|  |
| --- |
| PS> npm install -g @angular/cli |

all\_outcontent\_copy

Now you are ready to use the [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/) to create a new project. The command that you use to access the [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/) is ng. To create a new app type the following:

|  |
| --- |
| PS> ng new angular-get-started --style scss |

all\_outcontent\_copy

This will create a new Angular 2 project from scratch and set it up so that you can use [SASS](http://sass-lang.com/) as CSS preprocessor. If you go inside the angular-get-started folder that should have just been created you’ll find a folder structure that will look more or less like this:

|  |
| --- |
| // app source code (and specs)  - src  - app # your app source code (and specs)  - assets # static assets like images, etc  - index.html # the entry point to your app  - styles.scss # the global styles for your app  - environments # here you can define different environment configuration (prod, dev, etc)  // dependencies  - node\_modules # the source code of your app's dependencies  - package.json # the manifest of your app that states all dependencies  // TypeScript configuration  - tsconfig.json # TypeScript compiler configuration  - tslint.json # TypeScript linting configuration  // Testing  - e2e # a folder with end to end tests  - karma.conf.js # karma test runner configuration  - protractor.conf.js # protractor e2e tests configuration  - .gitignore  - README.md |

all\_outcontent\_copy

You can test that everything works as it should by running the development server. Type the following command:

|  |
| --- |
| PS> ng serve --open  # the short-hand syntax is: ng s -o |

all\_outcontent\_copy

This should start the development server, open a browser and load your app. You can stop the server any time by typing CTRL-C a couple of times in the terminal. But for the time being just rejoice! **You have created your first Angular 2 application!**

## Index.html The Entry Point for Your App

The index.html is the entry point to your application:

* It links to your styles and javascript files
* It provides the HTML markup for your application with the custom <app-root> element.

If you take a sneak peak at the index.html file in your project you’ll see the following:

|  |
| --- |
| <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>AngularGetStarted</title>  <base href="/">  <meta name="viewport" content="width=device-width, initial-scale=1">  <link rel="icon" type="image/x-icon" href="favicon.ico">  </head>  <body>  <!-- 3. Display the application -->  <!-- this is your application root component -->  <app-root>Loading...</app-root>  </body>  </html> |

all\_outcontent\_copy

You may be wondering… Hmm… where are the styles and javascript files? All I can see is a custom element *app-root*. You’ll be happy to know that you don’t have any sight problems as the styles and JavaScript files are indeed not there… yet. The reason why that is the case is that the [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/) relies on [Webpack](https://webpack.github.io/) to inject these files when they are needed.

### Webpack? What is that?

Web development is not what it used to be. In the olden days you would create a script tag, write your javascript and that would be it. With the need of highly interactive and rich web applications and the use of modern web technologies like TypeScript, SASS or ESnext and new features like ES6 modules we need something that can translate our applications into something that can run in any browser.

That’s where tools like [Webpack](https://webpack.github.io/) step in. Webpack is a module bundler that takes all these files and modules, processes them and makes them available in a way that can be run in any browser. Moreover it can optimize your application by taking advantage of a rich community of plugins.

Take a look at the awesome [Webpack docs](https://webpack.js.org/concepts/) if you’re interested in learning more.

## Bootstrapping Your App

In AngularJS we used the ng-app directive to point Angular 2 to the starting point of your application. In Angular 2 we use a bootstrapper. You can find the bootstrapping logic in the src/main.ts module:

### What is a Module?

A TypeScript/ES6 module is a way to encapsulate, group, reuse, distribute and load code in JavaScript.

Generally, a module will contain code that encapsulates a specific functionality. The module will expose this functionality to the rest of the application by defining a series of exports that other parts of the system can then import.

The core module for Angular 2 for instance is called 'angular2/core' and it gives you access to Angular core primitives like Component.

|  |
| --- |
| import { enableProdMode } from '@angular/core';  import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';  import { AppModule } from './app/app.module';  import { environment } from './environments/environment';  if (environment.production) {  enableProdMode();  }  platformBrowserDynamic().bootstrapModule(AppModule); |

all\_outcontent\_copy

In this file we import the platformBrowserDynamic object from the '@angular/platform-browser-dynamic' module and call its bootstrapModule function with the AppModule as argument.

