AngularJS IN ACTION

Lukas Ruebbelke

FOREWORD BY Martin Gontovnikas

SAMPLE CHAPTER

MANNING

AngularJS in Action

by Lukas Ruebbelke
with Brian Ford

Chapter 1

brief contents

Part 1

Get acquainted with AngularJS

Welcome to the world of AngularJS! Part 1 of this book provides a high-level overview of AngularAS, as well as a gentle introduction to AngularJS through building a simple—yet useful—web application.

In chapter 1 we introduce all the major pieces of AngularJS and discuss what they do and how they fit together. We also introduce a simplified version of the book's sample application and build it from the ground up. In chapter 2 we discuss how to assemble your AngularJS applications using best practices to make sure that your applications are maintainable and extensible.

By the end of part 1, you should have a good grasp of the major pieces of AngularJS and be conversational in how they all work together. If you work through how to build the sample application, you'll also have a good foundation for beginning your AngularJS journey.

AngularJS is a very dynamic and quickly evolving framework, so please reference the repository for the latest code samples as well as bonus content: https://github.com/angularjs-in-action. You can also find the code for the first project here: https://github.com/angularjs-in-action/angello-lite.

1 Hello AngularJS

This chapter covers

- Why you need AngularJS
- How AngularJS makes your life easier
- Understanding AngularJS from a high level
- Building your first AngularJS application

There was a time many internet years ago when any kind of logic within a web page had to be sent to the server for processing and then re-rendered as an entirely new web page. This "call and refresh" arrangement made for a disjointed user experience, which was only exacerbated when network latency was especially high.

The entire paradigm was upended with the introduction of `XMLHttpRequest` and the ability to make asynchronous calls to the server without actually having to refresh the page. This made for a much more coherent user experience because the user could perform a task that required a remote call and still interact with the application as the call was being made and processed. This is where the first wave of JavaScript frameworks landed and managed to prove that working with JavaScript could be done in a mostly sane way and no one was going to lose life or limb.

Most people would agree that jQuery won that round, partially because jQuery did such a good job of abstracting away all of the insanity surrounding browser variations, and allowed developers to use a single, simplified API to build websites. The next frontier was to make websites behave and operate as if they were actual applications; this ushered in an entirely new set of challenges. For instance, jQuery has done an exceptional job of providing tools to manipulate the DOM, but it offers no real guidance on how to organize your code into an application structure. We've all heard horror stories of how a jQuery "application" ballooned out into a monstrosity that could barely be maintained, much less extended.

This desperate need to write large, maintainable JavaScript applications has given birth to a JavaScript framework renaissance. In the last couple of years, a slew of frameworks has burst onto the scene, with many of them quietly fading off into oblivion. But a few frameworks have proven themselves to be solid options for writing large-scale web applications that can be maintained, extended, and tested. One of the most popular, if not *the* most popular, frameworks to emerge is AngularJS from Google.

AngularJS is an open-source web application framework that offers quite a bit to a developer through a stable code base, vibrant community, and rich ecosystem. Let's identify some of the high-level advantages of using AngularJS before we get into some of the more technical details of the framework.

1.1 Advantages of using AngularJS

In this section we'll take a quick look at what makes AngularJS so great.

AN INTUITIVE FRAMEWORK MAKES IT EASY TO ORGANIZE YOUR CODE

As previously stated, there's a pressing need to be able to organize your code in a way that promotes maintenance, collaboration, readability, and extension. AngularJS is constructed in such a way that code has an intuitive place to live, with clear paths to refactor code when it has reached a tipping point. Do you have code that needs to provide information on how a user interface is supposed to look and behave? There's a place for that. Do you have code that needs to contain a portion of your domain model and be available for the rest of the application to use? There's a place for that. Do you need to programmatically perform DOM manipulation? There's even a sane place for that as well!

TESTABLE CODE MAKES IT EASIER TO SLEEP AT NIGHT

Testable code isn't going to win any awards for being the most exciting feature of a framework, but it's the unsung hero of any mature framework. AngularJS was written from the ground up to be testable, and likely this feature, along with the design decisions that came from this commitment, has played a huge role in the adoption of AngularJS. How do you actually know if your application works? The fact that it hasn't broken yet is a flimsy answer, as it's only a matter of time before that black swan shows up at your door.

You can never entirely mitigate against bugs, but you can truly eliminate certain possibilities through rigorous testing. A framework that is conducive to writing testable

code is a framework that you're going to write tests in. And when you write tests, you'll spend less time looking over your shoulder wondering when everything is going to come crashing down. You'll be able to go to bed at night and not have to worry about a 2 a.m. call from DevOps that something has gone awry and you need to fix it immediately.

TWO-WAY DATA BINDING SAVES YOU HUNDREDS OF LINES OF CODE

Two-way data binding is the supermodel of the feature set. Hundreds of years ago, when we were writing jQuery applications, you would've had to use jQuery to query the DOM to find a specific element, listen for an event, parse the value of the DOM element, and then perform some action on that value. In AngularJS, you simply have to define a JavaScript property and then bind to it in our HTML, and you're done. There are obviously some variations to this scenario, but it's not uncommon to hear of jQuery applications being rewritten and thousands of lines of JavaScript just disappearing.

