Introduction

Introduction

Hello, I'm Joe Eames. Welcome to Pluralsight's course Angular: The Big Picture. In order to understand the purpose and value of Angular, you must first understand something about JavaScript. JavaScript is a programming language that was written in 2 weeks and was hated by developers for years because it was slow, difficult to work with, and frustrating to debug. But in recent years, tooling around JavaScript has improved significantly and companies like Google, Apple, and Microsoft have put millions of dollars into making JavaScript fast. This has made it not only possible, but even enjoyable to write large applications to run in the browser. Enter Angular: A framework for writing manageable, clean, full-featured applications in the browser with JavaScript. Angular was originally written by Misko Hevery. His intention was to create a way for designers to be able to do HTML design using components with actual functionality. As he got further and further along in the project, it became obvious that the framework he was building was actually a great way for developers to build entire sites. As a result, Angular was born. Here's a view of a very early version of the Angular website. Today, the Angular team has over 20 core members and dozens more extended team members, with many more occasional contributors, and tens of thousands of developers using it every day to build line-of-business applications. This course is meant for those who wish to understand Angular from a high level. We will be covering topics that will give a more meaningful and general understanding of what Angular is. In essence, we will be focusing on the forest and not on the trees. So if you are a developer who wants to backup from the details and understand Angular from a higher level, this course is for you. Or, if you are an architect or manager who is evaluating which front-end framework to use, this course is for you as well. This course is meant to be valuable to both developers and non-developers. As such, even though we may look at some code, you won't need any experience in JavaScript programming to understand and get the most from this course. This course is divided up into seven different modules. The first one, the introduction, is what you're watching now. Next, we will discuss the benefits and features of Angular. This module will focus on what makes Angular great, and show you why it is a compelling option for most web applications. After that, we will take a look into some of the more interesting architectural constructs and decisions of Angular. The next module, we'll look at a few technologies that play particularly well with Angular, and allow you to do some really amazing things. After that, we'll get our hands a bit dirty and look at the warts of Angular. We'll be looking at the more confusing aspects, and learn about the kinds of things you will need to look out for. The next module will be a comparison of Angular with other popular technologies for building web applications. This will show you how Angular separates itself from the crowd. Finally, we'll take a look at the future of Angular, both the future of Angular 1 and Angular 2, and what will happen to Angular 1 as Angular 2 comes on the scene. One of the main questions people have about Angular is why to use it over another solution. There are many front-end frameworks out there, from simply using jQuery, all the way to full-featured frameworks that run on both the front and back-end. Later on in this course, we will take a closer look at the advantages and benefits of Angular, and compare using Angular to many different alternatives. But here, we will talk about the high level points of Angular. Angular is a complete solution. The vast majority of use cases could be handled by Angular without resorting to any third-party tools or libraries. Angular is easy to learn. A Hello World application takes only moments to write, and the overwhelming amount of tutorials and documentation, not to mention vast quantities of developers who have significant experience with Angular, make it easy for any team to get up and running in no time. It even abstracts away much of the difficulties of working in JavaScript so that teams of full stack or server-side developers will find its paradigm more familiar than many other alternatives. Angular is extremely popular, which makes it easy to find answers and gives you the confidence to know that it is solid, and stable, and performant, and that you are unlikely to run into any bugs when using it. Angular is slightly opinionated, which means that it will give you guidance on how best to write applications while being versatile enough to adapt to your particular situation. Angular is open source and has been contributed to by hundreds of industry experts, but although it's open source, it is also backed by one of the most well-respected companies in the computer industry, Google. They are paying the salaries of dozens of developers who maintain and grow Angular, and they have a vested interest in seeing it succeed. Finally, Angular is highly testable, which is my personal favorite reason to use Angular. In fact, testing is such a high priority for the Angular Team that some awesome testing tools, including Karma and Protractor, have actually been produced by the team and are being used by developers to test applications that aren't Angular applications. Put all these items together, and it becomes pretty obvious why Angular is dominating the front-end landscape currently. Now that we've taken a very high level look at Angular's purpose and history, let's zoom way in and actually put together an Angular application in less than a couple of minutes. Please follow along, even if you've never written a line of code before, you will easily be able to build a simple Angular application. The easiest way to work with Angular is using the popular web development sandbox tool, Plunker. The URL is shown here. We'll start by clicking the Launch the Editor button, right here; next, we'll go over to this popular packages window, and up here in the Search packages I'm going to type in angular and hit Enter, and that makes sure that Angular bubbles up to the top in case it's not at the top, then I'm going to click on this More button over here and that's going to open up this drop-down. At this point, I'm going to select any version of 1.x; I don't want any of the 2.0 versions. I want a 1.x version. 1.4 is fine, 1.3 is fine. I'm personally just going to select 1.3.15 because it's the last release version, but really any version of 1.x is going to work just fine. Once I click that, then I can select this Add button, and we'll see over here in our code page on line 5 it's added that reference. Once I've done that, I'm done with this window. Now we'll come over here and click on the script.js file, which opens up our script file, and inside of here we'll write the following code. This line of code creates an Angular module, which is the container for all code in Angular. Next, we're going to create a controller, which allows us to manipulate the page. This is the controller function, which takes in two parameters. The first parameter is the name of our controller, which we called main, and the second parameter is a function, and that function takes in a single parameter which is $scope. Then we create our function body using curly braces, and inside of our function body we're going to say $scope.message equals the words Hello World. This message is piece of data that we created ourselves, but using Angular we could easily get that data from our own server, or from some third-party service like iTunes or Netflix, or even a third-party business partner. Now we're going to go back into our index.html page, and we're going to tell Angular that we want it to run on this page. We do that by adding an attribute to our html tag called ng-app, and set it equal to the string app, which matches the name of the module that we created back in our script.js file, and then on the body tag, we're going to add an attribute called ng-controller, and that's going to be set to the name of the controller we created, which was main. And now that we've done that, we can go ahead and do some interesting things in our page. In this case, we're just going to display the message that we created, Hello World, and we do that by inserting two curly braces, and then the name of that piece of data, which was message, and the closing curly braces. And that completes the coding of our Angular application, and we can now run it by clicking on the eye up here in our Plunker window; and you can see, in just this brief exercise, how simple it is to get up and running with Angular.

Angular Benefits and Features

Introduction

Hello, I'm Joe Eames. In this module we will be discussing the benefits and features of Angular. These range from the straightforward features of testability and two-way binding, to more vague concepts such as cohesion and guidance. Each of the benefits and features I mention in this module will provide a compelling case by themselves as to the value of Angular, but overall, there is one reason why Angular is a great framework to use for your development, and that is a reduction of cost due to a reduction of time spent by developers and other staff. Regardless of whether we are speaking of learning curve, staff recruiting and retention, development time of a project, or the maintenance costs of a project, using Angular reduces the time and, therefore, costs of building web applications, and that is usually the primary reason to choose a technology.

