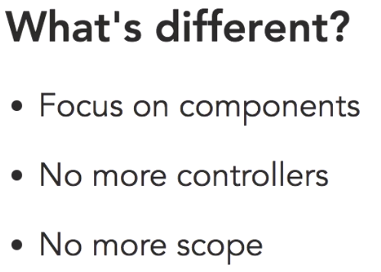


Angular JS is a modern framework for app development. That means that it's a structure that helps you create applications faster by providing several services and objects. Now those objects are going to make things easier for app developers. This is a dramatic upgrade to Angular One. It's really a rethinking of how applications should be put together.

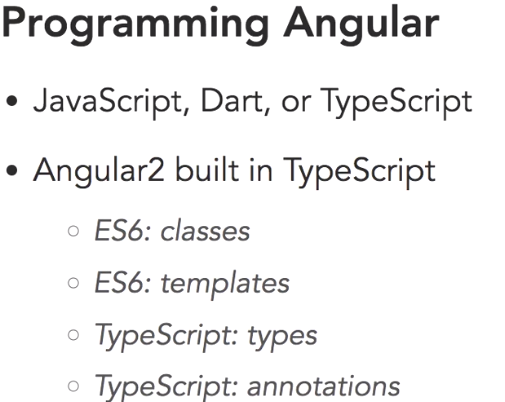
Angular JS is a modern framework for app development. That means that it's a structure that helps you create applications faster by providing several services and objects. Now those objects are going to make things easier for app developers. This is a dramatic upgrade to Angular One. It's really a rethinking of how applications should be put together.



First, in the new version of Angular, the structure revolves around components. Components are like Angular One's custom directives.

They'll let you create functionality around custom tags. So instead of adding an ng-app directive to a section of code, you simply invent your own tags and then program the functionality for these. It's a lot like using web components and it's one of the best features of the framework. Components are much simpler than creating custom directives. Since most of the work is handled by components, there's really no need for controllers in this new version of Angular

Another thing to get adjusted to is the fact that there's no more scope. Now this is a pretty big adjustment for somebody coming from Angular One. Scope is how we handled communication between our templates and our controllers.



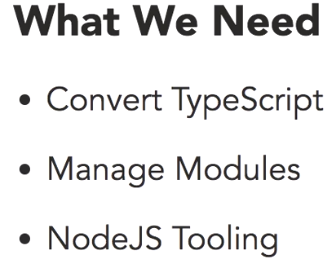
TypeScript is a language from Microsoft that is a super-set of JavaScript which means that it lets you use any of ES6's features and adds a couple of important features of its own. From version two on, Angular was built with TypeScript. So, there is a significant advantage to learning how to work with it. Mainly most of the documentation you'll find on the web and on the Angular site is specifically for TypeScript.

So, with TypeScript you get some of ES6's features and one of them is classes. ES6, which is also known as ECMAScript 2015, adds a few interesting features to this new version of Angular. And the first is classes.

ES6 also has a rich templating system that is extensively used in this new version of Angular, and it makes creating templates easier and much more powerful.

Now because TypeScript and ES6 are not supported in current browsers, you'll need to do something to convert this code to regular JavaScript

That means that you'll need to use a Build tool like Webpack or Gulp to handle the conversion. In this course I'll be using NPM Scripts.



**SystemJS**

Now because we're using the TypeScript version of Angular, we're going to need to have a process in place to convert the files to JavaScript so that our browser can understand it. In addition to that, because this version of Angular is modular in nature, and JavaScript doesn't understand natively how to manage modules, we need to have something that can do that for us. A popular library to take care of this is called SystemJS. It is used to process files.

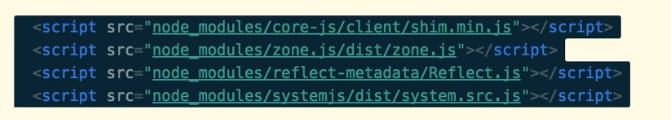
**Packages.json:**

Project has a lot of dependencies, mainly all the AngularJS dependencies.

TypeScript, as well as running a lite-server at the same time, and concurrently just makes that cross-platform.

Npm install command will download all the dependencies specified in the packages.json file.