By cutting out all of the boilerplate code that was previously required to keep our HTML and JavaScript in sync, you're able to accomplish more work in less time with significantly less effort. This gives you more time to do more of what you love.

TEMPLATES THAT ARE HTML MEANS YOU ALREADY KNOW HOW TO WRITE THEM

HTML is an inherently limited language that was designed to facilitate layout and structure, not complex interactions. In other words, it wasn't created to live in the world of the modern web application as we know it now. Some frameworks try to overcome this limitation by abstracting out HTML entirely into strings or some preprocessor dialect. The problem is that HTML is actually good as a declarative mechanism and there's this annoying reality about HTML—pretty much everyone knows it.

If you're working on a large team, there's a good chance that you're going to have a UI/UX contributor who'll be responsible for generating your HTML templates. It's important to leverage a workflow and skill set that they're already familiar with, and AngularJS makes this a breeze. AngularJS embraces HTML while giving developers the ability to overcome its limitations by extending it to do whatever it is we need.

DATA STRUCTURES THAT ARE JUST JAVASCRIPT MAKE INTEGRATION REALLY EASY

On the flip side, being able to work with Plain Old JavaScript Objects (POJOs) makes integrating with other technologies incredibly easy. By consuming and emitting JavaScript without having to wrap and unwrap it in proprietary framework mechanisms, you're able to consume data from other sources much more efficiently.

You can render JSON models from the server and instantly consume them in AngularJS when the application bootstraps. You can also take the model that you're working with and pass it off to another technology—such as an application server—without having to transform it at all.

There are some pretty interesting features of AngularJS that are fairly academic in nature; we've tried to outline a few major points of how AngularJS makes our lives easier in a very practical sense. At the end of the day, having a framework that allows us to write stable code quickly and efficiently so that we have more time and energy to do other meaningful things is a tool that we want to use!

1.2 The AngularJS big picture

We'll introduce AngularJS from a 10,000-foot view and lay the foundation for what we'll reinforce throughout the entire book (see table 1.1). If you reach the end of the book and you have a solid grasp of figure 1.1 and how all the pieces fit together, we'll have succeeded as authors. If you've absorbed these concepts in such a way that these pieces form a vocabulary in which you start to articulate and express ways to solve your own problems, then we'll have succeeded in a spectacular way!

Table 1.1 AngularJS at a glance

Component	Purpose
Module	Modules serve as containers to help you organize code within your AngularJS application. Modules can contain sub-modules, making it easy to compose functionality as needed.
Config	The config block of an AngularJS application allows for configuration to be applied before the application actually runs. This is useful for setting up routes, dynamically configuring services, and so on.
Routes	Routes allow you to define ways to navigate to specific states within your application. They also allow you to define configuration options for each specific route, such as which template and controller to use.
Views	The view in AngularJS is what exists after AngularJS has compiled and rendered the DOM with all of the JavaScript wiring in place.
`$scope`	`$scope` is essentially the glue between the view and controller within an AngularJS application. With the introduction of the *controller-as* syntax, the need to explicitly use `$scope` has been greatly reduced.
Controller	The controller is responsible for defining methods and properties that the view can bind to and interact with. As a matter of best practice, controllers should be lightweight and only focus on the view they're controlling.
Directive	A directive is an extension of a view in AngularJS in that it allows you to create custom, reusable elements that encapsulate behavior. You can think of directives as components or decorators for your HTML. Directives are used to extend views and to make these extensions available for use in more than one place.
Service	Services provide common functionality to an AngularJS application. For instance, if you have data that more than one controller needs, you would promote that data to a service and then make it available to the controllers via the service. Services extend controllers and make them more globally accessible.

Although we'll get into each of these AngularJS mechanisms in considerable depth in the following chapters, we wanted to introduce you to the major players at the outset so you would have a foundation to build on.

