Angular 2.0

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Angular 2 is an open source JavaScript framework to build web applications in HTML and JavaScript, and has been conceived as a mobile first approach.

Audience

This tutorial is designed for software professionals who want to learn the basics of AngularJS 2 and its programming concepts in simple and easy steps. It describes the components of AngularJS 2 with suitable examples.

Prerequisites

You should have a basic understanding of JavaScript and any text editor. As we are going to develop web-based applications using Angular 2, it will helpful if you have an understanding of other web technologies such as HTML, CSS, AJAX, AngularJS, etc.

Copyright & Disclaimer

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

tutorialspoint
SIMPLYEASYLEARNING

1. Angular 2 — Overview

Angular 2 is an open source JavaScript framework to build web applications in HTML and JavaScript, and has been conceived as a mobile first approach. The beta version of Angular 2 was released in March 2014.

Why use Angular 2?

- Angular 2 is simpler than Angular 1. The concepts here are easier to understand.
- You can update the large data sets with minimal memory overhead.
- It will speed up the initial load through server side rendering.

Features of Angular 2

- Angular 2 is faster and easier than Angular 1.
- It supports the latest version of browsers and also supports old browsers including IE9+ and Android 4.1+.
- It is a cross-platform framework.
- Angular 2 is mainly focused on mobile apps.
- Code structure is more simplified than the previous version of Angular.

Advantages of Angular 2

- If an application is heavy, then Angular 2 keeps it fully UI (User Interface) responsive.
- It uses the server side rendering for fast views on mobile.
- It works well with ECMAScript and other languages that compile with JavaScript.
- It uses dependency injection to maintain applications without writing lengthy codes.
- The applications here have a component-based approach.

Disadvantages of Angular 2

- Since Angular 2 is a newly introduced framework, there is less online community support.
- It takes time to learn if you are new to Angular 2.

tutorialspoint
SIMPLYEASYLEARNING

2. Angular 2 — Environment

In this chapter, let us discuss the Angular 2 development environment in detail.

- Angular uses TypeScript, which is a primary language for developing Angular applications.
- TypeScript is a super set of JavaScript, which is migrated to TypeScript. Here, the code written in TypeScript makes it less prone to runtime errors.

To set up the development environment, follow these steps:

Step 1: Create a project folder in your local drive by typing the commands in the command prompt as given below.

```
mkdir angular2-demo
cd angular2-demo
```

Creating Configuration Files

The creation of configuration files follows the step mentioned above.

Step 2: You need to create **tsconfig.json** which is the TypeScript compiler configuration file. It guides the compiler to generate JavaScript files.

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "system",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  },
  "exclude": [
    "node_modules",
    "typings/main",
    "typings/main.d.ts"
  ]
}
```

Step 3: Create a **typings.json** file in your project folder *angular2-demo* as shown below:

typings.json

```
{
  "globalDependencies": {
    "core-js": "registry:dt/core-js#0.0.0+20160602141332",
    "jasmine": "registry:dt/jasmine#2.2.0+20160621224255",
    "node": "registry:dt/node#6.0.0+20160621231320"
  }
}
```

A large number of libraries of the JavaScript extends JavaScript environment with features and syntax which is not natively recognized by the TypeScript compiler. The **typings.json** file is used to identify TypeScript definition files in your Angular application.

In the above code, there are three typing files as shown below:

- **core-js**: It brings ES2015/ES6 capabilities to our ES5 browsers.
- **jasmine**: It is the typing for Jasmine test framework.
- **node**: It is used for the code that references objects in the **nodejs** environment.

These typing files are used in the development of larger Angular applications.

Step 4: Add the **package.json** file to your angular2-demo project folder with the code given below:

package.json

```
{
  "name": "angular2-demo",
  "version": "1.0.0",
  "scripts": {
    "start": "concurrent \"npm run tsc:w\" \"npm run lite\" ",
    "tsc": "tsc",
    "tsc:w": "tsc -w",
    "lite": "lite-server",
    "typings": "typings",
    "postinstall": "typings install"
  },

  "license": "ISC",
  "dependencies": {
    "angular2": "2.0.0-beta.7",
    "systemjs": "0.19.22",
    "es6-promise": "^3.0.2",
    "es6-shim": "^0.33.3",
    "reflect-metadata": "0.1.2",
    "rxjs": "5.0.0-beta.2",
    "zone.js": "0.5.15"
  },
  "devDependencies": {
    "concurrently": "^2.0.0",
    "lite-server": "^2.1.0",
    "typescript": "^1.7.5",
    "typings":"^0.6.8"
  }
}
```

The **package.json** will contain the packages that our apps require. These packages are installed and maintained with **npm** (Node Package Manager). To install *npm* click here.

Step 5: To install packages, run the **npm** command in the command prompt as given below.

```
npm install
```

Error messages in red may appear while installing npm. These messages have to be ignored.

Creating Our First Angular Component

A component is the fundamental concept of Angular. A component is a class that controls a view template - a part of a web page where information to the user is displayed and user feedback is responded to. Components are required to build Angular apps.

Step 6: Create a sub-folder called *app/* inside your project folder to place the Angular app components. You can use the following command to create the folder:

```
mkdir app
cd app
```

Step 7: The files which you create need to be saved with the **.ts** extension. Create a file called **environment_app.component.ts** in your *app/* folder with the below code:

environment_app.component.ts

```
import {Component, View} from "angular2/core";

@Component({
   selector: 'my-app'
})

@View({
  template: '<h2>My First Angular 2 App</h2>'
})

export class AppComponent {

}
```

- The above code will import the ***Component*** and the ***View*** package from ***angular2/core***.
- The ***@Component*** is an Angular 2 ***decorator*** that allows you to associate metadata with the component class.
- The ***my-app*** can be used as HTML tag and also as a component.
- The ***@view*** contains a ***template*** that tells Angular how to render a view.
- The ***export*** specifies that, this component will be available outside the file.

Step 8: Next, create the **environment_main.ts** file with the following code:

environment_main.ts

```
import {bootstrap} from "angular2/platform/browser"
import {AppComponent} from "./environment_app.component"

bootstrap(AppComponent);
```

- The ***environment_main.ts*** file tells Angular to load the component.

- To launch the application, we need to import both ***Angular's browser bootstrap*** function and ***root component of the application***.
- After importing, the ***bootstrap*** is called by passing the ***root component type***, i.e., ***AppComponent***.

Step 9: Now create **index.html** in your project folder ***angular2-demo/*** with the code given below:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
    <script    src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
    <script              src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
    <script src="https://code.angularjs.org/2.0.0-
beta.6/angular2.dev.js"></script>
    <script>
      System.config({
        transpiler: 'typescript',
        typescriptOptions: { emitDecoratorMetadata: true },

        packages: {'app': {defaultExtension: 'ts'}},
        map: { 'app': './angular2/src/app' }
      });
      System.import('app/environment_main')
            .then(null, console.error.bind(console));
    </script>
  </head>
<body>
   <my-app>Loading...</my-app>
</body>
```

```
</html>
```

Angular will launch the app in the browser with our component. The app is further placed in a specific location on ***index.html***.

Compile and Run

After all the steps that we performed above, let us now perform the following step to compile and run.

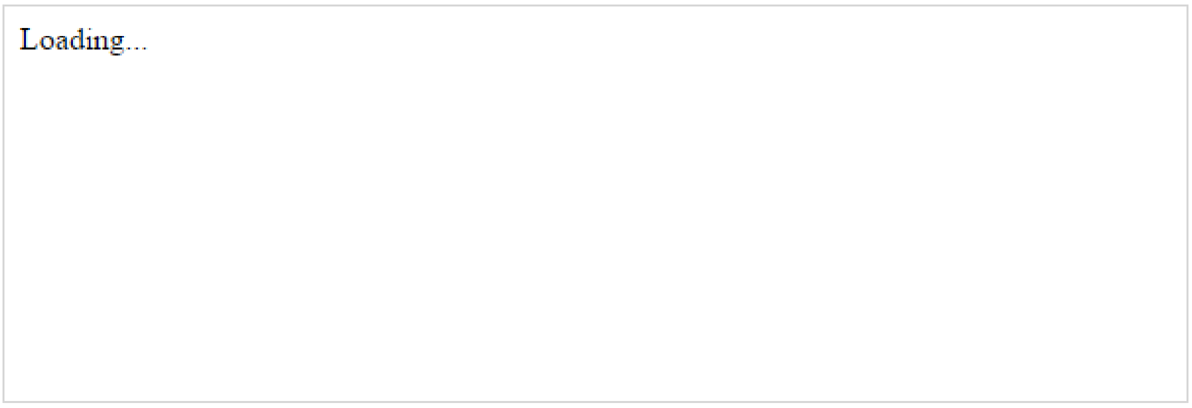
Step 10: To run the application, type the command in a terminal window as given below:

```
npm start
```

The above command runs two parallel node processes as listed below:

- TypeScript compiler in the watch mode.
- The **lite-server (static server)** loads the ***index.html*** in a browser and refreshes the browser as application files change.

After a few moments, a browser tab will open with the following output:



```
Loading...
```

3. Angular 2 — Hello World

In the previous chapter, we studied how to set up the development environment for Angular 2. In this chapter, let us create an example to display the *Hello World* text.