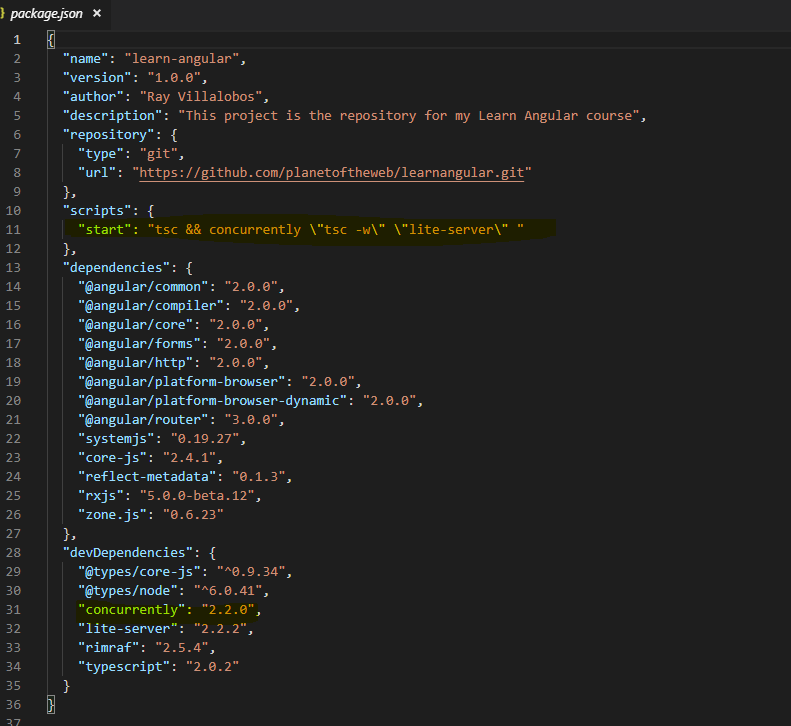
**Index.html:**

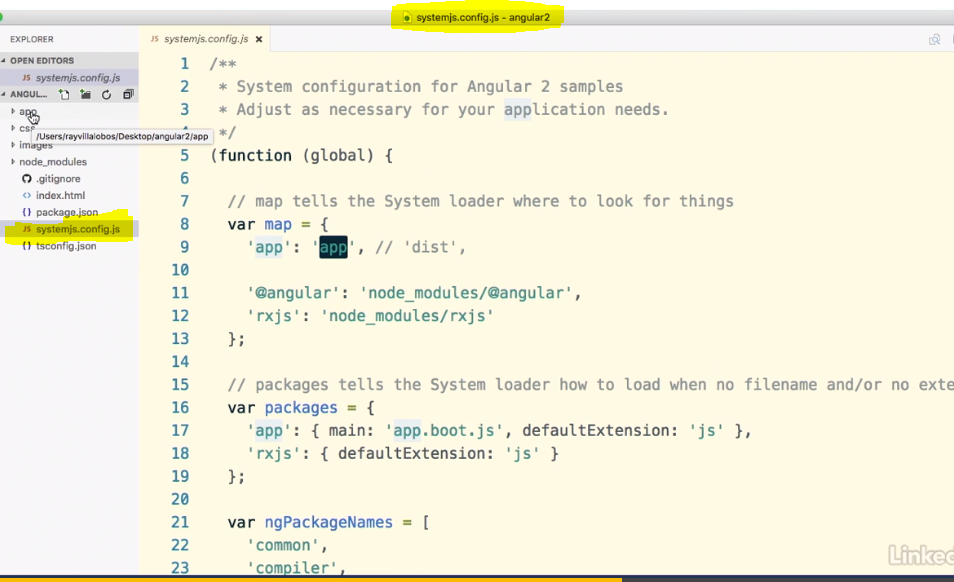


There are some scripts that Angular needs to be as backwards compatible as possible, and these are all them right here. Notice that this is using also the node\_modules folder. So these are all installations that are in the node\_modules folder.



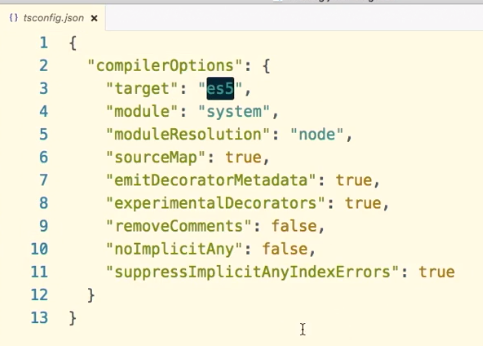
This file configures our SystemJS module loader.





**Systemjs.config.js** file sets up how the modules are going to be processed. So, we can see that the important parts of this is this **distribution** folder, which we've set to **app**, because our folder where we're going to place all our TypeScript is called app.