Code Reduction

The first benefit we will talk about is code reduction. This is possibly the most single, compelling, and important one we can list. Angular reduces the amount of code that developers write over most other options. Now we're not talking about a 10% or a 20% reduction in code, or even a 50% reduction in code; using Angular over solutions like jQuery or Backbone, or anything handwritten, will give you a code savings of 80% to 90%. That means that with Angular, it would take you 1000 to 2000 lines of code to produce the same functionality as 10,000 lines of code would take you in jQuery or Backbone, or a similar technology. This is an amazing statistic, and I have personally seen this on many projects that I've converted from an older technology to Angular. Those lines of code almost directly translate to savings in time and cost. That means that you save time, not only when the code is written, but also when developers are debugging, and later on when the project is maintained with new features. At each phase, you save the time and effort that fewer lines of code gives. Today there are other frameworks, which give similar savings over the more basic options, so this benefit isn't exclusive to Angular, but it is one of the most objective and easy-to-understand benefits.

Two Way Binding and Forms

One of the most commonly used features in a web app is forms. Here, Angular not only handles the things you would normally expect, like validation and value collection in a simple and straightforward manner, but there are a couple of specific features of Angular, which make it particularly easy to make your forms interactive. These are Angular's binding features. This includes the typical one-way binding where you can specify a part of a page that will display the value of some part of your data model, and when that data model changes, that part of the page is updated to reflect the new value. Angular also includes a feature called two-way binding, where any value you put into an input control, such as a text box, is watched and Angular can send your changes to the model, which can then propagate those changes to other bindings. Let's look at an example of how this works. Here, I've got a simple form which lets us transfer money. You can see that there's a single input box, which asks for how much money I'm going to transfer, then underneath that, it tells you what my current balance is. But this form goes one step further to tell me how much my balance will be after I transfer the amount of money I indicate. So let's go in and fill in a value. Instead of 0, we'll type in 10, and you can see that the balance after the transfer has instantly been changed. And as I change this amount, the balance after transfer continues to update itself, such that it always stays in sync no matter what I type into this box. This is the power of two-way binding. Now what's even more amazing is how much code it took to create this example. Using jQuery or JavaScript might take 20 or more lines of code, but using Angular, this is all the code that it takes to produce this behavior. You can see that after the initialization code, we are essentially talking about two lines. These kinds of features make it easy to have interactive forms that behave exactly the way you want them to.

Cohesive Solution

One of the less objective features of Angular is the fact that it is a cohesive, complete solution. This means that Angular provides all the typical pieces of a web application solution. It's not just one piece of the puzzle. That doesn't mean that you will never have any need to use a third-party library with Angular, but instead, all the major pieces of a front-end application are handled consistently by Angular. It doesn't just handle forms, or application structure, or routing, Angular handles all these pieces and more. From server communication to binding, from accessibility to routing, Angular does all of this. One of the main benefits is that you and your team won't get caught in analysis paralysis trying to determine exactly which routing library to use from 10 different choices, or how to handle server communication from the 5 different options. With Angular, everything is handled, and when you need something that one of the built-in pieces doesn't handle, it's clear when that's the case, and it's simple to replace it with a more complex component. This leads us to another related benefit of Angular. Angular provides gentle guidance on application structure without being overly prescriptive. That means that each solution you build with Angular won't look completely different from an architectural standpoint. That also means that one Angular application tends to look like another application from the code level, which makes it easier for developers to work on multiple projects, or get reassigned from one project to another. This benefit is difficult to quantify, but can radically effect ramp-up times and learning curves, and drastically reduced maintenance costs on a project, which makes this one of Angular's benefits that shouldn't be underestimated. Anyone who has looked underneath the covers of the browser's API, knows that it is riddled with inconsistencies and difficult-to-learn paradigms. Although this is improving over time, it's still a concern as developers are expected to be more and more effective on the front-end. Angular gives the benefit of hiding many of these difficult and unwieldy APIs by allowing developers to do the same things with more expressive syntax and higher level APIs. For example, if you wish to listen to a click event on a button, you have to write the code shown here in raw JavaScript, whereas with Angular you simply need to add the event listener on the HTML using the syntax shown here, and then somewhere in your code you create a click handler. The Angular code is significantly less complex, and easier to learn and remember. With this and other similar changes, Angular makes it much easier to deal with the DOM in the browser to make your web applications interactive and maintainable. These previous three benefits mean something else, because Angular is a complete solution with a reasonable amount of guidance, and which hides a lot of the ugliness of the browser environment, it will seem familiar to full stack developers. If you have a team of web developers who have mostly been using a full stack solution like JSP, or ASP.NET, or Rails, Angular is a great way to add more front-end functionality without putting your developers down a rabbit hole of learning an ecosystem which is completely foreign. Although it will definitely be a new technology and require a learning curve, full stack developers will find more in common with Angular than they will with many other options that are more a la carte, and which require a more detailed and technical understanding of browser-side JavaScript in order to get past the vagaries of client-side JavaScript applications.

Accessibility and Internationalization

A nice feature of Angular is its support for accessibility and internationalization. In both cases the support is good, and with little effort you can get some basic support into your applications, but in both cases, to make your application fully support these technologies you will have to put in some additional effort. First, let's talk about accessibility. Recently the Angular Team made accessibility much easier to implement in your applications with the ngAria module. With this module, by just adding one script to your application, you are immediately using a lot of the ARIA attributes, which make applications more friendly to accessibility devices like screen readers. So in literally less than 1 minute, you can make your application a heck of a lot more accessible than it already is. Of course, full accessibility is something that requires planning, but ngAria gets you started and makes it much easier to make your application accessible. Next, we'll talk about internationalization. With almost no effort you can make your numbers, dates, and currency, internationalized and localized. Let's look at how that's done. This code example shows how easy it is to bind to different kinds of data and format them. What's great about Angular is that formatting a piece of data as currency, a date, or a number, also localizes it, so that if your site is being viewed by someone from a different locale, then the data will be displayed in the correct format for that locale. So like accessibility, you get a good start on internationalization for free. But if you want to fully internationalize your application, then you'll need to translate all your strings and use the correct string for the current language and locale. Doing this with Angular is no problem, but you will want to use a third-party library. There are several available and I won't go specifically into any of them, but they all make doing internationalization with Angular straightforward and easy.