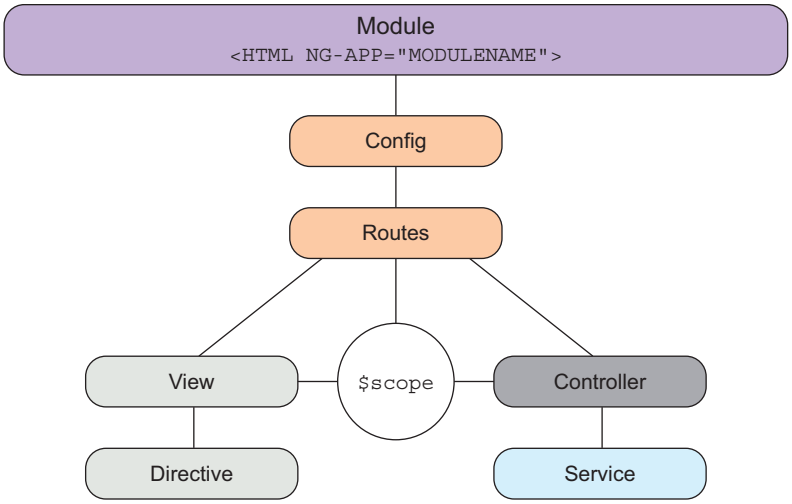

Figure 1.1 **The AngularJS big picture**

1.3 Build your first AngularJS application

Now that you have the AngularJS game pieces on the table, how do you actually use them to put together something useful? This is the perfect time to build something easy with AngularJS. You'll get your feet wet by building a scaled-down version of the sample application for the book; and, in the process, you'll see how these AngularJS pieces fit together without getting too advanced. True to the title of this book, you'll learn AngularJS by seeing it in action and assembling examples that tie into a larger, fully functional application.

The sample application for the book is called Angello; it's a Trello clone that's used to manage user stories. What do we mean by a Trello clone? Well, as some of you may know, Trello is a project management tool that is web-based and founded on a technique that was originally popularized by the Japanese car manufacturer Toyota in the 1980s. Units of work in a project are represented by items—stories, if you will—that can be moved to different positions on a board corresponding to each story's state of progress. The board itself represents the project. We'll properly introduce Angello in the next chapter, but you can see the main screen of Angello in figure 1.2 and the lite version of Angello in figure 1.3. The completed source code for Angello Lite can be found at https://github.com/angularjs-in-action/angello-lite. Download the latest version to your local machine by following the instructions in the README.md on that page.

Over the course of the book, we'll be building out Angello, which you can see in figure 1.2. In the left portion of the screen, the items of work are represented by the white boxes named First Story and Second Story, and the flow of progress is represented by the columns To Do, In Progress, Code Review, QA Review, and Verified, moving from left to right across the screen. As work progresses, each box is moved by

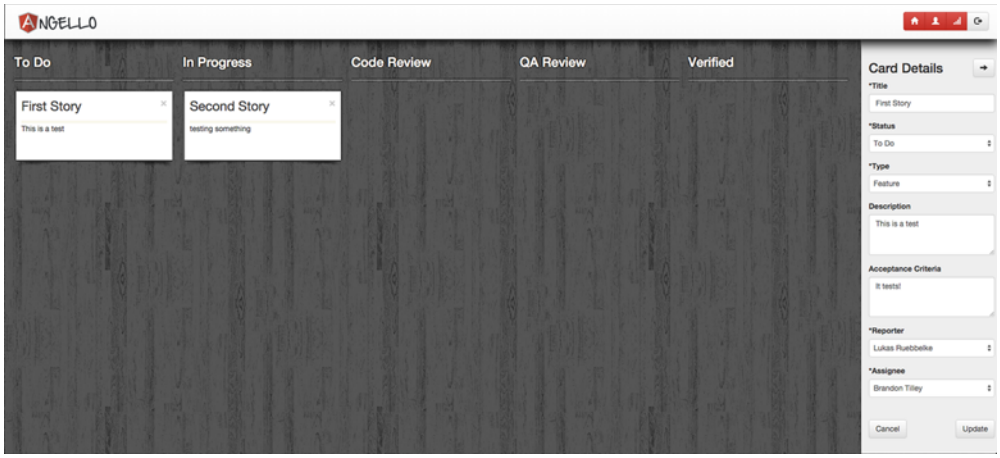

Figure 1.2 **Angello**

drag-and-drop to represent its state of completion within the project. The details of each work item, each story, can be viewed on the right of the screen. As you may have guessed, each story in Angello represents a unit of computer software that can pass from inception to completion as the project unfolds.

We'll start out by building a simplified version, which you can see in figure 1.3.