There’s two interesting things to note here:

* The '@angular/platform-browser-dynamic' bootstrapper hints at the fact that **Angular 2 is platform agnostic**. That is, you can run it in the browser, but you can also run it on a web worker or in the server using different bootstrappers.
* Notice how we import modules in a different way, Angular 2 modules are imported by name, and application components are imported using relative paths.

Calling the bootstrapModule function with the AppModule as argument tells Angular that this module is the main module for your application.

## Wait… But What is an Angular 2 Module? I Thought We Were Just Using ES6 Modules in Angular 2!

Yes, we do use standard ES6 modules in Angular 2. Angular modules are just a new convenience for us developers, a higher level module to wrap ES6 modules that are all about developer ergonomics.

Angular 2 modules and the new NgModule decorator let us declare in one place all the dependencies and components of our application without the need to do it on a per-component basis (like we used to do in previous versions):

|  |
| --- |
| import { BrowserModule } from '@angular/platform-browser';  import { NgModule } from '@angular/core';  import { FormsModule } from '@angular/forms';  import { HttpModule } from '@angular/http';  import { AppComponent } from './app.component';  @NgModule({  declarations: [ AppComponent ],  imports: [ BrowserModule, FormsModule, HttpModule ],  providers: [],  bootstrap: [AppComponent]  })  export class AppModule { } |

all\_outcontent\_copy

The NgModule decorator takes an object with the following information:

* an imports array where you declare your module dependencies, for instance, browser, forms, routing or http. The BrowserModule used above contains all the dependencies necessary to run Angular 2 on a browser.
* a declarations array where you declare the components and directives that belong to the current module.
* a bootstrap array that identifies the root component that Angular 2 should use to bootstrap your application.

In this example we import an AppComponent component from the app.component.ts module and set it as the root of our application.

## But What is a Component?

The component is the core building block of Angular 2 applications. It represents a reusable piece of UI that is usually depicted by a custom html element.

A component is self contained and is constituted by at least a piece of html code that is known as **template**, **a class** that encapsulates the data and interactions available to that template, and the aforementioned html element also known **selector**.

The AppComponent looks like this:

|  |
| --- |
| import { Component } from '@angular/core';  @Component({  selector: 'app-root',  templateUrl: './app.component.html',  styleUrls: ['./app.component.scss']  })  export class AppComponent {  title = 'app works!';  } |

all\_outcontent\_copy

It has an AppComponent class that is decorated by some metadata in the form a [TypeScript](https://www.barbarianmeetscoding.com/blog/2017/08/17/typescript-javascript-and-types-equal-awesome-developer-productivity/) decorator @Component which binds the class to its template, its styles and the app-root selector (Can you remember the *<app-root>* element from the *index.html* file?).

This metadata tells Angular 2 that whenever you see an <app-root> html element you should render the template in templateUrl, with the styles in stylesUrls and in the context of the AppComponent class.

### A Decorator?

If you are familiar with C# or Java, a decorator works exactly like an Attribute.

If you are not, then think of a decorator like a function that is applied to whatever it decorates. In this case, the Component decorator attaches some metadata to the AppComponent class.

[Want to learn more about decorators? Check this example](http://jsbin.com/wixuta/edit?js,console).

If we take a sneak peek at the template you’ll see the following:

|  |
| --- |
| <h1>  {{title}}  </h1> |

all\_outcontent\_copy

The title bit (which we call interpolation syntax) tells Angular 2 to display the content of the component title variable when rendering the component. Indeed, if you run ng serve you can see how the browser renders the text app works! inside an h1 element.

If you take a look back at the AppComponent definition and at your project structure you’ll realize that the [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/) follows a convention when creating components:

* All files for a given component are placed within the same folder named like the component itself (in this case app)
* These files are named with app.component and have different extensions based on their function:
  + app.component.ts for the component itself
  + app.component.html for the template
  + app.component.scss for the styles

Now let’s do a quick test to see how we can bind the AppComponent class members to the template. Let’s change the title property value from app works! to Star Wars PPlz!!!!!. If you now save the file, your browser should automagically refresh itself and you should be able to see the Star Wars PPlz!!! as the title of your app (If you closed your browser just open it again and if you stopped the development server then remember to run ng serve --open to start it again and see your changes).