Example

The following example describes how to display a simple text in Angular 2:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
    <script     src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
    <script              src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
    <script src="https://code.angularjs.org/2.0.0-
beta.6/angular2.dev.js"></script>
    <script>
      System.config({
        transpiler: 'typescript',
        typescriptOptions: { emitDecoratorMetadata: true },
        packages: {'app': {defaultExtension: 'ts'}},
        map: { 'app': './angular2/src/app' }
      });
      System.import('app/hello_world_main')
            .then(null, console.error.bind(console));
    </script>
  </head>
<body>
   <my-app>Loading...</my-app>
</body>
```

```
</html>
```

The above code includes the following configuration options:

- You can configure the *index.html* file using the *typescript* version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the *transpiler* option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors which are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the *emitDecoratorMetadata* option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and creates an impact on the application runtime.
- Angular 2 includes the packages from the *app* folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the 'app' selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files which you need to save under the *app* folder.

hello_world_main.ts

```
import {bootstrap} from "angular2/platform/browser"
import {MyHelloWorldClass} from "./hello_world_app.component"

bootstrap(MyHelloWorldClass);
```

We will now create a component in **TypeScript(.ts)** file as shown below:

hello_world_app.component.ts

```
import {Component, View} from "angular2/core";

@Component({
   selector: 'my-app'
})

@View({
  template: '<h2>Hello World !!</h2>'
})
```

```
export class MyHelloWorldClass {

}
```

- The *@Component* is a decorator that uses the configuration object to create the component.
- The *selector* creates an instance of the component where it finds **<my-app>** tag in parent HTML.
- The *@view* contains a *template* that tells Angular how to render a view.
- The *export* specifies that the component will be available outside the file.

Output

Let us now proceed with the following steps to see how the above code works:

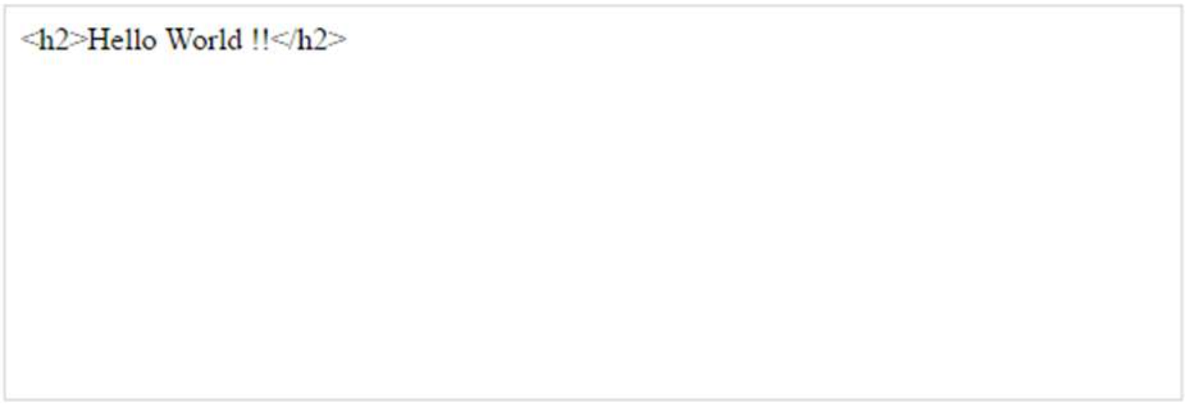
- Save the above HTML code as **index.html** file, the way we created in environment chapter and use the above *app* folder which contains ***.ts*** files.
- Open the terminal window and enter the command given below:

```
npm start
```

- After a few moments, a browser tab should open. The tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **angular2_hello_world.html** file in your server root folder.
- Open this HTML file as **http://localhost/angular2_hello_world.html**. You will receive the following output.

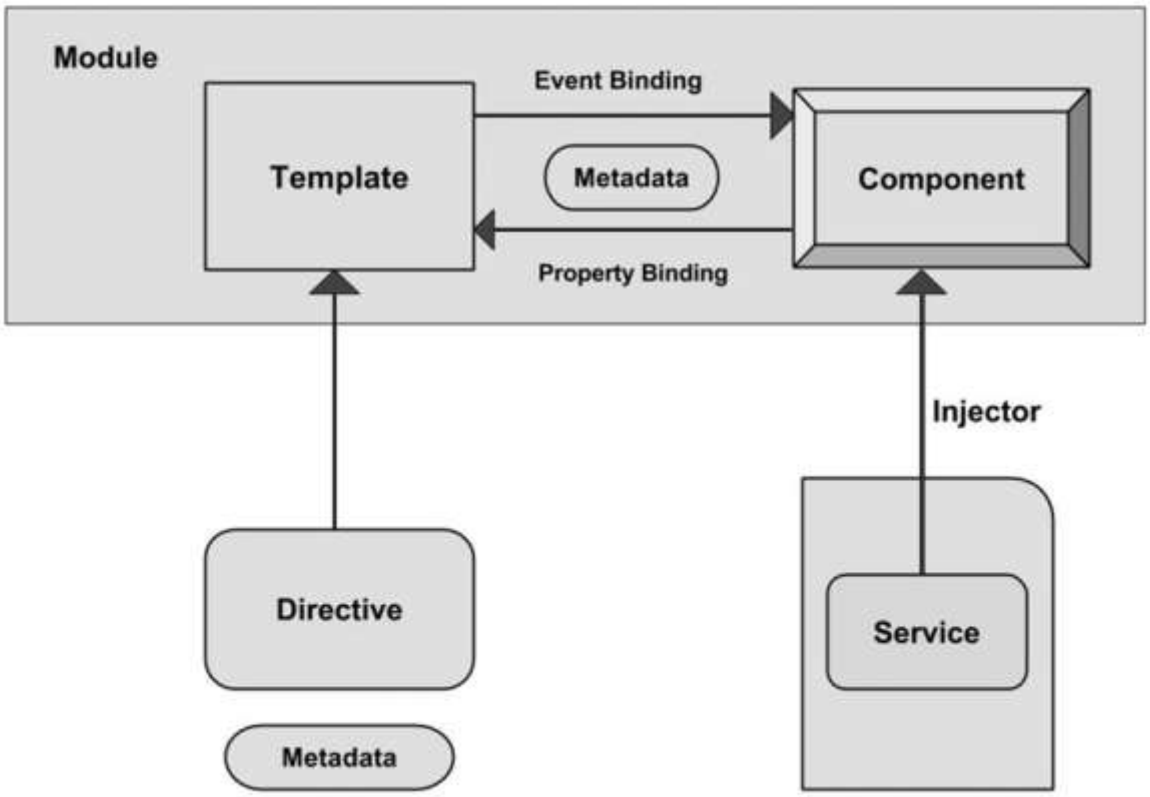


```
<h2>Hello World !!</h2>
```

4. Angular 2 — Architecture

In this chapter, we will discuss the architectural style of Angular 2 framework for implementing user applications.

The following diagram shows architecture of Angular 2:



The architecture of Angular 2 contains the following modules:

- Module
- Component
- Template
- Metadata
- Data Binding
- Service
- Directive
- Dependency Injection

Module

The module component is characterized by a block of code which can be used to perform a single task. You can export the value of something such as a class from the code. The Angular apps are called modules. Applications are built using many modules. The basic building block of Angular 2 is a component class, which can be exported from a module.

Some of the applications will have a component class named as **AppComponent** and you can find it in a file called **app.component.ts**. Use the export statement to export the component class from the module as shown below:

```
export class AppComponent { }
```

The *export* statement specifies that it is a module and its *AppComponent* class is defined as public. The *AppComponent* class can be accessible to other modules of the application.

Component

A component is a **controller class** with a template which mainly deals with a view of the application and logic on the page. It is a bit of code that can be used throughout an application. Component knows how to render itself and configure dependency injection. You can associate CSS styles using component inline styles, style URLs and template inline styles to a component.

To register component, we use the ***@Component*** annotation. This annotation can be used to break the application into smaller parts. There will be the only one component per DOM element.

Template

The component's view can be defined by using the *template* which tells Angular how to display the component. The following template shows how to display the name:

```
<div>
Your name is : {{name}}
</div>
```

To display the value, you can put the template expression within the interpolation braces.

Metadata

Metadata is a way of processing the class. Consider we have one component called *MyComponent*, which will be a class until we tell Angular that it's a component. You can use *metadata* to the class to tell Angular that *MyComponent* is a component. The metadata can be attached to TypeScript by using the *decorator*.

For instance:

