### Introduction to Angular 2

Angular 2 is an open source JavaScript framework to build web applications(SPA) in HTML and JavaScript.

It was built by Google developers.

Basic web development and JavaScript are the Prerequisites for Angular2

**Angular 1+** was released in October 2010, and by far the most popular JavaScript framework available for creating web applications. Many developers are already using Angular 1, so the obvious question that comes to our mind is why we should use Angular 2.

**Angular 2+** is not a simple upgrade from angular 1. Angular 2 is completely rewritten, so it has lot of improvements when compared with Angular 1.

**Angular 2 improvements**

**Performance:** From a performance standpoint, Angular 2 has faster initial loads, change detection, and improved rendering time. Not just performance, we also have improved modularity, Dependency injection and testability.

**Mobile Support:** Angular 1 was not built for mobile devices. It is possible to run Angular 1 on mobile but we will have to use other frameworks. Angular 2 on the other hand is designed from the ground up with mobile support. So with Angular 2 we can build a single application that works across mobile and desktop devices.

**Component Based Development:** Component based web development is the future of web development. In Angular 2, "everything is a component". Components are the building blocks of an Angular application. The advantage of the component-based approach is that, it facilitates greater code reuse. From unit testing standpoint, the use of components makes Angular2 more testable.

Features of Angular 2

* **Components** − The earlier version of Angular had a focus of Controllers but now has changed the focus to having components over controllers. Components help to build the applications into many modules. This helps in better maintaining the application over a period of time.
* **TypeScript** − The newer version of Angular is based on TypeScript. This is a superset of JavaScript and is maintained by Microsoft.
* **Services** − Services are a set of code that can be shared by different components of an application. So for example if you had a data component that picked data from a database, you could have it as a shared service that could be used across multiple applications.

In addition, Angular 2 has better event-handling capabilities, powerful templates, and better support for mobile devices.

To start working with Angular 2, you need to get the following key components installed.

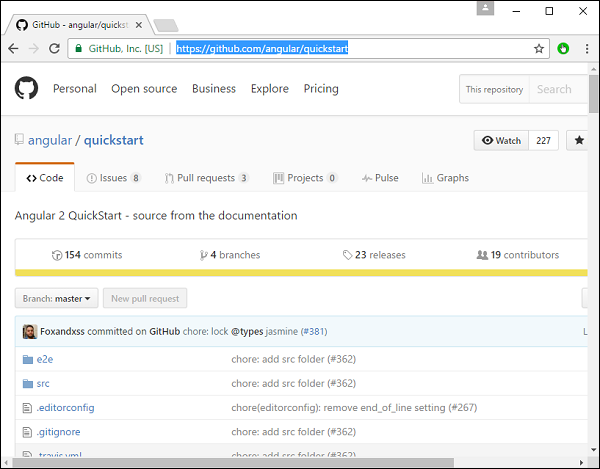
* **Npm** − This is known as the node package manager that is used to work with the open source repositories. Angular JS as a framework has dependencies on other components. And **npm** can be used to download these dependencies and attach them to your project.
* **Git** − This is the source code software that can be used to get the sample application from the **github** angular site.
* **Editor** − There are many editors that can be used for Angular JS development such as Visual Studio code and WebStorm and Visual Studio 2017 and later.

**How to get started with your first Angular JS application??**

* One way is to do everything from scratch which is the most difficult and not the preferred way. Due to the many dependencies, it becomes difficult to get this setup.
* Another way is to use the quick start at Angular Github. This contains the necessary code to get started. This is normally what is opted by all developers.
* The final way is to use Angular CLI. (We will discuss later)

Following are the steps to get a sample application up and running via github.

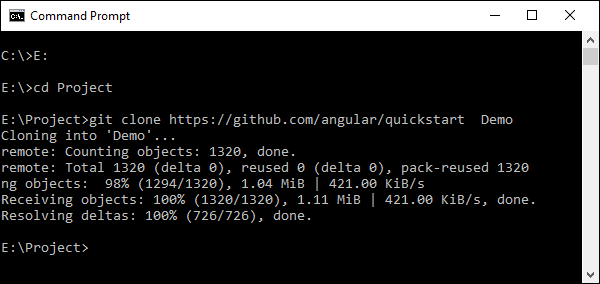
**Step 1** − Go the github url - <https://github.com/angular/quickstart>



**Step 2** − Go to your command prompt, create a project directory. This can be an empty directory. In our example, we have created a directory called Project.