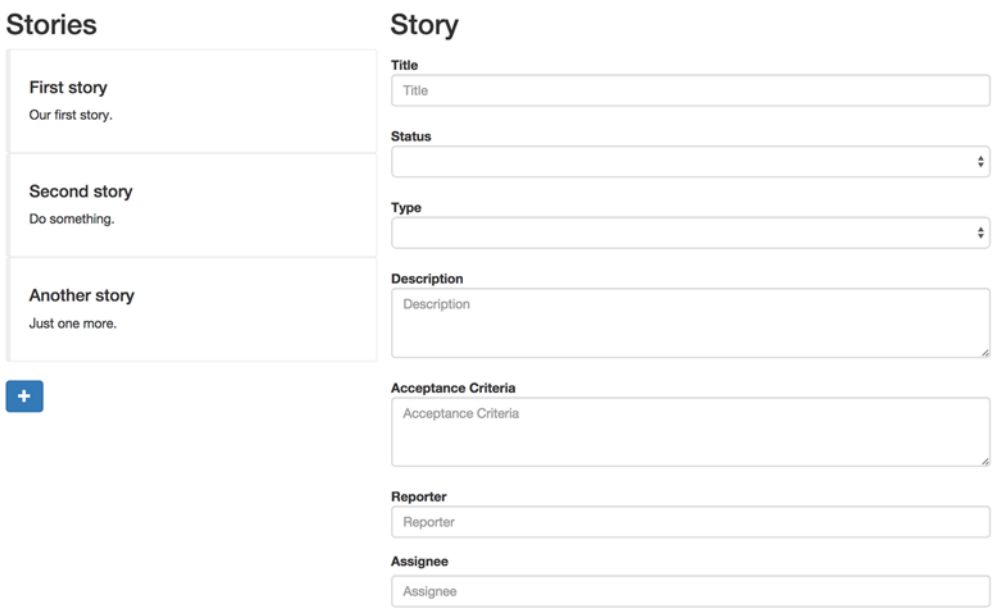

Figure 1.3 **Angello Lite**

ANGELLO LITE Because you're pulling files from a CDN, you'll need to run Angello Lite from a web server. There are a few ways to do this, but one of the easiest ways is to use the npm package `serve`.

The steps for installing Angello Lite are as follows:

- Install Node.js. You can find all of the information to do that at http://nodejs.org/.
- Install the serve package by running `npm install -g serve` from the command line.
- Download Angello Lite from GitHub, using the URL given above, and place it on your local machine in a directory named angello-lite.
- Navigate to the angello-lite directory from the command line and run `serve`.
- Go to http://localhost:3000 in your browser to see the application.

Angello Lite is a simplified version of the Angello app that we'll develop from chapter 2 onwards. All the data you add here will be stored in memory alone and not persisted, so when you reload the page, it will be lost. To display the details of an existing story, click the box showing its title and description on the left of the screen. Its details will appear on the right. Use these text boxes and drop-down lists to alter or augment the story and these updates will remain for as long as the page is loaded in the browser. To create a new story, click the plus sign on the left. A new title and description box will appear. Place a new title and description, along with other data, into the text boxes on the right and see how the title and description in the summary box change in real time as you type.

As a high-level overview, figure 1.4 shows the pieces that we'll be building out as they relate to the big picture. We'll start by constructing the module and then build

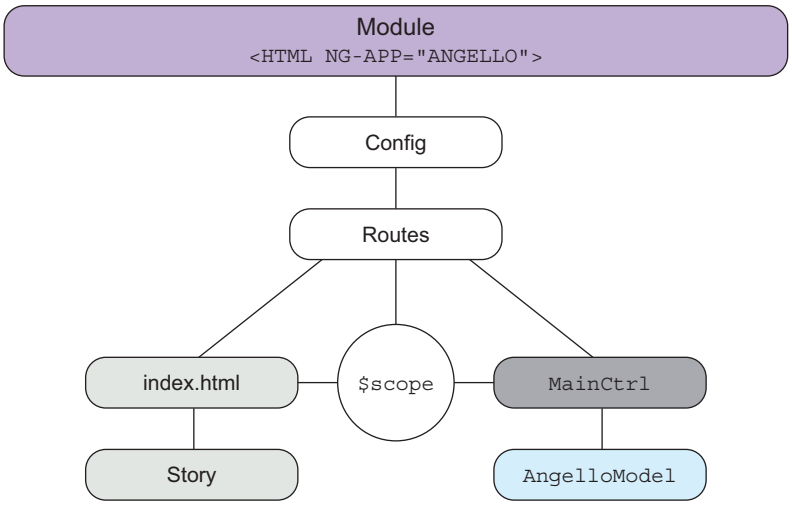

Figure 1.4 Angello and the big picture

out the view and controller via `index.html` and `MainCtrl`, respectively. From there we'll introduce services by creating `AngelloModel` and a directive by creating the `story` directive.

We won't go through Angello Lite line by line, but we'll sufficiently cover the highlights so that you'll be conversant in what's happening. By the time you finish this chapter, you'll at least be able to fake your way through an AngularJS dinner party!

The thing to keep in mind when building out Angello Lite is that it is a master-detail interface, which shows up in almost every single web application in one form or another. Understanding how to put together a master-detail interface is one of the best foundations for learning how to build web applications in general.

1.3.1 The module

Modules in AngularJS serve as containers to help you organize your application into logical units. Modules tell AngularJS how an application is configured and how it's supposed to behave. You can see how it fits into the big picture in figure 1.5.

In our code, we'll create a module called `Angello` and assign it to the `myModule` variable:

```
// app.js
var myModule = angular.module('Angello', []);
```

The second parameter is an array that accepts other sub-modules to provide additional functionality, if necessary. It's considered best practice to divide features into sub-modules and then inject them into the main application module. This makes it much easier to move a module around as well as test it.