```
@Component({
   selector : 'mylist',
    template : '<h2>Name is Harry</h2>'
    directives : [MyComponentDetails]
})
export class ListComponent{...}
```

The *@Component* is a decorator that uses the configuration object to create the component and its view. The *selector* creates an instance of the component where it finds the **<mylist>** tag in parent HTML. The *template* tells Angular how to display the component. The *directive* decorator is used to represent the array of components or directives.

Data Binding

Data binding is a process of coordinating application data values by declaring bindings between sources and target HTML elements. It combines the template parts with components parts and the template HTML is bound with markup to connect both sides. There are four types of data binding:

- **Interpolation**: It displays the component value within the div tags.
- **Property Binding**: It passes the property from the parent to the property of the child.
- **Event Binding**: It fires when you click on the component's method name.
- **Two-way Binding**: This form binds the property and the event by using the ***ngModel*** directive in a single notation.

Service

Services are JavaScript functions that are responsible for performing a specific task only. Angular services are injected using the Dependency Injection mechanism. Service includes the value, function or feature which is required by the application. Generally, service is a class which can perform something specific such as logging service, data service, message service, the configuration of application, etc. There is nothing much about service in Angular and there is no **ServiceBase** class, but still services can be treated as fundamental to Angular application.

Directive

The directive is a class that represents the metadata. There are three types of directives:

- **Component Directive**: It creates custom controller by using view and controller and is used as custom HTML element.
- **Decorator Directive**: It decorates the elements using additional behavior.
- **Template Directive**: It converts HTML into a reusable template.

Dependency Injection

Dependency Injection is a design pattern that passes an object as a dependency in different components across the application. It creates a new instance of class along with its required dependencies.

The following points need to be considered while using the Dependency Injection:

- The Dependency Injection is stimulated into the framework and can be used everywhere.
- The ***injector*** mechanism maintains service instance and can be created using a ***provider***.
- The ***provider*** is a way of creating a service.
- You can register the ***providers*** along with injectors.

5. Angular 2 — Modules

The applications in Angular follow the modular structure. The Angular apps will contain many ***modules***, each dedicated to a single purpose. Typically, module is a cohesive group of code which is integrated with the other modules to run your Angular apps.

A module ***exports*** some ***classes***, ***function***, and ***values*** from its code. The ***Component*** is a fundamental block of Angular and multiple ***components*** that make up your application.

A module can be a library for another module. For instance, the ***angular2/core*** library which is a primary Angular library module will be imported by another ***component***.

Example

The following example describes the use of modules in Angular 2:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Angular 2 Modules</title>
      <script  src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
      <script            src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
      <script src="https://code.angularjs.org/tools/system.js"></script>
      <script src="https://code.angularjs.org/tools/typescript.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
      <script                            src="https://code.angularjs.org/2.0.0-
beta.6/angular2.dev.js"></script>
      <script>
         System.config({
           transpiler: 'typescript',
           typescriptOptions: { emitDecoratorMetadata: true },

           packages: {'app': {defaultExtension: 'ts'}},

           map: { 'app': './angular2/src/app' }
         });
```

```
          System.import('app/modules_main')
                .then(null, console.error.bind(console));
       </script>
    </head>
    <body>
       <my-app>Loading...</my-app>
    </body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the *typescript* version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors that are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It further loads the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app' selector**, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files. These files need to save under the ***app*** folder.

modules_main.ts

```
import {bootstrap} from "angular2/platform/browser"
//importing bootstrap function
import {MyModulesClass} from "./modules_app.component"
//importing component function

bootstrap(MyModulesClass);
```

We will now create a component in the **TypeScript(.ts)** file and also a view for the component.

modules_app.component.ts

```
import {Component, View} from "angular2/core";


//framework recognizes @Component annotation and knows that we are trying to
create a new component
@Component({
   selector: 'my-app'
})


@View({
  //this template value will be displayed in the browser
  template: '<h2>Welcome to Tutorialspoint</h2>'
})


export class MyModulesClass { }
```

Output

When you run the above code, it will display the text specified within the ***template*** option, which is defined in the ***modules_app.component.ts*** file. Let's carry out the following steps to see how the above code works:

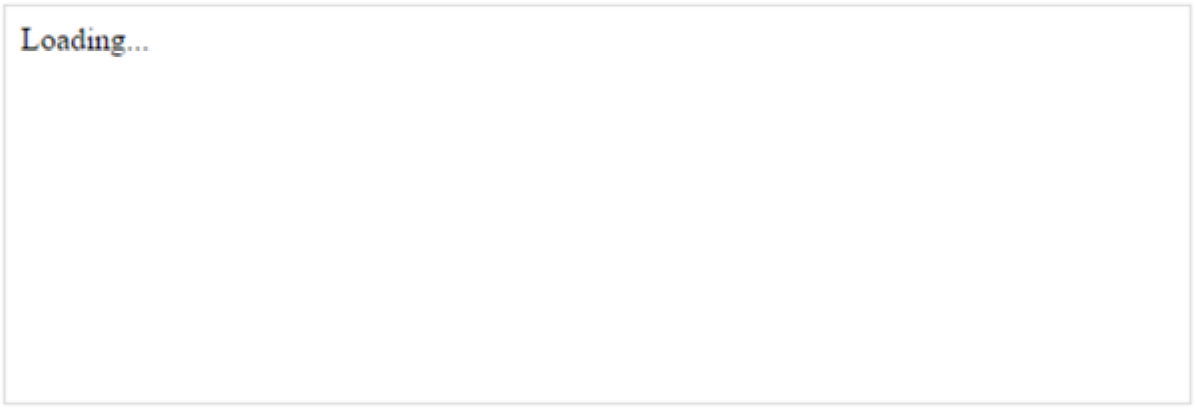
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above ***app*** folder which contains ***.ts*** files.
- Open the terminal window and enter the command given below:

  ```
  npm start
  ```

- After a few moments, a browser tab should open. The tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **angular2_modules.html** file in your server root folder.
- Open the HTML file as **http://localhost/angular2_modules.html.** You will receive the following output.



```
Loading...
```

6. Angular 2 — Components

The component is a controller class with a template which mainly deals with a view of the application and logic on the page. It is a bit of code that can be used throughout an application. The component knows how to render itself and configure dependency injection.

The component contains two important things; one is **view** and the other is **some logic**.

Example

The following example describes the use of component in Angular 2:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Angular 2 Component</title>
      <!--Load libraries -->
      <script   src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>
      <script             src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>
      <script src="https://code.angularjs.org/tools/system.js"></script>
      <script src="https://code.angularjs.org/tools/typescript.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
      <script                                src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>
      <script>
         System.config({
            //transpiler tool converts TypeScript to JavaScript
            transpiler: 'typescript',



            //emitDecoratorMetadata flag used by JavaScript output to create metadata from the decorators
            typescriptOptions: { emitDecoratorMetadata: true },
            packages: {'app': {defaultExtension: 'ts'}},
            map: { 'app': './angular2/src/app' }
```

```
          });
          System.import('app/component_main')
             .then(null, console.error.bind(console));
       </script>
    </head>
    <!--When Angular calls the bootstrap function in main.ts, it reads the
Component metadata, finds the 'app' selector, locates an element tag named app,
and loads the application between those tags.-->
    <body>
       <app>Loading...</app>
    </body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors which are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder. Here, the files have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app' selector**, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files. These files can be saved under the *app* folder.

component_main.ts

```
import {bootstrap} from "angular2/platform/browser";   //importing bootstrap
function
import {App} from "./component_app.component"          //importing component
function


bootstrap(App);
```

We will now create a component in **TypeScript(.ts)** file. We will also create a view for the component.

component_app.component.ts

```
// component's metadata can be accessed using this primary Angular library
import {Component, View} from "angular2/core";


//framework recognizes @Component annotation and knows that we are trying to
create a new component
@Component({
   selector: 'app'  //specifies selector for HTML element named 'app'
})


@View({
  //template property holds component's companion template that tells Angular
how to render a view
  template: '<h2>Welcome to {{name}}</h2>'
})


export class App {
   name : 'Tutorialspoint!!!'
}
```

Output

When you run the above code, it will display the text specified within the ***template*** option, which is defined in the ***component_app.component.ts*** file. The file holds component's companion template that tells Angular how to render a view.

Let's carry out the following steps to see how the above code works:

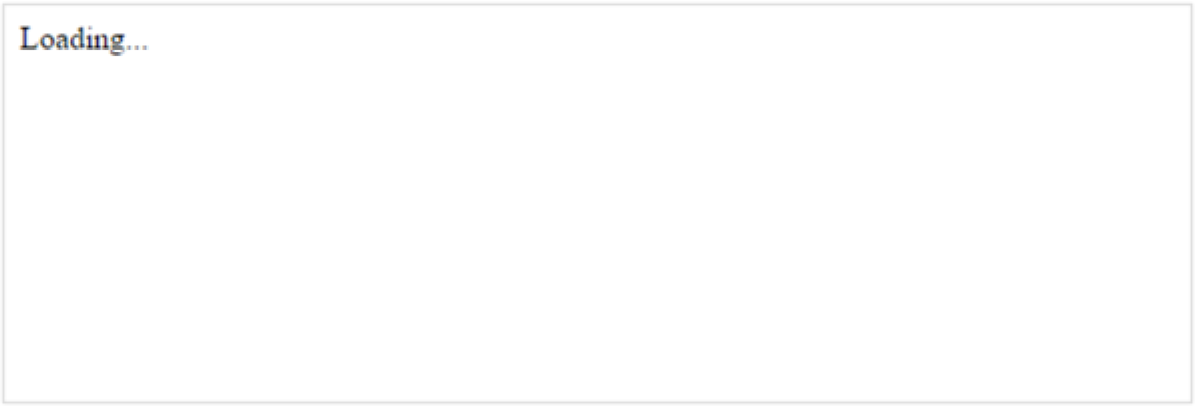
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above *app* folder which contains ***.ts*** files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. The tab displays the output as given below.

OR you can run this file in another way:

- Save the above HTML code as **angular2_components.html** file in your server root folder.
- Open this HTML file as **http://localhost/angular2_components.html**. You will receive the following output.