Popularity

One of the incontrovertible benefits of Angular is its popularity. Angular has quickly come to dominate the front-end web space, and is now the JavaScript framework of choice for more companies and development teams than any other technology. This, by itself, has several benefits. First, learning Angular is easier because many people are creating example applications, blog posts, and courses like this one. This also means that it's easier to find someone you know who has experience with Angular to teach you in person. It's also easier to find user groups or bdevs that are focused on Angular. Second, it's easier to find qualified staff because many developers have worked with Angular. Third, because of its popularity, it's easier to hire and retain Angular developers because developers want jobs that are doing the technologies that they are interested in; Angular definitely fits this bill. Finally, its popularity is a benefit because that means that its longevity is assured. You know that support for Angular isn't going anywhere, even if the entire Angular Team at Google were hit by a bus, the community at large would ultimately support it because it has so much momentum and so many companies have a vested interest in Angular.

Testability

Testing is an area where Angular stands head-and-shoulders above its peers. Angular was designed with testing in mind right from the start. In fact, Misko Hevery, the creator of Angular, was actually working at Google as a bit of a testing consultant when he invented Angular. Angular makes testing any of its components very easy through both unit testing and end-to-end, or functional testing. Its unit testing is facilitated by a built-in mocking component called ngMock that will let you isolate Angular from your back-end server. End-to-end testing is facilitated by a tool created by the Angular Team called Protractor. This tool lets you actually run Angular in a real browser and automate its functionality to make sure that things work the way you believe they should. In both cases, you can ease running of your unit tests with a tool called Karma, which was also developed by the Angular Team. This tool makes it easy to run your tests in multiple browsers all at the same time, and it has become extremely popular for automating testing, even on projects which don't use Angular. All in all, testing is one of the places where Angular really shines.

Summary

This module covered some of the features and benefits of Angular. We mentioned code reduction, two-way binding and forms, the fact that Angular is a cohesive solution, accessibility and internationalization, popularity, and finally, testability. Although this is a long list, it is hardly exhaustive. There are definitely too many other features and benefits of Angular to list in any reasonable amount of time. In the end, you can rest assured that Angular is a framework that many, many companies have evaluated in-depth and chosen as their front-end framework of choice.

Angular Architecture

Introduction

Hello, I'm Joe Eames. In this module we will look at the architecture of Angular to gain a better understanding of how Angular does its magic. Unlike many other sources you may find on Angular, this module will not discuss the hideable components; that is because this is not an introductory course to Angular. If you want to learn the difference between services and controllers, there are many places to get that rudimentary information. The purpose of this course is to give you the information you need to understand Angular well enough to either make a decision on using it or migrating to it for a web application, or, if you are already using it, understand it better from a high level standpoint. So instead of learning the basic concepts, we will be looking at some interesting architectural choices that Angular has made, which may impact your application. There are a few fairly novel things that Angular does in addition to reusing some tried and true architectural constructs. In this module, we will first look at Angular's overall architecture. In essence, Angular is an HTML compiler, and we will see how that works from a high level. Then, we will look at how Angular does change detection, and finally how it uses dependency injection to make consuming business logic components simple and easy.

HTML Compiler

Angular, at its core, is an HTML compiler, meaning everything starts from the HTML of your site, not from the code you write. You can write code all day long, but without the proper pieces in your HTML, not only will nothing happen, but none of your code will even get loaded because Angular doesn't even instantiate the components you create until it finds a need for that driven by the HTML. Let's look at how this works in practice. As Angular gets loaded, it processes, or compiles the HTML on your page. Since the HTML's essentially a tree structure, Angular processes this tree from top to bottom, starting with the HTML element. As it processes the tree, Angular is looking for something very specific. It is looking for a DOM node with a special attribute on it, the ng-app attribute. In our example, this attribute is on the HTML node. Once Angular finds this attribute, it knows that you want Angular to own that particular piece of the DOM and all its children. In this case, that would be the entire document. But it doesn't have to be. This is a way to let Angular run side-by-side with other MVC libraries with which it might otherwise normally collide. After the ng-app attribute is located, Angular goes into its next phase where it is looking for another attribute. This one is the ng-controller attribute. In our example, that lives on the body element. This attribute lets Angular know that a particular component of yours will now control that portion of the page. Again, you are free to use multiple controllers. There is a slight variation on this when including routing, but the net result is the same. Now that an ng-controller attribute has been found, Angular reads the value of that ng-controller attribute. In our example here, that is the word main. At this point, Angular will go into your code and look for a controller that you have created with the name of main. Once it finds that controller, it loads it up and can now process the rest of the page. Angular now goes into high gear, where it is looking not for a particular attribute, but instead is compiling every element looking for all kinds of things. A few examples of this would be a button element with an ng-click attribute. This lets Angular know that you want to handle the click event on that button. Another example would be an h1 tag with some content that includes the double curly braces. This tells Angular that you want to bind to a particular value and display it here. One last example of something that Angular will process is called a directive. This is where some of the magic of Angular shines through. A directive is a way for you to combine display and logic in a single component. For example, if you need a tile that will display the picture, name, and Twitter handle of a user, and also make it so that if clicked on, that user is selected so that you can then perform some operation, like send a message to them, all this functionality and the associated display can be bundled up into a directive. And what's great about that, is that you actually tell Angular about it in your HTML. You do that using custom HTML elements. We'll use an example of an HTML element called user-info. The presence of this element, which isn't a standard HTML element, but instead a custom one, causes Angular to process your component and replace that element with a piece of display and functionality that you have specified previously. Just to give you a more concrete idea of how this might look, here's our example in actual HTML. I've simplified this for this demonstration, but this is the essence of how Angular processes and compiles HTML.

Detecting Changes and the Digest

The next architectural topic we will look at in Angular is its change detection algorithm. This is one of the places where Angular diverges greatly from any other front-end web framework. Change detection is the process where your framework can tell that some part of the data model has changed, and it now needs to update the user interface. There are several ways to do this. The most rudimentary way is to require the user to initiate this update, perhaps by calling a render method on some component. Obviously, this has the immense drawback of forcing the developer to remember to render at the right moment, while not rendering too often and hurting performance. The most popular method is to take your data and wrap it in some kind of framework object. Let's say, for example, you have the following scenario. You have a user object with a name property, then, somewhere in your application the user's name gets changed, and somewhere on your page you are displaying the name of this user and now wish to update that display. With the typical front-end application, you have to wrap that data in a framework object, perhaps like this. As you can see, we have to create an entirely new object, then later on when we wish to change the data, we have to use some kind of special set method like this. This is very workable, but it causes a huge problem. We have to use an entirely unnatural syntax when changing data. If we ever accidentally forget to use the syntax and just try to change the data the normal way, or we try to bring in a third-party library, or some other code we wrote, which changes the data using the usual assignment syntax, then you have a major problem and probably a nasty bug you will need to find and fix. Angular, on the other hand, uses an entirely different and novel approach. Angular realizes that comparing two pieces of data in JavaScript is an extremely fast operation, even when comparing thousands, or even tens of thousands of different pieces of data, so Angular works using what's called dirty checking. The way this works is that Angular, with a little bit of help from the developers, simply watches every event that could change data. For example, a button click might cause data to change, or it might not be a user-driven event, but instead a call to retrieve some data might have returned. When Angular detects that one of these events has happened, it kicks off what is called a digest cycle. This cycle looks to see if any of the data that is being rendered on the page has changed. It knows, because when it renders anything, Angular makes a copy of the data. Then, when the data changes it compares the old value to the new value. Remember, this is a very fast operation. And finally, if any of the data has changed, then Angular will re-render that data. There are many benefits to this system, but one of the most obvious is that you can change your data in any way you want. You can simply write to your data properties as you would with any other JavaScript object, and Angular will detect that change and re-render. Because of Angular's dirty checking method for change detection, Angular is one of the easiest frameworks to work with when it comes to dealing with data.