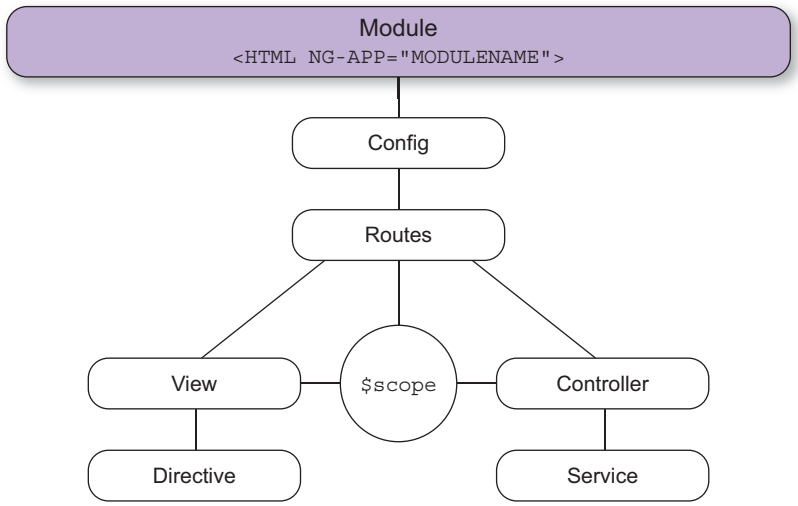

Figure 1.5 The module

You can now define the necessary components for Angello Lite on the `myModule` property. For instance, here we define two services called `AngelloHelper` and `AngelloModel`, as well as a controller called `MainCtrl` and a directive called `story`:

```
// app.js
var myModule = angular.module('Angello', []);
myModule.factory('AngelloHelper', function() { });
myModule.service('AngelloModel', function() { });
myModule.controller('MainCtrl', function() { });
myModule.directive('story', function() { });
```

With the Angello module defined and all of the necessary pieces stubbed out, you can now bootstrap your AngularJS application, using Angello as a starting point. The easiest way to bootstrap an AngularJS application is to add the `ng-app` attribute to the HTML element where you want the AngularJS application to reside. In our case, we want our application to use the entire page, so we'll add `ng-app="Angello"` to the `html` tag. This will bootstrap AngularJS with the `Angello` module:

```
<!-- index.html -->
<html ng-app="Angello">
```

From here, we'll flesh out the remaining pieces with commentary on how they work.

1.3.2 Views and controllers

One of the most critical concepts to understand when learning AngularJS is the separation of state from the DOM. AngularJS is officially a Model-View-Whatever (MVW) framework—"Whatever" being whatever pattern helps you be most productive. For the sake of conversation, let's assume that AngularJS follows the Model-View-View-Model (MVVM) design pattern, as established in figure 1.6.