### Classes in JavaScript??

ES6 brought classes to JavaScript amongst a host of other features. If you are a software developer with a strong typed language as background this will probably be great news for you.

Don’t be fooled though, JavaScript classes are just syntactic sugar over the existing prototypical inheritance model. If you want to know more about [ES6 classes read this article](http://www.barbarianmeetscoding.com/blog/2015/12/20/white-tower-summoning-enhanced-the-marvels-of-es6-classes/). [And TypeScript adds even more great features to classes](http://www.typescriptlang.org/docs/handbook/classes.html).

## Your First Component, Listing Star Wars People Of Note!

[Lisa](https://twitter.com/Ryrholm) and I were discussing for a while which super duper cool service we would use for our workshop… Spotify, SoundCloud, instagram… And all of the sudden we came to this free and open [Star Wars](https://swapi.co/) web service. So that’s what we’re going to be building, a gateway into the Star Wars universe. Starting with its peoples of note!

The first thing that we are going to do is not going to require services, not yet. We are going to create our first component to display a list of Star Wars people and we will start faking out that data.

It’s good practice to start by defining the domain model of our problem space, in this case a Person. We’ll take advantage of [TypeScript](https://www.barbarianmeetscoding.com/blog/2017/08/17/typescript-javascript-and-types-equal-awesome-developer-productivity/) interface (although we could’ve used a class as well) and create a Person within the person.ts file. But we won’t create it by hand, we will use the [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/)! [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/) to the rescue!

The [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/) has something that we call generators that let you scaffold parts of your application for you. For instance, you can use the following command to create an interface:

|  |
| --- |
| PS> ng generate interface person  # or in short-hand:  # ng g i person |

all\_outcontent\_copy

The result of running this command is a person.ts file in your app folder:

|  |
| --- |
| export interface Person{  } |

all\_outcontent\_copy

Now let’s add some properties that are representative of a person:

|  |
| --- |
| export interface Person{  name: string;  weight: number;  height: number;  } |

all\_outcontent\_copy

### Type Annotations

TypeScript let’s you add type annotation to your variable and function declarations. This helps you with better tooling like intellisense and to catch type errors.

If you are interested in learning more about [TypeScript type annotations, I wrote an article about it that you are more than welcome to read](https://www.barbarianmeetscoding.com/blog/2017/09/07/typescript-javascript-plus-types-equals-awesome-developer-productivity-type-annotations/).

Now that we have defined the core object of our domain model let’s create our very first component: the PeopleListComponent. Againt we take advantage of the [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/) and type the following:

|  |
| --- |
| PS> ng generate component --inline-template people-list  # or in short-hand:  # ng g c -it people-list |

all\_outcontent\_copy

This will generate a new folder people-list and will place a [TypeScript](https://www.barbarianmeetscoding.com/blog/2017/08/17/typescript-javascript-and-types-equal-awesome-developer-productivity/) and a style files within that folder. Since we have selected the --inline-template option, instead of using an external template file the component will use the template property inside the @Component decorator to define an inline template.

It will look like this:

|  |
| --- |
| import { Component, OnInit } from '@angular/core';  @Component({  selector: 'app-people-list',  template: `  <p>  people-list Works!  </p>  `,  styleUrls: ['./people-list.component.scss']  })  export class PeopleListComponent implements OnInit {  // you can ignore these for the time being :)  constructor() {}  ngOnInit() {}  } |

all\_outcontent\_copy

### app-people-list? Why App?

By default, when you create a new app with the [angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/) all your new components will be prefixed with the app prefix. That’s because the custom elements standard requires custom components to be named with at least two letters separated by a dash to avoid conflicting with built-in browser elements like p, section, body and head.

You can change the prefix for you app in the angular-cli.json file that contains the [angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/)configuration for your project. You can also set a prefix during project creation using the prefix flag with ng new

Ok, now we’ve got a component that we can render using the <app-people-list> element. Indeed, you can appreciate how the [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/) creates a dummy template so you can test that the component works right away. But we still need to do some changes before we can display a list of Star Wars celebrities.