```
Loading...
```

7. Angular 2 — Templates

The component's view can be defined by using the *template* which tells Angular how to display the component. The template describes how the component is rendered on the page.

Example

The following example describes the use of template in Angular 2:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Angular 2 Template</title>
      <script  src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
      <script src="https://code.angularjs.org/tools/system.js"></script>
      <script src="https://code.angularjs.org/tools/typescript.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>
      <script>
         System.config({
            transpiler: 'typescript',
            typescriptOptions: { emitDecoratorMetadata: true },
            packages: {'app': {defaultExtension: 'ts'}},
            map: { 'app': './angular2/src/app' }
         });
         System.import('app/template_main')
           .then(null, console.error.bind(console));
      </script>
   </head>
   <body>
      <my-app>Loading...</my-app>
   </body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors that are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files which you need to save under the ***app*** folder.

template_main.ts

```
import {bootstrap} from "angular2/platform/browser"   //importing bootstrap function

import {MyTemplate} from "./template_app.component"   //importing component function


bootstrap(MyTemplate);
```

We will now create a component in **TypeScript(.ts)** file. This will create a component and also a view for the component.

template_app.component.ts

```
// component's metadata can be accessed using this primary Angular library
import {Component, View} from "angular2/core";


//framework recognizes @Component annotation and knows that we are trying to create a new component
@Component({
   selector: 'my-app' //specifies selector for HTML element named 'app'
})
```

```
@View({
  //template property holds component's companion template that tells Angular
how to render a view
  template: '<h2>Welcome to the world of {{val}}</h2>'
})

export class MyTemplate {
   val : 'Tutorialspoint!!!'
}
```

Output

When you run the above code, it will display the text specified within the ***template*** option which is defined in the ***template_app.component.ts*** file and holds the component's companion template that tells Angular how to render a view.

Let us now proceed with the following steps to see how above the code works:

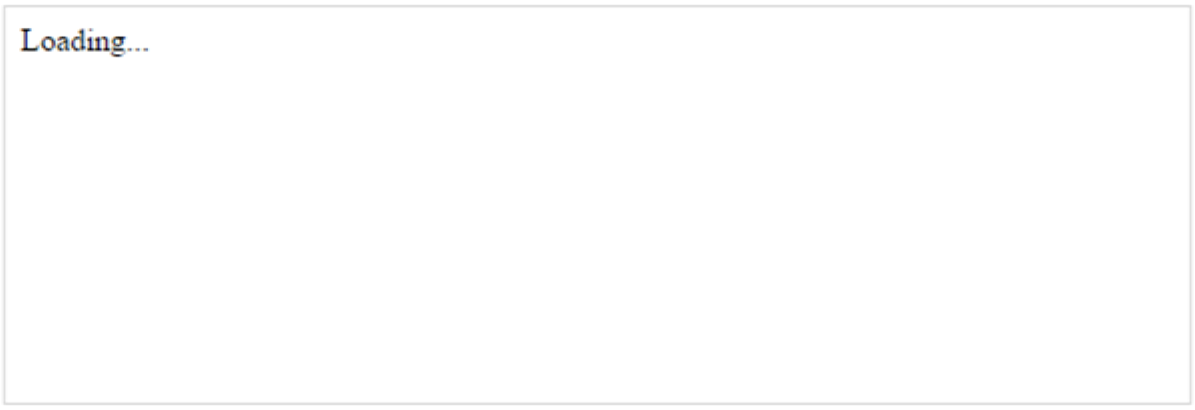
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above ***app*** folder which contains ***.ts*** files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **angular2_templates.html** file in your server root folder.
- Open this HTML file as **http://localhost/angular2_templates.html** and the output as below gets displayed.



```
Loading...
```

8. Angular 2 — Metadata

Metadata is a way of processing the class. Consider you have one component called **MyComponent** which will be a class until you tell Angular that it's a component. You can use metadata to the class to tell Angular that MyComponent is a component and metadata can be attached to **TypeScript** by using the **decorator**.

Example

The following example shows the use of metadata in Angular 2:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Angular 2 Metadata</title>
      <script  src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>
      <script src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>
      <script src="https://code.angularjs.org/tools/system.js"></script>
      <script src="https://code.angularjs.org/tools/typescript.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>
      <script>
         System.config({
           transpiler: 'typescript',
           typescriptOptions: { emitDecoratorMetadata: true },
           packages: {'app': {defaultExtension: 'ts'}},
           map: { 'app': './angular2/src/app' }
         });
         System.import('app/metadata_main')
            .then(null, console.error.bind(console));
      </script>
   </head>
   <body>
      <my-app>Loading...</my-app>
   </body>
```

```
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.

- If you do not transpile to JavaScript before running the application, you could see the compiler warnings and errors which are hidden in the browser.

- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impact on the application runtime.

- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.

- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.

- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app' selector**, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files. These files need to be saved in the ***app*** folder.

metadata_main.ts

```
import {bootstrap} from "angular2/platform/browser"     //importing bootstrap function

import {MyTemplate} from "./metadata_app.component"     //importing component function


bootstrap(MyTemplate);
```

We will now create a component in the **TypeScript(.ts)** file as shown below:

metadata_app.component.ts

```
import {Component} from "angular2/core";

import {ItemComponent} from './item-list.component';


@Component({

    selector: 'my-app',

    template: `<my-list></my-list>`,
    directives:[ItemComponent]
})
export class MyTemplate {}
```

- The ***@Component*** is a decorator that uses the configuration object to create the component and its view.
- The ***selector*** creates an instance of the component where it finds the **<my-app>** tag in the parent HTML.
- The ***directive*** decorator is used to represent the array of components or directives.

The following **TypeScript(.ts)** file displays the list of items on the output.

item-list.component.ts

```
import {Component} from "angular2/core";

@Component({
   selector:'my-list',
   template:`<h2>List of Fruits</h2>
   <ul>
      <li *ngFor="#myItem of itemList">{{myItem.name}}</li>
   </ul>
   `
})

export class ItemComponent{
   public itemList = [
      {name:"Apple"},
      {name:"Orange"},
      {name:"Grapes"},
   ];
}
```

- The ***template*** tells Angular how to display the component.
- The ****ngFor*** directive is used to loop the list of items from the array of ***itemList*** object.

Output

Let's carry out the following steps to see how the above code works:

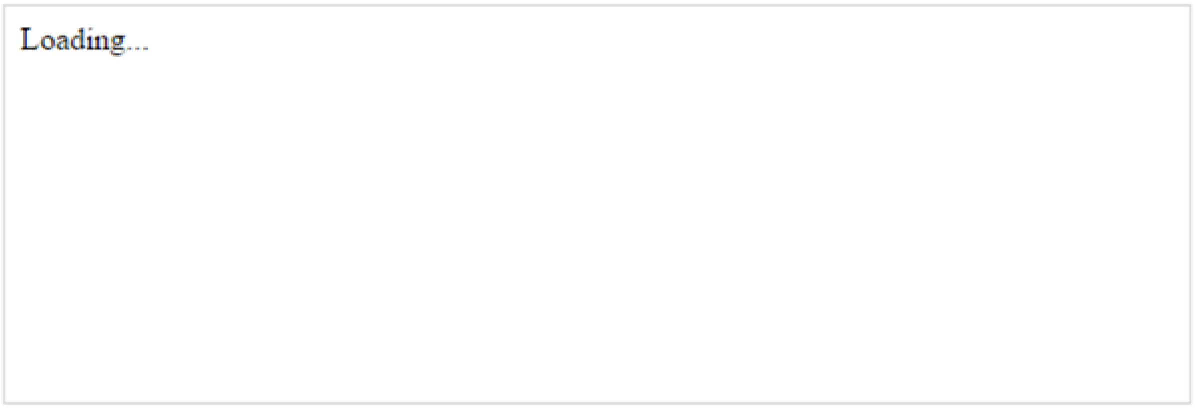
- Save the above HTML code as **index.html** file the way we created in environment chapter and use the above ***app*** folder which contains the ***.ts*** files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. You will receive an output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **angular2_metadata.html** file in your server root folder.
- Open this HTML file as **http://localhost/angular2_metadata.html**. You will receive an output as given below.