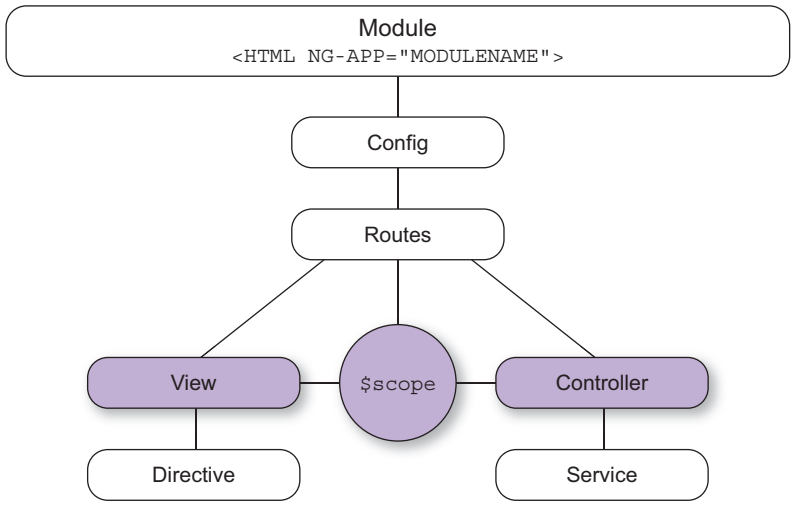

Figure 1.6 Views and controllers

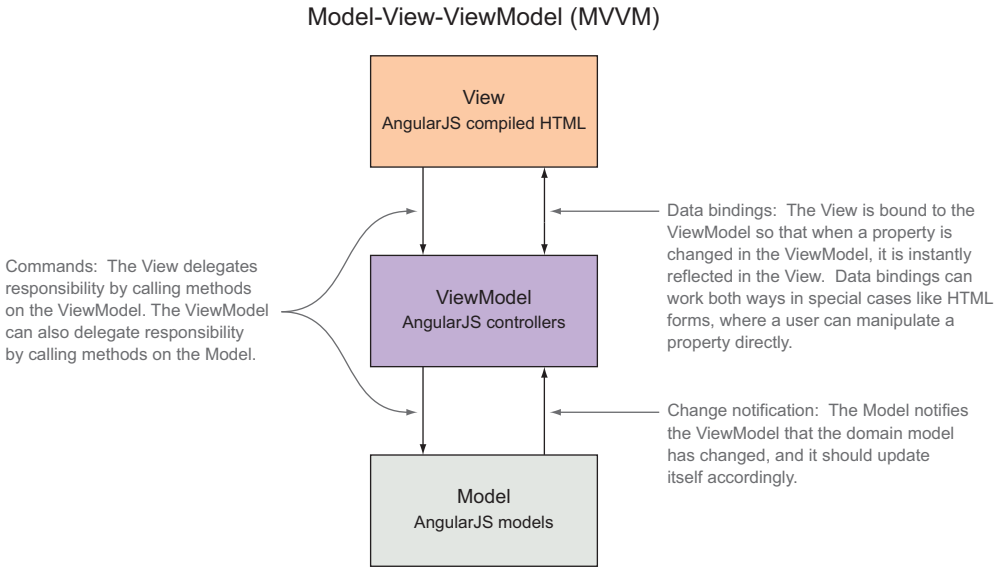

Figure 1.7 Model-View-ViewModel

We'll get to the Model portion in the services section; but for now let's focus on the View and ViewModel parts of this pattern. The View in the MVVM pattern is the view in AngularJS (naturally), and the controller plays the part of the ViewModel, as you can see in figure 1.7.

The controller is responsible for providing state for the view to bind to and commands that the view can issue back to the controller to do units of work. This frees up the view from having to maintain state (since it only has to display whatever state the controller is in) and it frees up the view from having to do any work (as the view always defers to the controller).

To illustrate this in action, we'll first instantiate the `MainCtrl` by adding it to the DOM with the `ng-controller` directive. We use the controller-as syntax by declaring the controller to be `MainCtrl as main`, which means that we'll reference the `MainCtrl` as `main` within the HTML file going forward:

```
<!--index.html-->
<div ng-controller="MainCtrl as main">
</div>
```

Making a property available for binding within the view is just a matter of declaring it on the controller. For instance, you could declare a property on `MainCtrl` such as `this.title` and then immediately bind to it in the view using double curly braces like this: `<h1>{{main.title}}</h1>`. Any changes to the `title` property would instantly be reflected in the DOM. Binding to a simple string property is fairly simplistic, so let's do something more in-depth and bind to an actual collection. We'll create an array containing multiple `story` objects and define it as `stories` on `MainCtrl`:

```
// app.js
myModule.controller('MainCtrl', function() {
    var main = this;

    //...
    main.stories = [
        {
            title: 'First story',
            description: 'Our first story.',
            criteria: 'Criteria pending.',
            status: 'To Do',
            type: 'Feature',
            reporter: 'Lukas Ruebbelke',
            assignee: 'Brian Ford'
        },
        {
            title: 'Second story',
            description: 'Do something.',
            criteria: 'Criteria pending.',
            status: 'Back Log',
            type: 'Feature',
            reporter: 'Lukas Ruebbelke',
            assignee: 'Brian Ford'
        },
        {
            title: 'Another story',
            description: 'Just one more.',
            criteria: 'Criteria pending.',
            status: 'Code Review',
            type: 'Enhancement',
            reporter: 'Lukas Ruebbelke',
            assignee: 'Brian Ford'
        }
    ];
    //...
});
```

THIS Per common convention, we like to store a reference to the top-level `this` object in case we need it later; `this` has a habit of changing context based on function level scope. We also like to name the reference to `this` the same name we declare the *controller-as* in the view—as in `main` for `MainCtrl as main`. This makes it easier to read and connect the dots as you jump between the HTML and the JavaScript.

We'll display the `main.stories` collection as a list of items that comprise the master portion of the master-detail view. The first thing we need to do is to repeat over the `main.stories` array and create an individual display element for each story item in the array. The `ng-repeat` directive accomplishes this by going through every item in the `main.stories` collection and creating a copy of the HTML element it was declared on and all of its child elements. So by declaring `ng-repeat="story in main.stories"` on our `callout` div, we're essentially telling AngularJS to loop through `main.stories` and reference each individual item as `story`—which we can bind to within the child elements:

```
<!-- index.html -->
<div ng-controller="MainCtrl as main">
    <div class="col-md-4">
        <h2>Stories</h2>
        <div class="callout"
             ng-repeat="story in main.stories"
             ng-click="main.setCurrentStory(story)">
            <h4>{{story.title}}</h4>
            <p>{{story.description}}</p>
        </div>
    </div>
</div>
```

Each `story` object has a title and description property, which we can bind to via `{{story.title}}` and `{{story.description}}`. AngularJS is really good at providing context within each template instance, so we don't have to worry about the `story` instance getting overwritten with each iteration. This is important when we want to start doing things like `ng-click="main.setCurrentStory(story)"`, in which the specific instance of `story` matters a great deal.