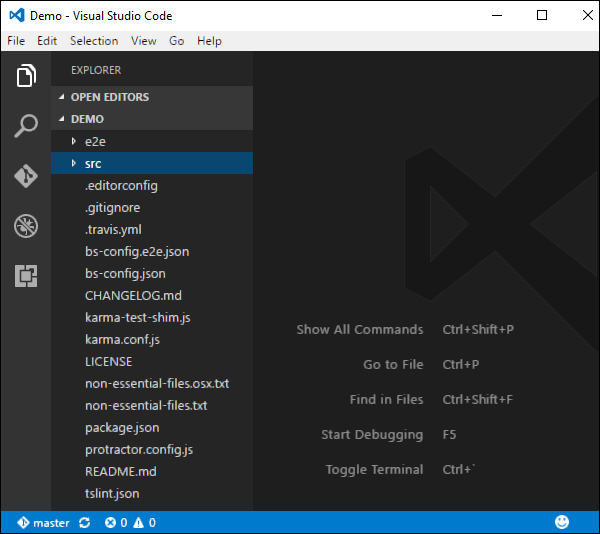
**Step 3** − Next, in the command prompt, go to this directory and issue the following command to clone the github repository on your local system. You can do this by issuing the following command −

git clone https://github.com/angular/quickstart Demo



This will create a sample Angular JS application on your local machine.

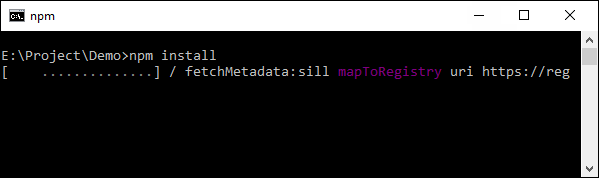
**Step 4** − Open the code in Visual Studio code.



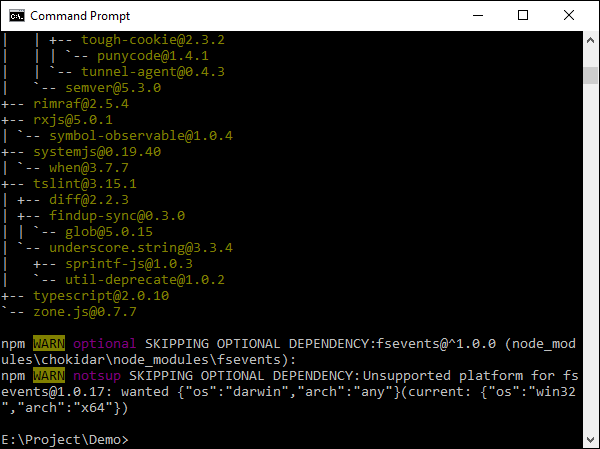
**Step 5** − Go to the command prompt and in your project folder again and issue the following command −

npm install

This will install all the necessary packages which are required for the Angular JS application to work.



Once done, you should see a tree structure with all dependencies installed.



**Step 6** − Go to the folder Demo → src → app → app.component.ts. Find the following lines of code −

import { Component } from '@angular/core';

@Component ({

selector: 'my-app',

template: `<h1>Hello {{name}}</h1>`,

})

export class AppComponent { name = 'Angular'; }

And replace the Angular keyword with World as shown below −

import { Component } from '@angular/core';