```
Loading...
```

9. Angular 2 — Data Binding

Data binding is the synchronization of data between the model and the view components. To display the component property, you can put its name in the view template, enclosed in double curly braces. Two-way data binding merges the property and the event binding in a single notation using the directive ***ngModel***.

Example

The following example describes the use of data binding in Angular 2:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Data Binding</title>
      <script  src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>
      <script          src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>
      <script src="https://code.angularjs.org/tools/system.js"></script>
      <script src="https://code.angularjs.org/tools/typescript.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
      <script                         src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>
      <script>
         System.config({
           transpiler: 'typescript',
           typescriptOptions: { emitDecoratorMetadata: true },
           packages: {'app': {defaultExtension: 'ts'}},
           map: { 'app': './angular2/src/app' }
         });

         System.import('app/data_binding_main')
               .then(null, console.error.bind(console));

      </script>
   </head>
```

```
    <body>
       <my-app>Loading...</my-app>
    </body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors which are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app' selector**, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files. These files need to be saved in the *app* folder.

data_binding_main.ts

```
import {Component} from 'angular2/core';


@Component({
    selector: 'my-app',
    template: `
           <ul>
              <li
              *ngFor="#Item of Items"
              (click)="onItemClicked(Item)">
                  {{ Item.name }}


              </li>
           </ul>
```

```
            <input type="text" [(ngModel)]="clickedItem.name">
    `
})
export class AppComponent {
     public Items = [
                    {name: "Butter"},
                    {name: "Milk"},
                    {name: "Yogurt"},
                    {name: "Cheese"},
                 ];
     public clickedItem = {name: ""};
     onItemClicked(Item) {
        this.clickedItem = Item;
     }
}
```

- The *@Component* is a decorator that uses the configuration object to create the component and its view.
- The *selector* creates an instance of the component where it finds the **<my-app>** tag in the parent HTML.
- Next is the ***ngFor** directive. This creates "**view exports"** which we bind into the template. The ***** is a shorthand for using Angular 2 template syntax with the template tag.
- The local variable *Item* can be referenced in the template. It will get the index of the array. When you click on the item value, the *onItemClicked()* event will get activated and Angular 2 will bind the model name from the array with the local variable of template.
- The model name *clickedItem* is bound with *name* and displays the item name in the text box when the user clicks on the item name from the list.

Output

Let us now proceed with the following steps to see how the above code works:

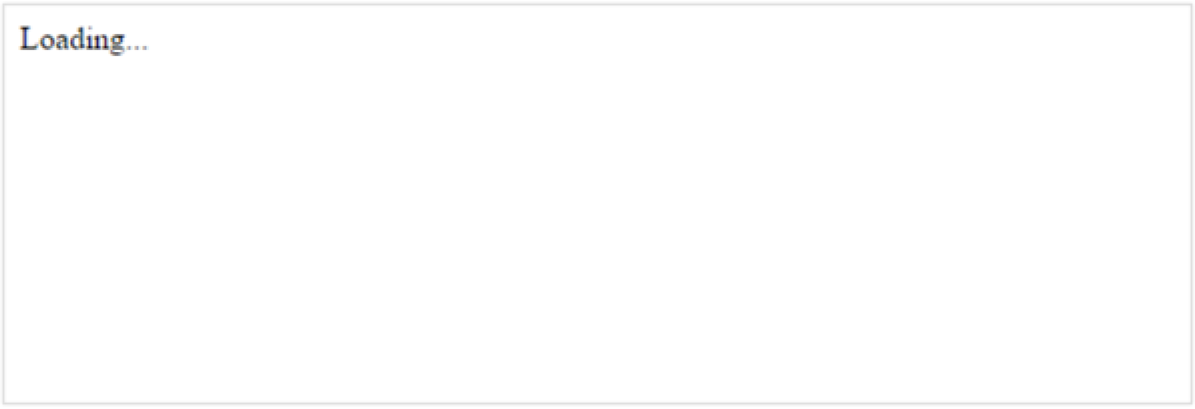
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above ***app*** folder which contains ***.ts*** files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. You will receive an output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **angular2_data_binding.html** file in your server root folder.
- Open this HTML file as **http://localhost/angular2_data_binding.html**. You will receive an output as given below.



```
Loading...
```

The example displays the **item name** in the text box when you click on **item name** from the list.

10. Angular 2 — Data Display

You can display the data with the help of binding controls in the UI. Angular will display the data by using interpolation and other binding properties such as using binding in HTML template to the Angular component properties.

Example

The following example shows the use of displaying data in Angular 2:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Angular 2 Data Display</title>
    <script     src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
    <script               src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
    <script                                src="https://code.angularjs.org/2.0.0-
beta.6/angular2.dev.js"></script>
    <script>
      System.config({
        transpiler: 'typescript',
        typescriptOptions: { emitDecoratorMetadata: true },
        packages: {'app': {defaultExtension: 'ts'}},
        map: { 'app': './angular2/src/app' }
      });



      System.import('app/datadisplay_main')
            .then(null, console.error.bind(console));
    </script>

  </head>
```

```
<body>
   <my-app>Loading...</my-app>
</body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors that are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, the large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the **'app' selector**, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files. These files need to be saved under the ***app*** folder.

datadisplay_main.ts

```
import {bootstrap} from "angular2/platform/browser"
import {MyTemplate} from "./datadisplay_app.component"

bootstrap(MyTemplate);
```

We will now create a component in **TypeScript(.ts)** file as shown below:

datadisplay_app.component.ts

```
import {Component, View} from "angular2/core";

@Component({
   selector: 'my-app'
})
```

```
@View({
  template: `
    <h2>Showing data using component properties with interpolation</h2>
    <h3>Player Name:{{player}}</h3>
    <h3>He is famous in: {{sport}}</h3><br>

    <h2>Showing data using constructor or variable initialization</h2>
    <h3>India capital is: {{capital}}</h3><br>

    <h2>Showing data using array property with NgFor</h2>
    <h3>My favorite fruit is: {{myfruit}}</h3>
    <p>List of Fruits:</p>
    <ul>
       <li *ngFor="#fruit of fruits">
          {{ fruit }}
       </li>
    </ul>
    `
})

export class MyTemplate {
   player: 'M.S. Dhoni ';
   sport:'Cricket';

capital: string;
constructor() {
   this.capital = 'New Delhi';
}

fruits = ['Apple', 'Orange', 'Mango', 'Grapes'];
   myfruit = this.fruits[1];
}
```

- The ***@Component*** is a decorator that uses the configuration object to create the component.

- The ***selector*** creates an instance of the component where it finds the **<my-app>** tag in parent HTML.
- The ***@view*** contains a ***template*** that tells Angular how to render a view.
- The ***export*** specifies that the component will be available outside the file.

Output

Let us now proceed with the following steps to see how the above code works:

- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above app folder which contains **.ts** files.

- Open the terminal window and enter the command as given below:

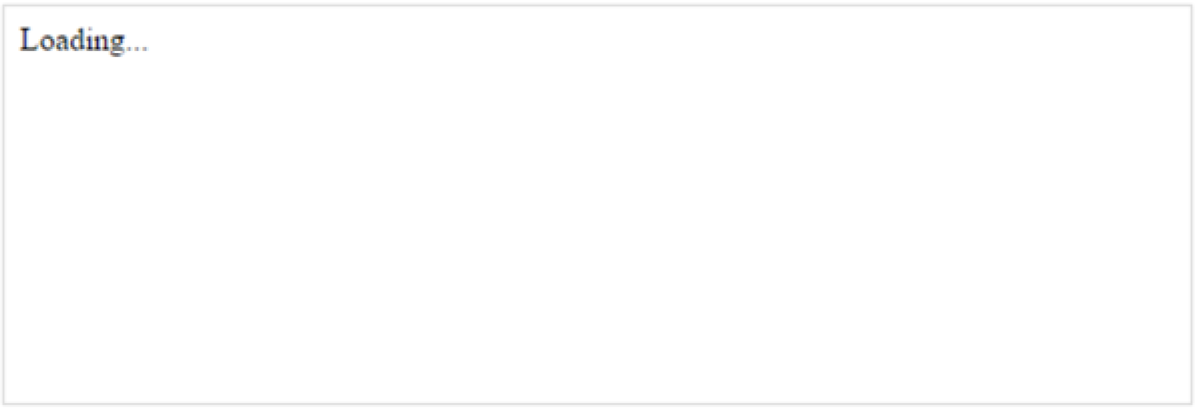
```
npm start
```

- After a few moments, a browser tab should open. You will receive an output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **angular2_data_display.html** file in your server root folder.

- Open this HTML file as **http://localhost/angular2_data_display.html.** You will receive the following output.



Loading...

11. Angular 2 — User Input

When the user clicks a button, enters text or clicks a link, these user interactions will trigger the DOM events. The following table shows how to bind to these events using the Angular event binding syntax.

Sr. No.	Event & Description
1	**Binding to User Input Events** You can input the text or display the text value when you click on it by using the Angular event binding.
2	**User Input from $event Object** You can display the input value by binding **keyup** event. This displays the text that you type onto the screen.
3	**User Input from Local Template Variable** You can display the user data by using the local template variable.
4	**Key Event Filtering** You can display the data of input box by pressing the 'Enter' key on the keyboard.
5	**On Blur Event** You can make the input value blur by clicking the mouse outside of the input box on the page.

Angular 2 - Binding User Input

Description

You can input the text or display the text value when you click on it by using the Angular event binding syntax.

Example

The following example shows the use of binding to user input in Angular 2:

```
<!DOCTYPE html>
<html>
  <head>
```

```
    <title>Angular 2 User Input</title>
    <script    src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
    <script              src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
    <script                                src="https://code.angularjs.org/2.0.0-
beta.6/angular2.dev.js"></script>
    <script>
      System.config({
        transpiler: 'typescript',
        typescriptOptions: { emitDecoratorMetadata: true },
        packages: {'app': {defaultExtension: 'ts'}},
        map: { 'app': './angular2/src/app' }
      });
      System.import('app/user_input')
          .then(null, console.error.bind(console));
    </script>
  </head>
<body>
   <my-app>Loading...</my-app>
</body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors which are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadat****a* option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.

- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in main.ts, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files. These files need to be saved under the ***app*** folder.

user_input.ts

```
import {bootstrap} from 'angular2/platform/browser';
import {AppComponent} from "./app.component";


bootstrap(AppComponent);
```

We will now create a component in the **TypeScript(.ts)** file as shown below:

app.component.ts

```
import {Component} from 'angular2/core';
import {ItemListComponent} from './shopping-list.component';


@Component({
    selector: 'my-app',
    template: `
    <my-list></my-list>


    `,
    directives:[ItemListComponent]


})
export class AppComponent {}
```

- The ***@Component*** is a decorator that uses configuration object to create the component.
- The ***selector*** creates an instance of the component where it finds the **<my-app>** tag in parent HTML.
- The above ***app.component*** will import the ***ItemListComponent*** component and uses ***directives*** to include the component.

shopping-list.component.ts

```
import {Component} from "angular2/core";
@Component({
   selector:'my-list',
   template:`
      <ul>
         <li *ngFor="#listItem of listItems"
            (click)="onItemClicked(listItem)">{{listItem.name}}
         </li>
      </ul>
      <input type="text" [(ngModel)]="selectedItem.name">
      <button (click)="onDeleteItem()">Delete Item</button><br><br>
      <input type="text" #listItem>
      <button (click)="onAddItem(listItem)">Add Item</button>
   `
})

export class ItemListComponent{
   public listItems = [
      {name:"apple"},
      {name:"orange"},
      {name:"grapes"},
   ];
   public selectedItem = {name: ""};

   onItemClicked(listItem){
      this.selectedItem=listItem;
   }
   onAddItem(listItem){
      this.listItems.push({name:listItem.value});
   }

   onDeleteItem(){
      this.listItems.splice(this.listItems.indexOf(this.selectedItem),1);
   }
}
```

- The ***template*** tells Angular how to display the component.
- The ****ngFor*** directive is used to loop the list of items from the array of ***listItems*** object.
- The ***shopping-list.component*** component uses ***(click)*** for binding events.
- To add an item, enter an item and click on **Add Item** button and to delete an item, click on the item and hit **Delete Item** button.

Output

Let us now proceed with the following steps to see how the above code works:

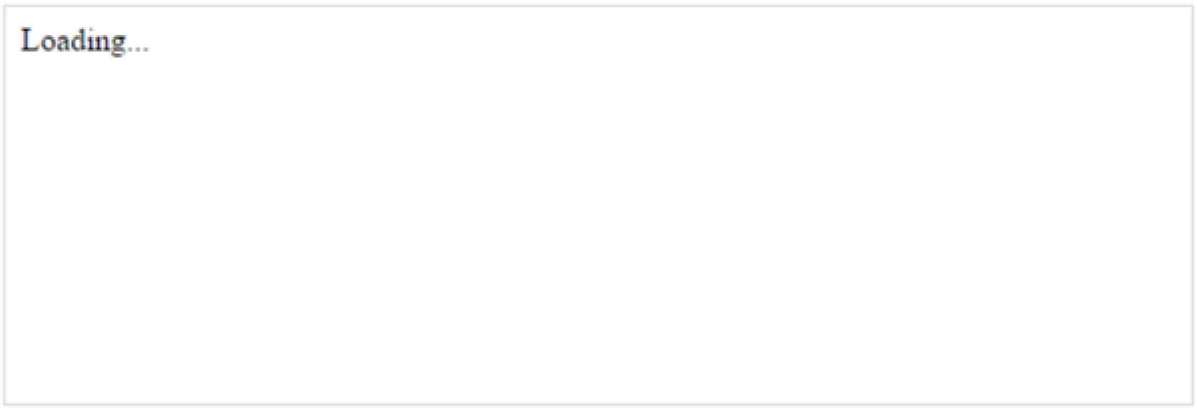
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above ***app*** folder which contains ***.ts*** files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **binding_user_input.html** file in your server root folder.
- Open this HTML file as **http://localhost/binding_user_input.html**. You will receive the following output.



```
Loading...
```

Click on any item and hit **Delete Item** to delete an item from the list and enter an item and click **Add item** to add an item.

Angular 2 - User Input from Event Object

Description

You can display the input value by binding the key event. This displays the text that the user types onto the screen.

Example

The following example describes the user input from the event object in Angular 2:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Angular 2 User Input Keyup Event</title>



    <script    src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
    <script              src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
    <script                              src="https://code.angularjs.org/2.0.0-
beta.6/angular2.dev.js"></script>
    <script>

      System.config({
        transpiler: 'typescript',
        typescriptOptions: { emitDecoratorMetadata: true },
        packages: {'app': {defaultExtension: 'ts'}},
        map: { 'app': './angular2/src/app' }
      });
      System.import('app/user_input_keyup')
          .then(null, console.error.bind(console));
    </script>
  </head>
<body>

   <my-key>Loading...</my-key>
</body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors that are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

Let's create the ***TypeScript(.ts)*** files and save them in the ***app*** folder.

user_input_keyup.ts

```
import {bootstrap} from 'angular2/platform/browser';
import {KeyUpComponent} from "./key_up.component";

bootstrap(KeyUpComponent);
```

We will now create a component in the **TypeScript(.ts)** file as shown below:

key_up.component.ts

```
import {Component} from 'angular2/core';
@Component({
  selector: 'my-key',
  template: `<h2>Key Up Event Example</h2>
    <input (keyup)="onKey($event)">
    <p>{{val}}</p>
  `
})
export class KeyUpComponent {
  val='';
  onKey(event:KeyboardEvent) {
    this.val += (event.target).value + ' | ';


  }
}
```

- The ***@Component*** is a decorator that uses the configuration object to create the component and its view.
- The *selector* creates an instance of the component where it finds the **<my-key>** tag in parent HTML.
- Angular makes an event object available in variable ***$event*** and it is passed to the ***onKey()*** method.
- The ***onKey()*** component method will extract the user's input from the event object and adds it to the list of the user data.

Output

Let us now proceed with the following steps to see how the above code works:

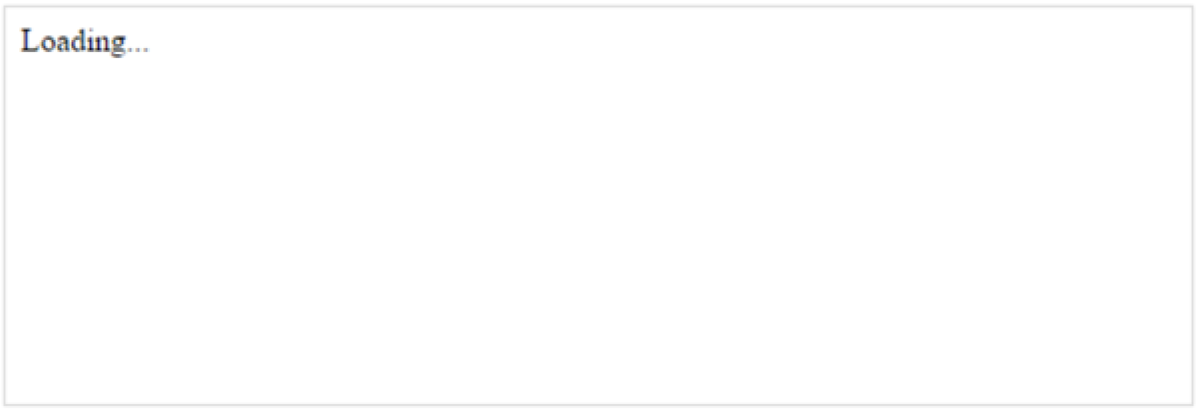
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above ***app*** folder which contains ***.ts*** files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **user_input_event_object.html** file in your server root folder.
- Open this HTML file as **http://localhost/user_input_event_object.html** and the output as below gets displayed.



```
Loading...
```

Enter any text in the above input box. Each time you enter a letter, it is added to the previous text and will be displayed as separated text. If you hit backspace, the last letter will be deleted and the remaining letters will be displayed as text.

Angular 2 - User Input from Local Template Variable

Description

You can display the user data by using the local template variable and this variable can be defined by using identifier with the **hash(#)** character.

Example

The following example describes the user input from the local template variable in Angular 2:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Angular 2 User Input from Local Template Variable</title>
    <script    src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>
    <script              src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
    <script                              src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>


    <script>
      System.config({
        transpiler: 'typescript',
        typescriptOptions: { emitDecoratorMetadata: true },
        packages: {'app': {defaultExtension: 'ts'}},
        map: { 'app': './angular2/src/app' }
      });
      System.import('app/user_input_loop_back')
         .then(null, console.error.bind(console));
    </script>
  </head>
<body>
   <loop-back-event>Loading...</loop-back-event>
</body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors that are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files which you need to save under the ***app*** folder.

user_input_loop_back.ts

```
import {bootstrap} from 'angular2/platform/browser';
import {LoopBackEventComponent} from "./loop_back.component";

bootstrap(LoopBackEventComponent);
```

We will now create a component in the **TypeScript(.ts)** file as shown below:

loop_back.component.ts

```
import {Component} from 'angular2/core';
@Component({
  selector: 'loop-back-event',
  template:`
```

```
    <h2>Get user input from a local template variable</h2>


    <!--declare a local template variable by preceding an identifier with a hash
character (#)-->

    <input #key_val (keyup)="0">


    <!-- The key_val variable is a reference to the <input> element itself, and
grab the input element's value and display it with interpolation between <p> tags
-->

    <p>{{key_val.value}}</p>
  `

})

export class LoopBackEventComponent { }
```

- The ***@Component*** is a decorator that uses the configuration object to create the component and its view.
- The ***selector*** creates an instance of the component where it finds the **<loop-back-event>** tag in parent HTML.
- The ***key_val*** is the template reference variable on the ***<input>*** element.
- The ***key_val*** variable grabs the input element's value and is displayed with interpolation placed between the **<p>** tags.