The next thing that we’ll do is to expose a people property with an array of Person with the mighty Luke, Darth Vader and Solo.

|  |
| --- |
| import { Component, OnInit } from '@angular/core';  import { Person } from '../person';  @Component({  selector: 'app-people-list',  template: `  <p>  people-list Works!  </p>  `,  styleUrls: ['./people-list.component.scss']  })  export class PeopleListComponent{  people: Person[] = [  {name: 'Luke Skywalker', height: 177, weight: 70},  {name: 'Darth Vader', height: 200, weight: 100},  {name: 'Han Solo', height: 185, weight: 85},  ];  constructor(){}  ngOnInit(){}  } |

all\_outcontent\_copy

And now we’ll update its template so that we can represent a list of star wars people. Just like ng-repeat in AngularJS, Angular 2 provides a repeater directive that let’s us repeat a part of a template: \*ngFor:

|  |
| --- |
| <!-- this is the new syntax for ng-repeat -->  <ul>  <li \*ngFor="let person of people">  {{person.name}}  </li>  </ul> |

all\_outcontent\_copy

The \* before the ngFor denotes the fact that \*ngFor is what is called a structural directive, a directive that is going to affect the DOM in some way adding or removing elements to it. Soon you’ll notice how Angular 2 uses a lot of these cues in their syntax to convey a specific meaning and help you learn and understand what the different directives and parts of the framework do.

The let person creates a local template variable that contains each item within the people array and is bound to each li element.

Putting everything together we get:

|  |
| --- |
| import { Component, ngOnInit } from '@angular/core';  import { Person } from '../person';  @Component({  selector: 'app-people-list',  template: `  <!-- this is the new syntax for ng-repeat -->  <ul>  <li \*ngFor="let person of people">  {{person.name}}  </li>  </ul>  `  })  export class PeopleListComponent{  people: Person[] = [  {name: 'Luke Skywalker', height: 177, weight: 70},  {name: 'Darth Vader', height: 200, weight: 100},  {name: 'Han Solo', height: 185, weight: 85},  ];  constructor(){}  ngOnInit(){}  } |

all\_outcontent\_copy

### Why Does that String Delimiter Look So Weird?

That’s because it is a backtick (`). Backticks are used to define the new ES6 template strings. They are a new great feature of ES6 that let’s you write multiline strings natively and insert any expression directly within a string with a very straightforward syntax.

[If you want to know more about the marvels of template strings and how the revolutionize JavaScript string formatting check this article](http://www.barbarianmeetscoding.com/blog/2015/10/19/mastering-the-arcane-art-of-javascript-mancy-for-csharp-developers-a-guide-to-strings-finding-the-right-words-and-proper-spell-intonation/).

Now we can update our AppComponent to display the list of StarWars people. We only need to update the app.component.html template:

|  |
| --- |
| <h1>{{title}}</h1>  <app-people-list></app-people-list> |

all\_outcontent\_copy

And voilà!!!!! It automagically works! (again check the browser for an update or run ng serve -o).

Once thing that is interesting to mention is that if you had followed the previous steps but you had created the new component with your bare hands it would not have worked. Why is that? That’s because the [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/)does a lot of things for you in the background. For instance, when we created the component before, the CLI not only created the component itself but it also wired it up so we could use it in our application.

Indeed, in order to be able to use a component within another component you need to make your application aware of it. You can do that in the app.module.ts file, that is, in your Angular 2 module. If you take a look at this file you’ll discover the following:

|  |
| --- |
| import { AppComponent } from './app.component';  // who did this?  import { PeopleListComponent } from './people-list/people-list.component';  @NgModule({  declarations: [  AppComponent,  PeopleListComponent // and this??  ],  // etc...  })  export class AppModule { } |

all\_outcontent\_copy

Exactly! The [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/) has wired it up for you and saves you the need from remembering to do it yourself. Magnificent! Thor bless the [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/)!

## Concluding

Good job! You’ve started your path to learning Angular 2! Yey! We touched a ton of stuff in this blog post, Angular, [TypeScript](https://www.barbarianmeetscoding.com/blog/2017/08/17/typescript-javascript-and-types-equal-awesome-developer-productivity/), the [Angular CLI](https://www.barbarianmeetscoding.com/blog/2016/08/24/angular-2-cli-bootstrap-your-angular-2-app-in-a-breeze/), modules, Webpack, bootstrapping, components, decorators, classes and template strings. **Wow!** Hope you’ve enjoyed the tutorial!