**Tsconfig.json:**

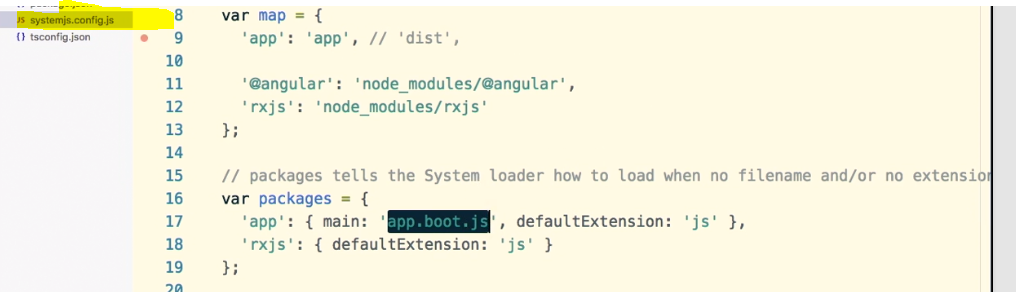


This determines how our TypeScript is going to be converted into regular JavaScript for us. So, we can see that we are targeting **es5**, which is the most common version that most browsers support.

We're using system as our module handler, as well as some of these other options.

To set up a typical application structure we're going to need **three** different files.

We need to use a file **to bootstrap our application (systemjs.config.js)**, another one **to describe all our modules(packages.json),** and then, finally, a module to start up what we want to do in the application**(app.boot.js)**. So first, if you notice, in our system configuration.js file, first, in the system.jsconfig file, we are calling this app.boot.js file, so this actually what our system loader is going to use to manage our modules.





NgModule is what we called a decorator, decorators are functions that modify JavaScript classes, essentially, it's just a configuration object, so think of it like a settings document for your specific component.

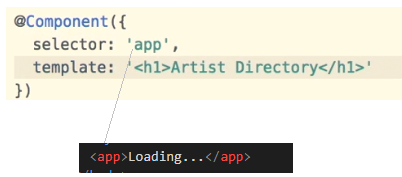
Inside NgModule we have 3 components:

1. **Import**: this is where we add modules that we need for our application to run
2. **Declarations**: these are the view classes that belong to this module.

Difference between the import command and the declarations is that the import imports modules that we're going to need, think of them as features for your application, while declarations will allow us to import all the modules that we have within our application

1. **Bootstrap**: Finally, we need to decide the bootstrap component for the application. This is going to be the component that has the other components, sometimes called the root component.

Now, other components are going to need this file to understand the relationships between the different files and components, so we need to go ahead and export this as an AppModule.



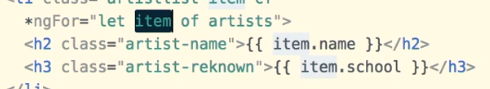
**Interpolation:**

Templates become more powerful when they can do things that are connected to our component. Now the simplest way to tie a template to a component is through interpolation. You can use double curly braces to bind the property name from your component to your template.



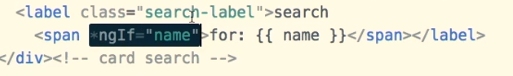


**To iterate through the list of arrays:**





**ngIf:**

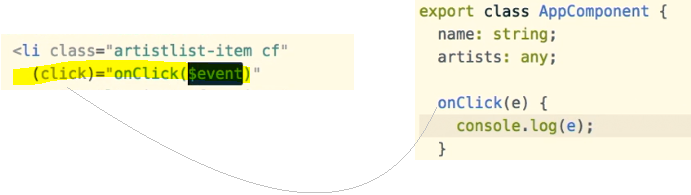


**If the name exist then only print “for name”**

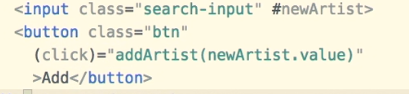
**Events:**

Parenthesis is used for events.

**Ex: Onclick event**

****

Template Variable:



**newArtist** is called as the template variable. In this case it is the input text variable

**Binding:**

Angular provides a way to bind properties to your template using **square brackets** So binding is just another way of connecting things with it you can control say a dom property through a variable or a method in your component

Bind-innerHTML and [innerHTML] will work similarly.

The square brackets notation gives us an easier way to access and modify properties that are in a template using something within the component.

**Two-Way Data Binding:**

Sometimes, it's convenient to have an element that can both respond to an event and modify a property at the same time. Angular gives you a shortcut to do this that combines both techniques that we've doing so far. The technique is called two-way data binding because it's tracking an event and setting a value at the same time

Angular has a couple of directives called **ngModel** and **ngControl** that allow you to handle **two-way binding.**

**ngModel**: when we want to work with an input field.

If we have a control, say something like a checkbox or a radio button or a pop-up, we will use ngControl.

We need to import **FormsModule** from Angular forms library to use 2-way data binding.