Dependency Injection

Angular uses a concept called dependency injection to make it easy to use the different components that Angular provides for you, and even your own custom business logic components that you may create. Let's look at an example. A typical function in JavaScript takes in arguments like the ones shown here. We provide the value of those arguments when we invoke that function ourselves. We might pass in a 3 and a 4, for example, but with Angular there are certain functions that we don't invoke ourselves, instead we hand these functions over to Angular, and it will invoke the functions for us. Let's look at some overly simplified examples so that we understand the concept. These examples themselves won't apply to Angular, but they will serve to show us in a simple way how Angular uses dependency injection. Let's consider a getData function. This function will gather some data from a remote server somewhere. In order to do that job, it needs some way to communicate with that server. In our example, our function has a parameter that we named $http. This is not an arbitrary name that we chose ourselves. We are actually asking Angular to give us an Angular component called the http component. This component has the functionality that allows us to easily communicate with remote servers. At Angular, components like this are called services. Because this is a built-in service, not one we created ourselves, it starts with a $. Another example might be a calculateCost function. This function needs two things to get its job done, the list of items it must calculate the cost for, and a shipping calculator that will perform any complex shipping calculations. In this case, these two services are ones that we created ourselves. They weren't built into Angular; they are custom to our application. So at some point, we register with Angular these services and from that point on, anytime we need one of them, we can simply ask for it by name using a parameter with the proper name, as shown here. This is the basics of how Angular uses dependency injection to make it easy to get ahold of components, or services, in Angular lingo, that can hold encapsulated functionality and data that we can use to build complex applications out of small and simple pieces.

Summary

This module covered the high level architecture of Angular. We discussed how Angular parses and compiles our HTML to find the instructions we give it to turn a plain HTML page into a fully-functional Angular application. We also discussed how Angular manages change detection without putting any constraints on the way you use and change your data. And finally, we looked at how Angular uses a dependency injection to make it simple to get a reference to a business logic component to let you compose your application out of small pieces.

Complementary Technologies

Introduction

Hello, I'm Joe Eames. In this module we'll be looking at some complimentary technologies to Angular. These are tools that work very well with Angular, and although not everyone will have a reason to use all of them, everyone involved in an Angular project should at least know about them, their place in the Angular ecosystem, and when and why to use them. We'll start off by talking about the most popular JavaScript library, jQuery, and how it fits with Angular. Then we'll talk about Ionic for making mobile applications. And finally, we'll discuss Karma and Protractor for testing your applications.

jQuery

JQuery is well known as the most popular JavaScript library in the world. It is used on more sites than any other library. It's also the most well-known library by web developers. That means that being able to leverage jQuery in your Angular applications can be very valuable, since instead of learning a new tool, developers are most likely already versed in it. Angular and jQuery are tied very closely together. Angular actually has a minimal version of jQuery included in its core library. This version includes the basic selection and manipulation functions from jQuery, but Angular also supports using the full version of jQuery, so if any page with Angular also has jQuery on it, then Angular will use the full version internally, instead of its minimal version. In Angular, jQuery is used when creating directives, which are those custom HTML elements I showed earlier, to let the developers find and manipulate the DOM. This isn't something that needs to be used in every directive, but when a directive does need to manipulate the DOM directly, jQuery is a tool used to do it. Angular will actually hand the developer a jQuery-wrapped DOM node to manipulate. Of course, you can use a different method to manipulate the DOM if you wish, but Angular makes it easy by incorporating jQuery seamlessly.

Ionic

Ionic is a very interesting framework. It allows you to turn Angular applications into native phone applications. This is a really big deal. Now a company that wants a mobile app doesn't have to hire expensive, specialized iOS or Android developers or outsource to an even more expensive mobile development shop. Instead, using the skillsets already on the team, you can build the mobile application you need. Ionic is actually built on top of a tool called PhoneGap, which allows a web application to access the native features on a mobile device. Coupled with the ability to compile a web app into a native app, PhoneGap and Ionic together vastly reduce the costs of mobile development. This technique is called a hybrid app, since it's a hybrid of both web and native. It is important to understand that the fidelity of the user experience can be lower with a native application, but for many products the savings and cost is more than worth it. Ionic is easy to learn and there are many tutorials on it. In fact, it's easy enough that a reasonably competent Angular developer can learn to make decent mobile applications in just an afternoon. The pairing of these two technologies is perfect for companies who need a mobile application, but don't want to spend a large amount of money on it.

Karma and Protractor

Karma and Protractor are two tools built by the Angular Team to help you with testing your Angular applications. We'll look at Karma first. Karma is built on Node. For anyone who isn't very familiar with Node, it's not just a tool used to build websites, but it is also used in many command-line developer tools such as Karma. Karma is used to test your JavaScript code, not your actual HTML, so its purpose is not to make sure a piece of the page looks right, or that a button works, but instead it's for testing with the business logic you write in JavaScript is correct. Karma has some awesome features, the first of which is that it will watch your files on the disk, and when they change it will rerun the tests. That means that every time you change your code, you know if you broke anything. Second, Karma will run your code in multiple browsers. That means that you can be sure that your code runs in every browser that you care about. Third, Karma is framework agnostic, so it doesn't care if what it's testing is Angular or React, or just plain vanilla JavaScript, it will still run the tests and give you the results. Finally, Karma is fast. Very fast. In most cases, it will run tests in milliseconds. Unless you have thousands of tests, Karma will run them without causing any noticeable delay. And since it's a command-line tool, it's easy to integrate it into your continuous integration process, or check-in process, or any other automated process in which you care to incorporate it. Protractor, unlike Karma, is not meant for just running your JavaScript, but it's actually meant to run your code directly in the web page where it belongs. This is called end-to-end testing. This means that we aren't trying just some small piece of the entire system, but instead we are writing our actual site and then testing that it truly behaves that way we expect. Protractor is actually built on top of another tool called WebDriver. WebDriver is a tool that was built to control existing web pages, meaning things like clicking buttons, entering text into text boxes, etc. WebDriver has a history of being frustrating and difficult to use. Protractor seeks to ameliorate this pain by adding a friendlier interface on top of WebDriver, and even adding some Angular-specific features. WebDriver, like Karma, can work with any front-end framework, but it does have some additional features if you are using Angular. Let's look at a quick example of a Protractor test. The above code shows a few interesting things. Although the code is somewhat readable, for example, it's obvious what browser.get does, or the sendKeys function, but we can see some of the Angular integration when we see this by.model. This lets us locate an element in the DOM, not by its location in the DOM, but instead by finding an element that is bound to a particular property on the model. It's not hard to realize that a feature like this is far less brittle than a simple CSS selector. Now, no matter how someone adjusts the HTML on the page, we will still find the correct element because we don't care about where it lies in the page, but only the data that it's bound to. Because of this and its excellent API, Protractor is a very popular tool for developers and testers alike, to test Angular applications from end-to-end.

