        //...
                    }
                    else {
                        done();
                    }
                }
            };
        });
```

This is why, in the preceding code, we check to see if `className` is equal to `details-visible`, and if it's not then we call the `done` callback.

6.4.4 TweenMax

Now that we know that we're dealing with the `details-visible` class specifically, it's time to add in the TweenMax code to actually do the animation work:

```
// client/src/angello/app/animations/DetailsAnimation.js
angular.module('Angello.Common')
    .animation('.details-animation',
        function () {
            return {
                addClass: function (element, className, done) {
                    if (className == 'details-visible') {
                        TweenMax.to(element, 0.5,
                        ➥  {right: 0, onComplete: done });
                    } else {
                        done();
                    }
                },
                removeClass: function (element, className, done) {
                    if (className == 'details-visible') {
                        TweenMax.to(element, 0.5, {
                            right: -element.width() + 50,
                            onComplete: done
                        });
                    } else {
                        done();
                    }
                }
            };
        });
```

When `details-visible` is added, we use TweenMax to animate the element to an absolute position of 0 pixels to the right. When `details-visible` is removed, we use TweenMax to animate it off the screen by setting the `right` property to the negative value of `element.width()` plus 50 pixels so the Show button is still visible.

6.4.5 Making it move

The final piece to make the details-animation work is to add it to the DOM and set `ng-class` to toggle the `details-visible` class.

The following is the same code we used earlier, but with a few small additions to the outer div. We've added `details-animation` to the class attribute, so now the animation has a hook into the DOM. And we're also dynamically adding or removing the `details-visible` class based on the value of `detailsVisible` with `ng-class="{'details-visible':storyboard.detailsVisible}"`:

```
<!-- client/src/angello/storyboard/tmpl/storyboard.html -->
<div class="details details-animation"
     ng-class="{'details-visible':storyboard.detailsVisible}">
```

```
    <div class="details-nav">
        <div ng-if="!storyboard.detailsVisible">
            <button class="btn pull-left btn-default"
                    ng-click="storyboard.setDetailsVisible(true)">
                <span class="glyphicon glyphicon-arrow-left"></span>
            </button>
        </div>
        <div ng-if="storyboard.detailsVisible">
            <button class="btn pull-right btn-default"
                    ng-click="storyboard.setDetailsVisible(false)">
                <span class="glyphicon glyphicon-arrow-right"></span>
          </button>
        </div>
    </div>

    <div ng-if="storyboard.detailsVisible" class="my-fade-animation">
    <!-- ... -->
    </div>
</div>
```

The resulting animation works in conjunction with the CSS transition animation we defined so that the details elements fade out as the details section slides off the screen, and fade in as the details section slides back in.

MANUALLY TRIGGERED ANIMATIONS You can manually trigger your own animations using the `$animate` service. See http://docs.angularjs.org/api/ngAnimate.$animate for more details.

6.5 Testing

Because animations target the visual aspect of our application more than the functionality aspect, we usually leave animations out of our unit tests. But if you'd like to know how to test animations, visit http://www.yearofmoo.com/2013/08/remastered-animation-in-angularjs-1-2.html#testing-animations.

Full-page animations

Here's a super easy way to get full-page transitions: set an animation class on the tag with the `ng-view` directive defined on it. In the context of Angello, if you were to find the `<div ng-view=""></div>` element in `index.html` and add `class="my-fade-animation"`, each route would automatically start fading in and out!

6.6 Best practices

Memorize the naming conventions for AngularJS animations. Seriously, you'll be an animation alchemist. You'll be able to throw together pro animations in no time at all.

Use CSS transitions/animations when possible. We like to use CSS transitions and animations for simpler visuals, and only build them using JavaScript when they involve multiple animations and/or complex transitions. That way, we can keep our styles in our CSS files and let the JavaScript focus on the business logic.

6.7 Summary

Now that you have three examples under your belt, we hope that it's easy to identify the event and naming-convention patterns that surround AngularJS animations. AngularJS has proven itself time and time again to be a great framework for doing functional things, but animations bring some fashion to that functionality with an easy-to-use API that leverages all CSS and JavaScript to do any kind of animation you can imagine. Let's do a quick recap:

- There are five animation event hooks in AngularJS: `enter`, `leave`, `move`, `addClass`, and `removeClass`.
- You learned what triggers each type of event.
- You discovered the naming conventions that make animations tick.
- You viewed examples of CSS transitions, CSS animations, and animations using JavaScript.
- You got a quick introduction to TweenMax and how it interacts with AngularJS.

WEB DEVELOPMENT/JAVASCRIPT

AngularJS IN ACTION

Lukas Ruebbelke

AngularJS is a JavaScript-based framework that extends HTML, so you can create dynamic, interactive web applications in the same way you create standard static pages. Out of the box, Angular provides most of the functionality you'll need for basic apps, but you won't want to stop there. Intuitive, easy to customize, and test-friendly, Angular practically begs you to build more interesting apps.

AngularJS in Action teaches you everything you need to get started with AngularJS. As you read, you'll learn to build interactive single-page web interfaces, apply emerging patterns like MVVM, and tackle key tasks like communicating with back-end servers. All examples are supported by clear explanations and illustrations along with fully annotated code listings.

What's Inside

- Get started with AngularJS
- Write your own components
- Best practices for application architecture
- Progressively build a full-featured application
- Covers AngularJS 1.3
- Sample application updated to the latest version of Angular

This book assumes you know at least some JavaScript. No prior exposure to AngularJS is required.

Lukas Ruebbelke is a full-time web developer and an active contributor to the AngularJS community.

"Learn how to build an exciting application from top to bottom with AngularJS."
—From the Foreword by Martin Gontovnikas
Developer Advocate, Auth0

"The coolest way to create a web application I have ever seen!"
—William E. Wheeler
ProData Computer Services

"The best introduction to AngularJS so far."
—Gregor Zurowski, Sotheby's

"Packed with practical examples and best practices."
—Ahmed Khattab
Cisco Services

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/AngularJSinAction

ISBN 13: 978-1-617291-33-3
ISBN 10: 1-617291-33-1
54499
9 781617 291333

MANNING $44.99 / Can $51.99 [INCLUDING eBOOK]