This is a perfect segue for moving beyond binding to properties and learning how to bind to expressions. You can also make methods available to the view by declaring them on the controller. For instance, we'll define a method called `main.createStory` that pushes a new `story` object into the `main.stories` array:

```
// app.js
myModule.controller('MainCtrl', function() {
    var main = this;

    //...
    main.createStory = function() {
        main.stories.push({
            title:'New Story',
            description:'Description pending.',
            criteria:'Criteria pending.',
            status:'Back Log',
            type:'Feature',
            reporter:'Pending',
            assignee:'Pending'
        });
    };
    //...
});
```

Now that `createStory` is defined on the `MainCtrl`, it's available to be called from the view. We can then call `main.createStory` from the view by using `ng-click` on an anchor tag:

```
<!-- index.html -->
<div ng-controller="MainCtrl as main">
    <div class="col-md-4">
        <h2>Stories</h2>
        <div class="callout"
             ng-repeat="story in main.stories"
             ng-click="main.setCurrentStory(story)">
```

```
            <h4>{{story.title}}</h4>
            <p>{{story.description}}</p>
        </div>
        <br/>
        <a class="btn btn-primary" ng-click="main.createStory()">
            <span class="glyphicon glyphicon-plus"></span>
        </a>
    </div>
</div>
```

Using a ViewModel inverts the application flow that traditionally existed in jQuery-style applications. In jQuery, you would've queried the DOM and attached an event listener. When that event fired, you would try to interpret the event and parse the DOM for state so that you could perform some imperative operation. This forces a tight coupling between the HTML and the JavaScript that drives it. By introducing a ViewModel, you're able to break this relationship. The controller no longer is responsible for listening to the view, but rather the view is responsible for issuing specific commands to the controller that it operates on.

MVVM A full-fledged discussion on the MVVM pattern is outside the scope of this book, but we recommend reading up on it here: http://en.wikipedia.org/wiki/Model_View_ViewModel. Having a clear separation between declarative markup and imperative logic is conducive to better, more stable code that is easier to test.

1.3.3 Services

If controllers should be lightweight and specific to the view for which they're responsible, what happens if two controllers need to share the same information? Controllers definitely shouldn't know about each other. So what happens if some piece of information starts in one controller and you realize that it needs to be available in another controller? The answer to these questions is an AngularJS *service*. You promote (extract) the common data from the controller and make it available to the entire application by exposing it via a service. As you can see in figure 1.8, this is the Model portion of the Model-View-ViewModel pattern.

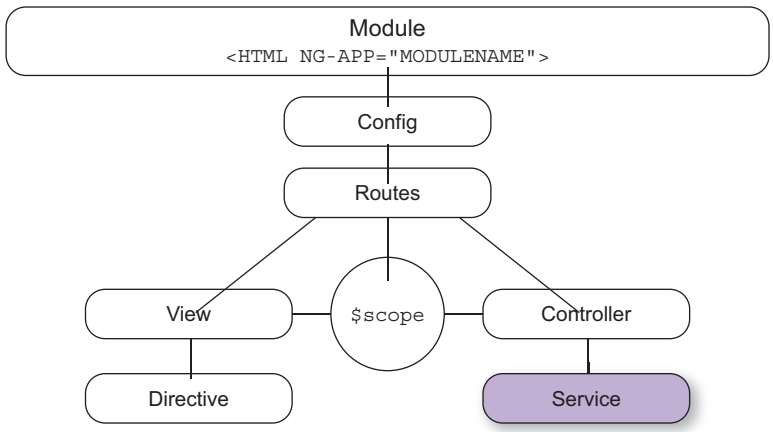

Figure 1.8 Services

In the previous section, we populated our `stories` collection directly in the `MainCtrl`, but now we'll promote that collection to the `AngelloModel` service and make it available to the `MainCtrl`. We'll declare a `stories` property in `AngelloModel` and then populate it with the same collection we used in `MainCtrl`:

```
// app.js
myModule.service('AngelloModel', function() {
    var service = this,
        stories = [
            {
                title: 'First story',
                description: 'Our first story.',
                criteria: 'Criteria pending.',
                status: 'To Do',
                type: 'Feature',
                reporter: 'Lukas Ruebbelke',
                assignee: 'Brian Ford'
            },
            //...
        ];

    service.getStories = function() {
        return stories;
    };
});
```

From here, we'll make `AngelloModel` available to `MainCtrl` by passing it into the constructor function as a parameter. AngularJS uses *dependency injection* (DI) to provide dependencies where they're needed. Dependency injection is fancier in name than it is in implementation. AngularJS can detect that we need an instance of `AngelloModel`, so it creates an instance for use and injects it into `MainCtrl`, thus fulfilling that dependency:

```
// app.js
myModule.controller('MainCtrl', function(AngelloModel) {
    var main = this;

    //...
    main.stories = AngelloModel.getStories();
    //...
});
```

We can now populate `main.stories` by assigning to it the return value of `AngelloModel.getStories()`. The beauty of this arrangement is that `MainCtrl` is completely oblivious to where the stories data is coming from or how we got it. We'll get into this in much greater depth in the following chapters, but we could've just as easily made a remote server call and populated the data that way.