Summary

In this module, we talked about several technologies that compliment Angular very well. We first talked about jQuery, and how developers can build on their existing knowledge of it to write their Angular applications. We also talked about Ionic, which lets us build mobile applications for a fraction of the cost of traditional mobile development. And finally, we talked about testing your apps with Karma and Protractor, and how these amazing tools let us gain confidence that our code works exactly the way we want, and as we maintain and add features to a product, we don't break existing functionality.

Angular Gotchas and Problems

Introduction

Hello, I'm Joe Eames. In this module we'll be looking at the common problem areas that you may encounter with Angular. Although Angular is a great product, there are still some places where you can run into trouble with it. This module will make you aware of those issues, and show you how best to deal with them. We'll break this module into two parts. The first part will be about problems that most projects will encounter. These problems are browser compatibility and external events and how they interact with the digest cycle. The second part of this module will deal with problems that only certain projects will encounter. Although you are less likely to see these problems, they are still common enough that you should know about them and know how to deal with them. Those problems are SEO, which stands for Search Engine Optimization, performance, and very large code bases, by which we mean projects with 100,000 lines of Angular or more. For each of these problems we will discuss what they are and how to deal with them when they happen.

Browser Compatibility

These next two sections are problems that most projects will encounter, which is browser compatibility and external events. Not every project needs to support older browsers, and simple projects probably won't encounter issues with external events, but these problems are still very common. We'll talk about browser compatibility first. When discussing browser compatibility with Angular, there's really only one browser we're talking about, Internet Explorer. The support for Internet Explorer in Angular is pretty straightforward. We'll look at each version of IE and which version of Angular you will need to be on if you want to support it. For IE 6 and 7, you will need to be using either Angular 1.0 or 1.1. Now that doesn't mean that as long as you are using one of these versions everything is hunky-dory, you will still have some extra hoops to jump through. You can't use custom elements in these older versions of Internet Explorer without doing a little bit of work. Fortunately, the documentation for Angular is pretty explicit on what you need to do to get it working with these versions. For IE 8, you will need to be using Angular 1.2 or earlier. Again, there are a few extra hoops to jump through to make it work, but you can guarantee that it will work. For IE version 9, you can use the current version of Angular, and for the most part, it just works. There is no extra effort on the developer's part for IE 9 or later, so browser compatibility can definitely be an issue if you are trying to support older versions of Internet Explorer. The good news is that the older versions of Angular are still great versions of the framework. They are stable and they're well documented. So if you are stuck with an older version, it's certainly not the end of the world.

External Events and Digests

In Angular, just about every event you can deal with in the browser is handled by Angular before your event handling code executes. This is important, because in order for Angular to keep the display up-to-date, it has to know whenever any data changes. Under the hood, whenever Angular is notified about an event you care about such as a button click, or a mouse move, or some data returning from the server, then Angular triggers a digest cycle. This is when Angular checks to see if any data has changed through its dirty checking mechanism, and if it has then it will update the display. Unfortunately, it's not possible to be aware of every event that could happen in the browser for a few reasons. First, there are a few browser events that Angular doesn't handle natively, such as events raised by the HTML5 video element. You can also create and raise custom events on HTML elements, which wouldn't be handled by Angular by default. Also, Angular doesn't natively have a way to deal with events coming from sockets, and if you incorporate some third-party code that raises events, then Angular won't know about that. And finally, if you write some non-Angular code yourself that invokes an asynchronous event, Angular won't know about that either. Thankfully, there are easy ways to deal with all these problems. For unhandled and custom events, you can write custom directives to listen to these events, and make Angular aware of them when they happen. For more information on this, see my course on directives. For sockets, third-party code, and non-Angular custom code, the solution is the same in all three cases. You simply have to listen to when the event happens using whatever is the native way, which is typically a callback, and then in that callback, you have to manually trigger a digest cycle, which is simple to do. In each of these cases, the first time you encounter the problem, if you're not looking for it, you can end up scratching your head for a while, but once a developer learns how to wrap an event in a directive, or manually trigger a digest, these issues become very straightforward to deal with.

SEO

These next three sections are problems that only a minority of all projects will encounter. This doesn't mean that you can just ignore these sections. These issues are still common enough that everyone who does Angular development, or anyone who is deciding on whether or not to use Angular for a project, should be aware of them. SEO, or Search Engine Optimization, is the process of affecting the visibility of your website in a search engine's results. This might normally seem like a huge deal, but for many sites and applications it's not very important at all. But for some sites, those who rely on customers being able to discover the site or products being sold on the site, or information available on the site, SEO can be the difference between profit and failure. Angular, like other front-end frameworks, utilizes what's called client-side rendering. That means that most of your HTML isn't sent directly with the page, but instead, it comes later on. That means that for most search engines, what they see is almost none of your site because without the JavaScript rendering the HTML, your site has almost nothing to display. When SEO is important to your Angular site, there are three different ways to deal with the issue. The first method is to do nothing. No, this doesn't mean ignore the problem, but it means to rely on Google to correctly crawl your site even with the client-side rendering. In the recent past, Google has made major strides to begin executing JavaScript while indexing sites. That means that your site may still appear correct to it. With Google taking the lead, the other search engines are sure to follow suit. The problem with this method is that it is far from foolproof. Google gives you some tools to see how your site looks so you can see if it's indexing correctly, but this method is basically only good for when SEO is somewhat important, but not extremely critical to you. Of course, it is by far the easiest method. The next option is to just take a few portions of your site and render them on the server. The effectiveness of this method relies on how much, and which portions of your site are important to have indexed, and which aren't. For example, for many sites just the home page and a few other informational pages are worth indexing, and the rest won't matter. In this case, just use server-side technology to render these pages. You can still have Angular running on these pages, but you can't use it to render the HTML in the client. The final option is to use a prerenderer. A prerenderer is a tool that will run through your whole site and execute the JavaScript, produce the resulting HTML, and then make static versions of those pages. Once that's done, it will cache these static pages, and whenever Google or Bing come calling, that is what they will see instead of your Angular site. This method gives the best results for SEO. It lets you optimize exactly what is seen by the crawler, and make sure that everything that you want to be discoverable by search engine is easily found. There are lots of different prerendering technologies, some specific to Angular, some not, and there are many variations of how this works, but in all cases, this method takes a reasonable investment of time to implement, and maintain, and optimize.