@Component ({

selector: 'my-app',

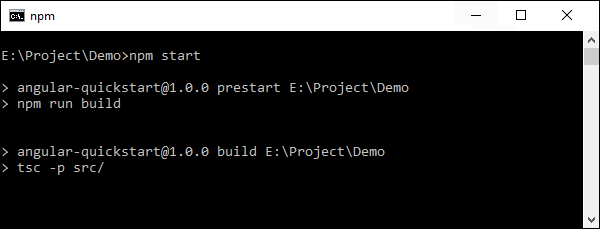
template: `<h1>Hello {{name}}</h1>`,

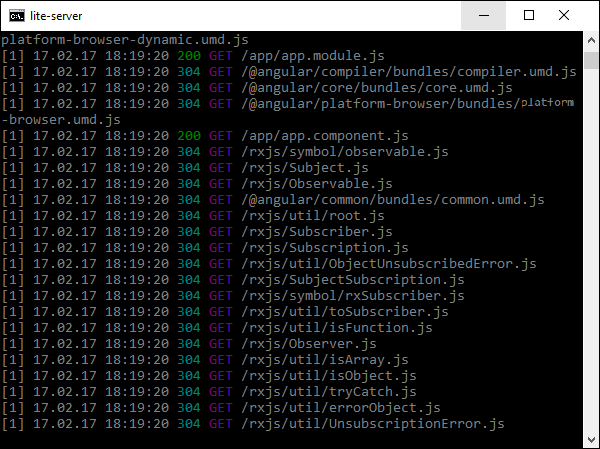
})

export class AppComponent { name = 'World'; }

**Note** − Visual Studio Code will automatically compile all your files and create JavaScript files for all your typescript files.

**Step 7** − Now go to your command prompt and issue the command npm start. This will cause the Node package manager to start a lite web server and launch your Angular application.





The Angular JS application will now launch in the browser and you will see “Hello World” in the browser as shown in the following screenshot.



## Components of Angular 2

Angular 2 has the following components −

* **Modules** − In Angular, a module is a mechanism to group components, directives, pipes and services that are related, in such a way that can be combined with other modules to create an application.
* **Component** − Components are the most basic building block of an UI in an Angular application.
* **Templates** − This is used to define the views of an Angular JS application.
* **decorator**− This can be used to add more data to an Angular JS class.
* **Service** − This is used to create components which can be shared across the entire application.

**Angular 2 Modules**

To be able to define modules we have to use the decorator NgModule.

import { NgModule } from '@angular/core';

@NgModule({

imports: [ ... ],

declarations: [ ... ],

bootstrap: [ ... ]

})

export class AppModule { }

In the example above, we have turned the class AppModule into an Angular module just by using the NgModule decorator. The NgModule decorator requires at least three properties: imports, declarations and bootstrap.

The property imports expects an array of modules. Here's where we define the pieces of our puzzle (our application). The property declarations expects an array of components, directives and pipes that are part of the module. The bootstrap property is where we define the root component of our module. Even though this property is also an array, 99% of the time we are going to define only one component.

+

# Angular 2 – Components

A component in Angular is a class with a template and a decorator. So in simple terms a component in Angular is composed of these 3 things

**Template** - Defines the user interface. Contains the HTML, directives and bindings.

**Class** - Contains the code required for template. Just like a class in any object oriented programming language like C# or Java, a class in angular can contain methods and properties. Properties contain the data that we want to display in the view template and methods contain the logic for the view. We use TypeScript to create the class.

**Decorator** - We use the Component decorator provided by Angular to add metadata to the class. A class becomes an Angular component, when it is decorated with the Component decorator.

## Class

The class decorator. The class is defined in TypeScript. The class normally has the following syntax in TypeScript.

### Syntax

class classname {

Propertyname: PropertyType = Value

}

### Parameters

* **Classname** − This is the name to be given to the class.
* **Propertyname** − This is the name to be given to the property.
* **PropertyType** − Since TypeScript is strongly typed, you need to give a type to the property.
* **Value** − This is the value to be given to the property.

### Example

export class AppComponent {

appTitle: string = 'Welcome';

}

In the example, the following things need to be noted −

* We are defining a class called AppComponent.
* The export keyword is used so that the component can be used in other modules in the Angular JS application.
* appTitle is the name of the property.

## Template

This is the view which needs to be rendered in the application.

### Syntax

Template: '

<HTML code>

class properties

'

### Parameters

* **HTML Code** − This is the HTML code which needs to be rendered in the application.
* **Class properties** − These are the properties of the class which can be referenced in the template.

### Example

template: '

<div>

<h1>{{appTitle}}</h1>

<div>To Tutorials Point</div>

</div>

'

In the example, the following things need to be noted −

* We are defining the HTML code which will be rendered in our application
* We are also referencing the appTitle property from our class.

Decorator

This is used to decorate Angular JS class with additional information.

Let’s take a look at the completed code with our class, template, and metadata.