Output

Let us now proceed with the following steps to see how the above code works:

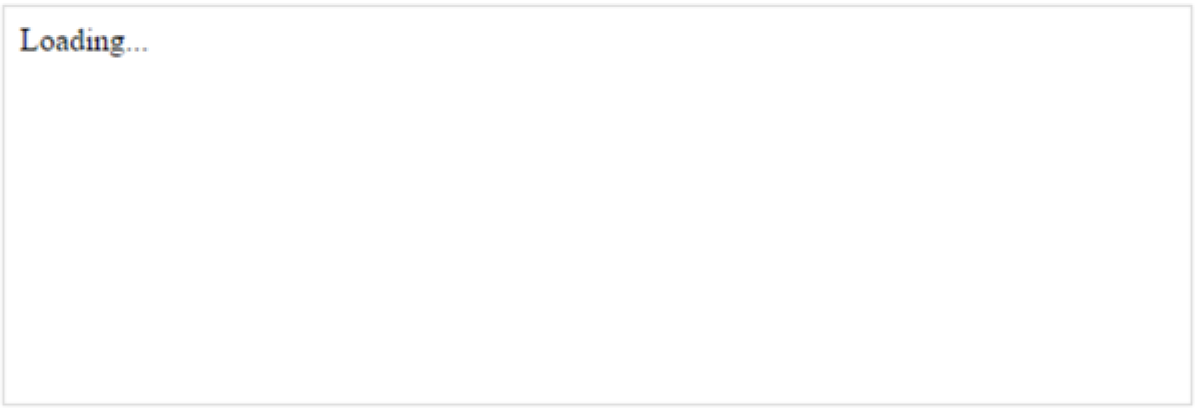
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above ***app*** folder which contains ***.ts*** files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **user_input_template_var.html** file in your server root folder.
- Open this HTML file as **http://localhost/user_input_template_var.html**. You will receive the following output.


Loading...

Angular 2 - Key Event Filtering

Description

The user can display the data of input box by pressing the **'Enter'** key on the keyboard.

Example

The following example shows the user input by using the key event filtering in Angular 2:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Angular 2 User Input Key Event Filtering</title>
      <script  src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>
      <script           src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>
      <script src="https://code.angularjs.org/tools/system.js"></script>
      <script src="https://code.angularjs.org/tools/typescript.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>
      <script>
         System.config({
           transpiler: 'typescript',
           typescriptOptions: { emitDecoratorMetadata: true },
           packages: {'app': {defaultExtension: 'ts'}},
           map: { 'app': './angular2/src/app' }
```

```
          });
          System.import('app/user_input_event_filtering')
             .then(null, console.error.bind(console));
       </script>
    </head>
    <body>
       <event-filtering>Loading...</event-filtering>

    </body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors that are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the *app* folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files. These files need to be saved under the ***app*** folder.

user_input_event_filtering.ts

```
import {bootstrap} from 'angular2/platform/browser';
import {EventFilteringComponent} from "./event_filtering.component";

bootstrap(EventFilteringComponent);
```

We will now create a component in the **TypeScript(.ts)** file as shown below:

event_filtering.component.ts

```
import {Component} from 'angular2/core';

@Component({
  selector: 'event-filtering',
  template: `
    <input #myval (keyup.enter)="values=myval.value">
    <p>{{values}}</p>
  `
})
export class EventFilteringComponent {
  values='';
}
```

- The ***@Component*** is a decorator that uses the configuration object to create the component and its view.
- When the user presses the **'Enter'** key on the keyboard, Angular 2 calls the ***keyup*** event. This displays the text entered by the user.

Output

Let us now proceed with the following steps to see how the above code works:

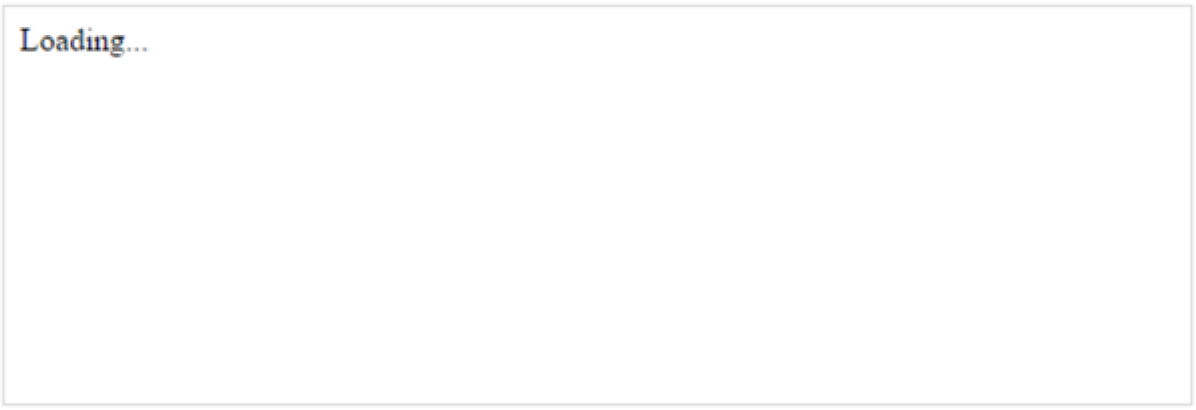
- Save above HTML code as **index.html** file the way we created in the environment chapter and use the above ***app*** folder which contains ***.ts*** files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **user_input_key_event_filtering.html** file in your server root folder.
- Open this HTML file as **http://localhost/user_input_key_event_filtering.html** and the output as below gets displayed.


Loading...

The example displays the data by pressing **'Enter'** key on the keyboard when you enter the data in the input box.

Angular 2 - On Blur Event

Description

You can make the input value blur by clicking the mouse outside of the input box on the page.

Example

The following example describes the user input on the **blur** event in Angular 2:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Angular 2 User Input On Blur</title>
      <script  src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>


      <script src="https://code.angularjs.org/tools/system.js"></script>
      <script src="https://code.angularjs.org/tools/typescript.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>
      <script>
         System.config({
           transpiler: 'typescript',
           typescriptOptions: { emitDecoratorMetadata: true },
           packages: {'app': {defaultExtension: 'ts'}},
           map: { 'app': './angular2/src/app' }
         });
         System.import('app/user_input_onblur_event')
            .then(null, console.error.bind(console));
      </script>
   </head>
   <body>
      <onblur-event>Loading...</onblur-event>
   </body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors that are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files. These files need to be saved under the *app* folder.

user_input_onblur_event.ts

```
import {bootstrap} from 'angular2/platform/browser';
import {OnblurEventComponent} from "./user_onblur_event.component";

bootstrap(OnblurEventComponent);
```

We will now create a component in the **TypeScript(.ts)** file as shown below:

user_onblur_event.component.ts

```
import {Component} from 'angular2/core';

@Component({
  selector: 'onblur-event',
  template: `<h2>User Input On Blur Event</h2>
    <input #myval
      (keyup.enter)="values=myval.value"
      (blur)="values=myval.value">
    <p>{{values}}</p>
  `
})
export class OnblurEventComponent {
  values='';
}
```

- The *@Component* is a decorator that uses configuration object to create the component and its view.
- When the user leaves a form field, Angular 2 calls the ***keyup*** event and the ***onblur*** event makes the input box blur.

Output

Let us proceed with the following steps to see how the above code works:

- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above app folder which contains the **.ts** files.
- Open the terminal window and enter the command as given below:

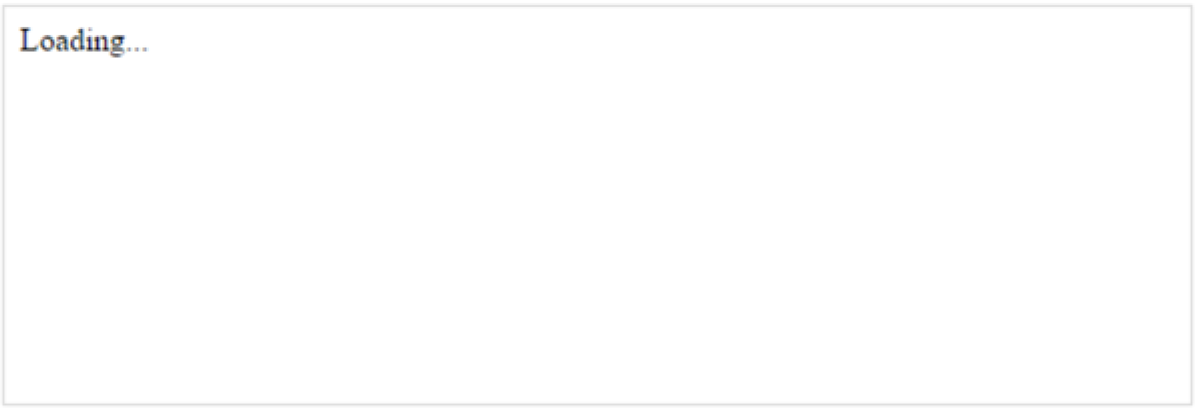
```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **user_input_onblur.html** file in your server root folder.

- Open this HTML file as **http://localhost/user_input_onblur.html**. You will receive the following output.