Performance

Performance is one of those areas where there is far more uninformed fear and ignorance than just about any other area. The biggest problem is the unnecessary and costly quest for maximum performance. With applications, there's no reason to pay for more performance than you need, especially since the difference in cost is often an order of magnitude or greater. For example, if a user action takes 30 ms to complete, then there is no reason to work until it takes 10 ms since your users will never know the difference. So never prematurely optimize, and never give more weight to performance than it truly deserves. With that in mind, let's discuss performance in Angular applications. Angular is quite fast, and thousands and thousands of applications have been developed without a single performance problem, but there are rare circumstances when the performance in Angular could become unacceptable, and at that point you will need to know what to do to address the issue. The first thing we need to understand is what causes performance problems in Angular. When you do have a performance issue, 99% of the time it will be because of too many bindings. Remember, a binding is when you point Angular at some piece of data on your model and tell it to display that data on the page, and whenever that data changes you want Angular to update the page. Now since bindings are the cause of performance problems, the first thing you can do to fix a performance issue is to simply use fewer bindings. This may seem like a flippant statement, but is actually a very viable approach. You can do things like look at pieces of the page that aren't being shown all the time, and remove bindings from those places until you need them. The details of how to do this are varied, and will often depend on exactly how your page works, and going into those details is beyond the scope of this course, but this is the simplest and first technique to have in your belt when addressing performance issues in Angular. The next technique for dealing with a performance problem is to change the user experience. This is really just an extension of the previous piece of advice. Again, the goal here is to reduce the number of bindings, but in this case you do it by redesigning your page. For example, instead of an infinite scroll list, you can change to a page data display, or you could not show details on the main page and require the user to go to a details screen to see them. There are more possibilities here than there is time to cover them, but again, this is a viable technique for addressing performance issues in Angular. The third technique is to use one time bindings. This feature was added in Angular 1.3, and allows you to take a piece of data from the model, render it, but turn off the ability for that rendering to be updated if the data changes. This works well with data that is relatively static. Say, for example, you are displaying a catalog of items to purchase. Well, it's pretty unlikely that the item description will change in the middle of a user session. In fact, it probably only changes very rarely. So you can simply display that data and then turn off the ability to re-render it should the data change, since you won't ever be changing this data on the client. This is one of the easiest ways to deal with a performance issue, and can be very simple to implement. For more information, you can see the details of how to do this in my course on What's New in Angular 1.3? The final technique, and by far the most drastic one, is to use a different rendering engine. Angular's rendering engine is reasonably performant, but there are certainly other rendering engines that are faster. Currently, React's rendering engine is getting a lot of press because of how fast it is. It is completely possible to use React with Angular. You can also use other rendering engines like Handlebars, but in every case the developers doing the implementation will need to understand Angular quite a bit, and the other rendering engine just as well, since this is by far not a trivial thing to do. The most important thing to remember here is that this isn't something you will want to do wholesale. It would likely be a mistake to build an application with Angular that never uses Angular for rendering, but instead uses a different technology for rendering in every view. You will be missing out on so much of the value of Angular that your development costs will likely be significantly higher than they need to be. So use this technique only sparingly. Dave Smith gave a great talk on doing exactly this at ng-conf 2015. You can find his video titled Angular + React = Speed, at this URL here. These are the different ways to deal with performance issues in Angular. Again, although these are things that you only have a small chance of needing since Angular's performance is already perfectly acceptable for the vast majority of use cases, but when you do need to do something, there are certainly plenty of options available to you to make Angular as performant as you need it to be.

Very Large Code Bases

When it comes to large code bases in JavaScript, there's probably no better tool for managing that than Angular. Its guidance on helping you segment your code into small, maintainable chunks that can be used to compose a larger application is probably the best that is currently available in any front-end framework. But maintenance is only part of the difficulties when dealing with large code bases. Now let's be clear, we're not talking about 1000, or even 10,000 lines of code. We're talking about big, big, big JavaScript code bases. Now of course writing and maintaining that code is difficult and Angular does a lot to help out with that, but as you hit this amount of code, you'll begin running into a specific issue that Angular doesn't currently address, and that is the size of the code itself. Remember that in a JavaScript application, all the code has to be sent from the server down to the client in order to run. This means bytes sent across the wire and that takes time. The more bytes, the more time. Of course, you can minimize and concatinate your code and use GZip on the server to make that byte count as small as possible, but even with these techniques you may still run into the fact that the code just takes a long time to get to the browser. And then once it's on your browser, it has to interpret all that code and that takes time too. So once you reach this problem, the common way to address it is to begin lazy loading your code. This means that you only transmit the code that you need to start using the application, and then as the user goes to different parts of the application, the code needed to run that is either downloaded at that point, or is downloaded beforehand, but not until after the application initially loads. Unfortunately with Angular up to version 1.4, lazy loading is not natively supported. It looks as though Angular 1.5 will support lazy loading, but at least for versions up through 1.4, this is not something that Angular will do for you. So if you do get to where your code is taking too long to send across the wire in one package, there are some options. If the built-in Angular method for lazy loading has been released, you can use that, but if you are stuck on an older version, you can use a third-party product to allow you to lazy load your Angular code. Currently there are two relatively popular options. The first is called ocLazyLoad, which was built by Olivier Combe. This option is best supported and most popular currently. The other option is called Overmind, and was built by Dave Geddes. This option is less popular and less well-supported, but is worth looking into when choosing a lazy loading library. In any case, dealing with very large code bases in Angular is very accomplishable and in general, much easier than alternative frameworks.

Summary

In this module, we talked about several gotchas and problems with using Angular. First, we talked about browser compatibility, which basically comes down to knowing which versions of Internet Explorer you need to support, and therefore, which version of Angular you have to use. Then, we discussed external events and how we notify Angular of those events so that it will keep our UI up-to-date. After that, we discussed some less-common, but no less important issues, the first of which was SEO and the ways to deal with that. Then we talked about performance issues in Angular and how to address them, and finally we talked about very large code bases and how lazy loading becomes a necessity when the code base reaches a certain size, and how to deal with that in Angular. In all these cases, even though there are some unique challenges that come with Angular, none of them are very difficult to handle, and in many cases are quite simple to address.