One more quick example, and then we'll move on to directives. AngularJS services aren't just for storing common state, but also for sharing common functionality, such as utility functions. For example, we needed a very general `buildIndex` method to take an array and create an index based on a property parameter. That way we

wouldn't have to loop over the array every single time we needed to find an object in it. This type of a general function could be used in more than one place, so we put it in an `AngelloHelper` service:

```
// app.js
myModule.factory('AngelloHelper', function() {
    var buildIndex = function(source, property) {
        var tempArray = [];

        for(var i = 0, len = source.length; i < len; ++i) {
            tempArray[source[i][property]] = source[i];
        }

        return tempArray;
    };

    return {
        buildIndex: buildIndex
    };
});
```

This kind of finely grained code is much easier to maintain and test because it is in isolation and doesn't depend on some other runtime context.

1.3.4 Directives

Directives are one of the most powerful and exciting things in AngularJS. In fact, you've already seen some directives in action in the previous sections. For instance, when you attach `ng-click` to an element, you're using a built-in AngularJS directive to augment the behavior of that specific element. When you add `ng-app` or `ng-controller` to the page, you're using AngularJS directives to provide new behavior to an otherwise static page. In figure 1.9, you can see how they fit into the big picture.

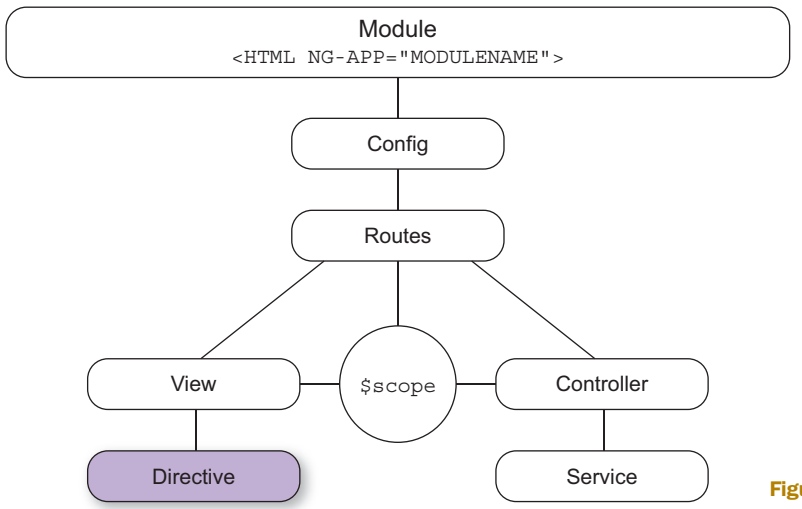

Figure 1.9 Directives

We'll introduce a simple directive to Angello Lite to get our feet wet. We'll create a `story` directive that represents a story in the page. Directives are defined similarly to controllers and services in that they take a name and a function defining their behavior:

```
// app.js
myModule.directive('story', function(){
    return {
        scope: true,
        replace: true,
        template:'<div><h4>{{story.title}}</h4>
        ➥ <p>{{story.description}}</p></div>'
    }
});
```

The function returns a *directive definition object* (DDO) that defines how the directive is supposed to be configured. We're telling AngularJS that each instance of this directive gets a new scope and that the template we defined replaces the element the directive was defined on. The template markup should be familiar, as it's the same code we used previously to display the `title` and `description` of the current story item.

Now that we've defined the directive, we update the HTML in our page to use a `story` tag and not a `div` tag. A story tag? Is there even such a thing? There is now!

```
<div ng-controller="MainCtrl as main">
    <div class="col-md-4">
        <h2>Stories</h2>
        <story class="callout"
               ng-repeat="story in main.stories"
               ng-click="main.setCurrentStory(story)">
        </story>
        <!-- ... -->
    </div>
</div>
```

Even though this is a small example of how to extend HTML to do new things by using directives, we want to start your wheels turning on what kind of applications you would write if you could create HTML tags and attributes to do whatever you wanted.

1.4 Summary

This concludes our tour of Angello Lite. Now that you've seen most of the major players from figure 1.1 in action, we'll spend the rest of the book digging into these concepts a lot deeper and in a more useful context as we start to work with Angello.

Let's do a quick review before we finish this chapter and head into the next one:

- AngularJS was created as a framework designed to make it easier to write and organize large JavaScript applications.
- AngularJS was written from the ground up to be testable; as a result, it's much easier to write clean, stable code that can scale.

- Data binding saves you from writing literally thousands—if not tens of thousands—of lines of code because you no longer have to write tedious boilerplate around DOM events.
- Because AngularJS templates are just HTML, it's easy to leverage existing skill sets to build out UIs in AngularJS.
- Plain Old JavaScript Objects make integration with other systems much easier.
- AngularJS modules are containers for organizing your application.
- Views in AngularJS are compiled and rendered HTML with a controller attached.
- A controller is the ViewModel for the view and is responsible for providing data and methods for the view to bind to.
- Services encapsulate and provide common functionality in an AngularJS application.
- A directive is a custom component or attribute that extends HTML to do new and powerful things.

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]