### Example

import { Component } from '@angular/core';

@Component ({

selector: 'my-app',

template: ` <div>

<h1>{{appTitle}}</h1>

<div>Hello World from Angular</div>

</div> `,

})

export class AppComponent {

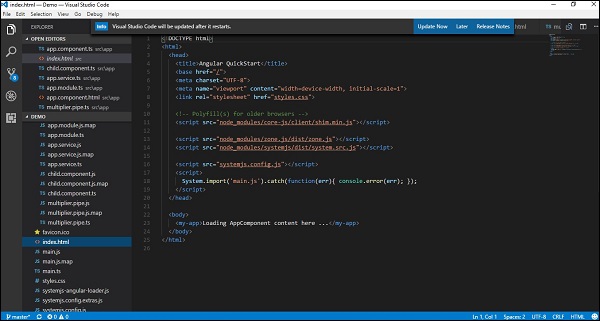
appTitle: string = 'Welcome';

}

In the above example, the following things need to be noted −

* We are using the import keyword to import the ‘Component’ decorator from the angular/core module.
* We are then using the decorator to define a component.
* The component has a selector called ‘my-app’. This is nothing but our custom html tag which can be used in our main html page.

Now, let’s go to our index.html file in our code.



Let’s make sure that the body tag now contains a reference to our custom tag in the component. Thus in the above case, we need to make sure that the body tag contains the following code −

<body>

<my-app></my-app>

</body>

### Output

Now if we go to the browser and see the output, we will see that the output is rendered as it is in the component.

# Angular 2 - Templates

template: '

<div>

<h1>{{appTitle}}</h1>

<div>To Tutorials Point</div>

</div>

'

This is known as an **inline template**. There are other ways to define a template and that can be done via the templateURL command. The simplest way to use this in the component is as follows.

## Syntax

templateURL:

viewname.component.html

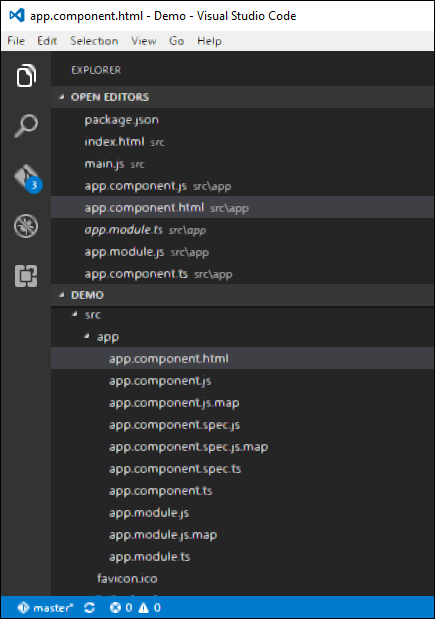
## Parameters

* **viewname** − This is the name of the app component module.

After the viewname, the component needs to be added to the file name.

Following are the steps to define an inline template.

**Step 1** − Create a file called app.component.html. This will contain the html code for the view.



**Step 2** − Add the following code in the above created file.

<div>{{appTitle}} Tutorialspoint </div>

This defines a simple div tag and references the appTitle property from the app.component class.

**Step 3** − In the app.component.ts file, add the following code.

import { Component } from '@angular/core';

@Component ({

selector: 'my-app',

templateUrl: 'app/app.component.html'

})

export class AppComponent {

appTitle: string = 'Welcome';

}

From the above code, the only change that can be noted is from the templateURL, which gives the link to the app.component.html file which is located in the app folder.

**Step 4** − Run the code in the browser, you will get the following output.

Angular template vs templateurl

Decorator with template(inline)

import { Component } from '@angular/core';

@Component({

    selector: 'my-app',

    template: `<h1>Hello {{name }}</h1>`

})

export class AppComponent {

    name: string = 'Angular';

}

Decorator with templateUrl

import { Component } from '@angular/core';

@Component({

    selector: 'my-app',

    templateUrl: 'app/app.component.html'

})

export class AppComponent {

    name: string = "Angular";

}

**What are the differences between template and templateUrl properties and when to use one over the other**  
Angular2 recommends to extract templates into a separate file, if the view template is longer than 3 lines. Let's understand why is it better to extract a view template into a seprate file, if it is longer than 3 lines.  
  
**With an inline template** 

1. We loose Visual Studio editor intellisense, code-completion and formatting features.
2. TypeScript code is not easier to read and understand when it is mixed with the inline template HTML.

**With an external view template**

1. We have Visual Studio editor intellisense, code-completion and formatting features and
2. Not only the code in "app.component.ts" is clean, it is also easier to read and understand