Angular vs. X

Introduction

Hello, I'm Joe Eames. In this module we'll be looking at the major differences between Angular and several alternative front-end technologies in frameworks. This module will highlight when you might want to use Angular over a specific technology, and when you might not. Although every attempt has been made to be as unbiased as possible, and although I have used every technology and framework compared, it will still be to your benefit to investigate other alternatives to Angular when deciding on what direction to take a project. We'll begin by discussing when and why to use Angular over server-side rendering, and when not to. Then we'll move onto a comparison of using just jQuery and other vanilla JavaScript for a complex front-end versus Angular. After that, we'll discuss using Angular versus older MVC frameworks, of which we'll primarily focus on Backbone. Then we'll discuss the difference between Angular and modern MVC frameworks like Ember. And finally, we'll discuss the difference between Angular and React, and when to use each.

Server-side Rendering

Server-side rendering is any technology where the HTML is completely rendered on the server and sent down to the client where it will display untouched. Each new view requires a roundtrip to the server. This includes technology such as .NET, JavaServer Pages, PHP, Rails, and many others, although some of these technologies blur the line a little bit by sending server-side rendered HTML fragments down to the client to be plugged into specific places. Until a few years ago, this was pretty much the only way that web applications were built. Today, the vast majority of existing web applications are still using server-side rendering, but client-side rendering options are becoming more and more popular. Let's look at the advantages and disadvantages of using server-side rendering over Angular. First, server-side rendering has a big advantage when it comes to SEO. As we saw in the last module, SEO is much easier to optimize using server-side rendering. There are still options to get around this using a client-side render technology, and prerenderers will fix most of that, but it still involves more work. The next advantage is familiarity. More developers are full stack developers and know a server-side render technology than a client-side rendered one, so it can be easier to find qualified developers, and because of the supply, you may be paying less for these skills. Finally, the languages used in server-side rendered technologies are often more mature languages with fewer blemishes than JavaScript. Now, let's talk about the disadvantages of server-side rendering versus using Angular. By far, the most compelling reason and the one that drives most projects to go to a client-side rendered technology, is usability. Because every action involves a roundtrip to the server, it's just not as nice to the users of your site, and in this day and age, usability requirements are high and only getting higher. Server-side rendering simply can't meet this demand. A smaller concern would be desirability. As technologists, developers, especially the most sought after ones, want to be working with more interesting and recent technology. This is hardly a universal rule, but it is a concern to keep in mind. As such, it may be harder to find the kind of developers you want if you are using purely server-side technology. Because of these considerations in just about all cases, Angular wins out over server-side rendered tech simply because you can build better products with it for roughly the same cost.

jQuery

The next tech we will compare to Angular will be jQuery and vanilla JavaScript. When I say vanilla JavaScript, I mean not using any significant amount of third-party code, but instead just using plain JavaScript that you write yourself. This is a pretty short comparison list. The advantages of a tech stack like this lie mostly in a single point, the ability to fit edge cases better. If you have some very strange requirements, if your app is way different from any kind of typical web application, then something homegrown might fit your needs better. The disadvantages of this over Angular, unfortunately, are significant. Mainly this comes down to cost. The cost to build and maintain custom code is significantly higher, probably an order of magnitude higher. Another significant factor is the sanity of your developers. Having to work in a homegrown framework can quickly become so taxing on developers' mental states, that you may find yourself with an unusually high turnover rate. Finally, the desirability of this is low since the experience they gain is unlikely to help them in a future job and few people want to learn someone else's homegrown framework.

Older MVC Frameworks

Next, we will compare Angular to older MVC frameworks. I will focus on Backbone, but other frameworks like CanJS, Batman, and Knockout are just as applicable. When it comes to advantages, these generation one frameworks are very short on things that they do better than Angular. Since most people don't consider these older frameworks for a new product, primarily the advantage here will be that your product is already written in one of these frameworks, and so it may not be important enough to switch to a newer technology, but there are some very compelling reasons to do so. Building a product in Angular is significantly cheaper than these older frameworks. If nothing else, just because you write so much less code. Second, Angular will have significantly more features than an older framework. Most of these frameworks only have a small subset of the features of Angular. Third, it will be far easier to hire for an Angular position than something working on an out-of-date framework. And finally, these older frameworks aren't kept updated nearly as much, as there is far less interest in them. Even older versions of Angular will get bug fixes from the community as they are identified, because of the wild popularity of Angular.

Ember and Other Modern MVC Frameworks

In this section, we will compare Angular to Ember, and other modern MVC frameworks. There are several others such as the up-and-coming Aurelia. These frameworks are very similar to Angular in terms of feature set, so as you compare them you will quickly get into subjective preferences. We will attempt to stay clear of such things, and simply talk about some of the more significant differences. One of the advantages of Ember in particular, is the fact that it is highly opinionated. This is not considered by all to be an advantage, but for many developers the stricter guidance of how to build an application is a benefit, so we will list it. Although this is the only advantage I will list, don't take that to mean that Ember and other modern MVC frameworks don't have lots of advantages. Ember is a very full-featured framework, and like Angular it will save you lots of lines of code over older frameworks in vanilla JavaScript, but as far as clear advantages over Angular, we will keep our list to just this one item. On the disadvantages side, first off, Ember is highly opinionated. For a reasonable amount of people, the very strict guidance can become stifling, so we will list it as both an advantage and a disadvantage. Angular has a reasonable amount of guidance, but it is far more versatile when it comes to allowing you to structure your code. Next, these other modern frameworks suffer greatly from reduced popularity. This affects many things. First, the amount of resources being invested in building them; second, the number of resources there are to learn them, and also, how many developers already know them, and also how desirable they are for developers who don't already know them. Angular is currently the most popular framework, so it is a clear leader on this point. Finally, even though Angular is not necessarily the most performant framework out there, its performance is still very good. Some of these other modern frameworks like Ember actually suffer from far worse performance than Angular, so that's something to keep in mind.

React

In our last section, we will contrast Angular and the React framework. React is an exciting new technology built by a team at Facebook. First, its performance is unrivaled. React is fast, almost an order of magnitude faster than Angular. Again, how much this really matters depends on your application. Second, React has great browser support. The current version of React supports some very old versions of Internet Explorer, so if you have to support old versions of IE, React is a clear winner. As far as the disadvantages go, React is still far less popular than Angular. Although its popularity is growing, it's unlikely to become near as popular as Angular is. So many large companies have made Angular their official front-end framework, that it's going to be a while before any other framework gets close to its popularity. Second, React suffers from what is known as Frankenstein Framework Syndrome. React by itself is really only a rendering engine; it doesn't do other things like routing or server communication, or any of the other things that Angular does, so you will have to look around and piece together the different libraries and parts that you will need to make a complete solution. Again, the amount of effort this takes will depend on the nature of your application, but this is not a trivial task, so most teams will want a very senior and experienced architect to help make the decisions on which pieces will work best for your application. With Angular, all of the necessary pieces to build a full app are already part of it, so there's no extra work to evaluate and plug in the different parts. It is a complete solution.

Summary

In this module, we contrasted Angular with several other options for building web applications. We talked about using a server-side rendering; we also talked about using jQuery or a homegrown framework. We discussed the disadvantages of using older MVC frameworks, and then we talked about the pros and cons of using a different modern MVC framework. Finally, we compared Angular to React. In all these cases, the items that I listed are the most objective and significant differences, but in reality you could spend days comparing two different options and still not exhaust the subject. Choosing a framework is something that should be done with care and consideration. Angular stands as an obvious choice for most web applications, but for certain products, a different solution might be the right one.

The Future of Angular

Introduction

Hello, I'm Joe Eames. Angular is an amazing framework, but no discussion of it will be complete without looking at its future and where it's headed. In this module, we'll look at the future of Angular both in the near term and the long term. We'll look at that future by discussing both where Angular 1 is heading, and where Angular 2 is currently at, and where it's going as well. We'll also consider the role of Angular 1, as Angular 2 is released and gains its own traction.

The Future of Angular 1

We're going to first look forward to the future of Angular 1 by talking about what is in store, both before Angular 2 releases, and after it releases as well. Angular 1's near future is defined by essentially moving forward in the same way it has been moving forward until this point. Even though Angular 2 is on the horizon, Angular 1 is not being forgotten. Just recently the Angular Team announced that Pete Bacon Darwin will be the new lead developer for the Angular 1 framework. Adding such a well-known person as dev lead will certainly address any concerns that there is not enough talent dedicated to Angular 1 currently. As time goes on, we will continue to see regular releases of Angular 1, just as we have in the past. That means new features and bug fixes just like we have been used to up until now. We will also probably be seeing some new features that come from a very unusual place, Angular 2. Recently the Angular Team released a new router for Angular 1, which is a backport of the Angular 2 router. This is a great example of the ways that Angular 1 will benefit from Angular 2. We might possibly see other backported features in the future. After Angular 2 releases, then we can expect a natural shift in Angular 1. Of course, the releases won't just dry right up. We will continue to see releases as new features are added to Angular 1. The Angular Team doesn't plan to simply cut off Angular 1 as soon as Angular 2 is released. In fact, Google itself is one of the most prolific users of Angular 1, so there will be plenty of reason to keep adding new features to it. It is not unreasonable to expect that this phase will last for many years. After a period of time, we can expect the new features will become less and less frequent in Angular 1, and the Angular 1 Team will move to a more maintenance mode of just fixing identified bugs. Again, this phase will likely last for many years based on the popularity of Angular 1. Finally, at some point, Google will drop official support for Angular 1. Nobody knows when this will be, and truly it will be determined by the developers who use Angular 1. As the number of current Angular 1 projects reaches a specific threshold where there just aren't enough users to justify continued investment, Google will drop support. Based on what I've personally seen, I don't expect this to be for 5-10 years. Google is certainly not a company that is afraid to put money towards things that don't directly make money, but do make customers happy. At this point, it will be the community who are supporting Angular since it's an open source project. So even those few companies who continue to support projects written in Angular 1 will still have the opportunity to receive bug fixes and maintenance as time goes by. Perhaps the biggest question on everyone's mind will be how much investment will there be by Google and Angular 1, as Angular 2 gets released. This chart shows something very interesting about that. As Angular 1 exploded on the scene, the amount of resources that Google was investing in it was relatively small. The Angular Team was very small in the past. Then, as Angular became more popular, Google added to the resources devoted to Angular, but many of those resources were dedicated to building Angular 2, so it wasn't until Angular 2 started that we really saw a big increase in what was devoted to Angular 1. This is the current state of affairs. Once Angular 2 releases, we can expect more resources within Google to get shifted to Angular 2. And long term, many years down the road, the team will definitely shrink, but notice that it will be a while until the team is as small as it was back when Angular first became popular. This is why we can rest assured that Angular 1 will be well supported for a long time to come.

The Future of Angular 2

Angular 2 is an exciting new future for the Angular framework. With Angular 2, the Angular Team has done something that others rarely do. They have completely revamped the framework to move it forward in a way that really wouldn't be possible without major changes. Because of this, Angular 2 doesn't have backwards compatibility with Angular 1. In many respects, it's like a completely new framework. Let's take a brief look at the design goals in Angular 2, so we can understand why these radical changes were required. First off, Angular 2 was completely designed with performance in mind. Angular 2 is significantly faster than Angular 1; in fact, it's between 5 and 10 times faster. This puts it on par with the fastest frameworks available today, such as React. This change cannot be understated. For an interesting look at this performance difference, Dave Smith's talk at ng-conf 2015 shows the difference in performance between Angular 1, React, and Angular 2. Next, Angular 2 has a simplified conceptual model. Angular 2 has removed the idea of controllers and modules, simplified how services works, and removed much of the configuration overhead. This means that learning Angular 2 will be quite a bit more straightforward than Angular 1. Angular 2 is also designed to be more mobile friendly so that both mobile apps and hybrid apps can take advantage of the unique nature of mobile devices through lower memory and battery needs. Finally, Angular 2 was designed with web standards in mind. That means that it was built to work with ECMAScript 6, the newest version of JavaScript, and even versions beyond that, plus, emerging web standards like Web Components and the Shadow DOM. This puts Angular 2 on the bleeding edge of current front-end application development and assures that it will stay relevant for a long time to come. The road ahead for Angular 2 is pretty straightforward. You can expect a release between the last part of 2015 and the first half of 2016. But if necessary, the team will delay that release to make sure that the framework is solid and truly ready. After release, you'll be able to migrate your current Angular 1 applications to Angular 2 using a side-by-side approach, where the 2 frameworks can both exist in the same application, and over time you can rewrite a piece at a time to move your application to Angular 2, if that's something that makes financial sense for your organization. Of course, many applications will make more sense to just remain as Angular 1 apps. Since we have seen already, Angular 1 will be supported for a long time and it is still a fantastic framework. Overall, Angular 2 is an exciting new framework, which will have a significant, positive impact on the web development world.

Conclusion

In this module, we looked at the future for Angular, both Angular 1 and 2 have bright futures ahead of them, and you can rest assured that they will be around for a long time. Thank you for watching this course. Angular is an amazing framework, which has benefited the lives of millions and millions of people. It is easy to learn, easy to build with, and easy to maintain. For your web application needs, you really can't go wrong choosing Angular.