```
Loading...
```

The example displays the functionality of the ***onblur event when the user leaves a form field*** and makes the input box blur.

12. Angular 2 — Forms

In this chapter, let us study how to create a *form*. We will make use of the following classes and directives in our example.

- The ***form-group***, ***form-control*** and ***btn*** classes form ***Bootstrap***.
- The ***[(ngModel)]*** for data binding and ***NgControl*** directive to keep track of control state for us.
- The ***NgControl*** is one among many in the ***NgForm*** directive family which is used for validation and tracking of the form elements.
- The ***ngSubmit*** directive is used for handling the submission of the form.

Example

The following example describes how to create a form in Angular 2:

```
<html>
  <head>
    <title>Contact Form</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
    <script    src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
    <script              src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
    <script src="https://code.angularjs.org/2.0.0-
beta.6/angular2.dev.js"></script>
    <script>
      System.config({
        transpiler: 'typescript',
        typescriptOptions: { emitDecoratorMetadata: true },
        packages: {'app': {defaultExtension: 'ts'}},
        map: { 'app': './angular2/src/app' }
      });
      System.import('app/form_main')
            .then(null, console.error.bind(console));
    </script>

  </head>
  <body>
    <my-app>Loading...</my-app>
  </body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors that are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files. These files need to be saved under the ***app*** folder.

form_main.ts

```
import {bootstrap} from 'angular2/platform/browser';
import {AppComponent} from "./data_binding_app.component";

bootstrap(AppComponent);
```

contact.ts

```
export class Contact {
  constructor(
    public firstname: string,
    public lastname: string,
    public country: string,
    public phone: number
  ) {  }
}
```

The ***Contact*** class contains ***firstname***, ***lastname***, ***country*** and ***phone*** that are used in our form.

forms_app.component.ts

```
import {Component} from 'angular2/core';
import {ContactComponent} from './contact-form.component'
@Component({
  selector: 'my-app',
  template: '',
  directives: [ContactComponent]
})
export class AppComponent { }
```

- The ***@Component*** is a decorator that uses the configuration object to create the component.
- The ***selector*** creates an instance of the component where it finds the **<my-app>** tag in parent HTML.
- The ***template*** tells Angular what to render as view.
- The above ***app.component.ts*** will import the ***ContactComponent*** component. The component also uses ***directives*** to include the component.

contact-form.component.ts

```
import {Component} from 'angular2/core';
import {NgForm}    from 'angular2/common';
import { Contact } from './contact';
@Component({
  selector: 'contact-form',
  templateUrl: 'app/contact-form.component.html'
})
export class ContactComponent {
  countries = ['India', 'Australia', 'England', 'South Africa', 'USA', 'Japan',
'Singapore'];
  contact = new Contact('Ravi', 'Shankar', this.countries[0], 6445874544);
  submitted = false;
  onSubmit() { this.submitted = true; }
  active = true;
  newContact() {
    this.contact = new Contact('', '');
    this.active = false;
    setTimeout(()=> this.active=true, 0);
  }
}
```

- The **NgForm** is imported. This provides ***CSS classes*** and *Model states*.
- The ***templateUrl*** property provides the path to the ***contact-form.component.html*** file. This path contains our form elements.
- The ***onSubmit()*** method will alter the ***submitted*** value to ***true*** once invoked and the ***newContact**()* will create a new contact.

contact-form.component.html

```
<div class="container">
  <div [hidden]="submitted">
    <h2>Contact Form</h2>
    <form *ngIf="active" (ngSubmit)="onSubmit()" #contactForm="ngForm" novalidate>

      <div class="form-group">
        <label for="firstname">First Name</label>
        <input type="text" class="form-control" placeholder="Enter Your First
Name" required
          [(ngModel)]="contact.firstname"
            ngControl="firstname"  #firstname="ngForm" >

        <div [hidden]="firstname.valid || firstname.pristine" class="alert
alert-danger">
          firstname is required
        </div>
      </div>
      <div class="form-group">
        <label for="lastname">Last Name</label>
```

```
         <input type="text" class="form-control" placeholder="Enter Your Last Name"
           [(ngModel)]="contact.lastname"
             ngControl="lastname" >
       </div>
       <div class="form-group">
         <label for="country">Country</label>
         <select class="form-control" required
           [(ngModel)]="contact.country"
             ngControl="country" #country="ngForm" >
           <option value="" selected disabled>Select Your Country</option>
           <option *ngFor="#coun of countries" [value]="coun">{{coun}}</option>
         </select>
         <div [hidden]="country.valid || country.pristine" class="alert alert-
danger">
           Country is required
         </div>
       </div>

       <div class="form-group">
          <label for="phone">Phone Number</label>

          <input type="number" class="form-control" placeholder="Enter Your
Phone Number"
             [(ngModel)]="contact.phone"
             ngControl="phone"
          >
       </div>

       <button type="submit" class="btn btn-primary"
 [disabled]="!contactForm.form.valid">Submit</button>
       <button type="button" class="btn btn-primary" (click)="newContact()">New
Contact</button>
     </form>
   </div>
   <div [hidden]="!submitted">
     <h2>Your contact details :</h2>
     <div class="well">
         <div class="row">
```

```
          <div class="col-xs-2">First Name</div>
          <div class="col-xs-10  pull-left">{{ contact.firstname }}</div>
        </div>
        <div class="row">
          <div class="col-xs-2">Last Name</div>
          <div class="col-xs-10 pull-left">{{ contact.lastname }}</div>

        </div>
        <div class="row">
          <div class="col-xs-2">Country</div>
          <div class="col-xs-10 pull-left">{{ contact.country }}</div>
        </div>
        <div class="row">
        <div class="col-xs-2">Phone Number</div>
        <div class="col-xs-10 pull-left">{{ contact.phone }}</div>
      </div>
        <br>
        <button class="btn btn-primary" (click)="submitted=false">Edit Contact</button>
    </div>
  </div>
</div>
```

- The above code contains the form, on submitting the form the ***ngSubmit*** directive calls the ***onSubmit()*** method.
- The contact details are displayed when the ***submitted*** is set to ***false***.
- It uses ***pristine*** and ***valid*** for validating a form input element.
- The ***ngIf*** directive checks whether the ***active*** is set to ***true*** for displaying the form.

Output

Let us now proceed with the following steps to see how the above code works:

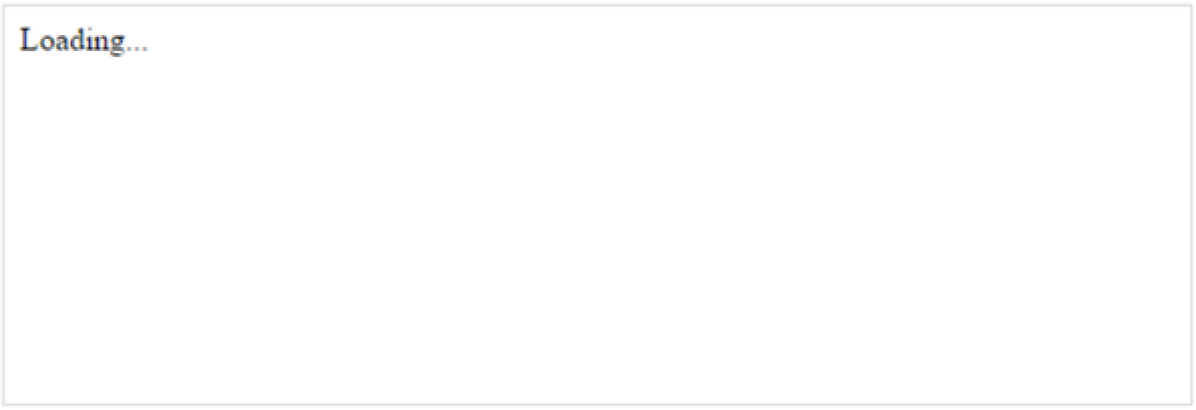
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above ***app*** folder which contains the ***.ts*** files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **angular2_forms.html** file in your server root folder.
- Open this HTML file as **http://localhost/angular2_forms.html**. You will receive the following output.



```
Loading...
```

13. Angular 2 — Services

Services are JavaScript functions that are responsible for performing a specific task only. Angular services are injected using the **Dependency Injection** mechanism and include the **value**, **function**, or **feature** required by the application. There is no ServiceBase class in Angular, but still services can be treated as fundamental to Angular application.

Example

The following example shows the use of services in Angular 2:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Angular 2 Services</title>
    <!--Load libraries -->
    <script     src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-shim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-polyfills.js"></script>
    <script              src="https://code.angularjs.org/2.0.0-beta.6/angular2-polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>
    <script>
      System.config({
        transpiler: 'typescript',

        typescriptOptions: { emitDecoratorMetadata: true },
        packages: {'app': {defaultExtension: 'ts'}},

        map: { 'app': './angular2/src/app' }
      });

      System.import('app/service_main')
            .then(null, console.error.bind(console));
```

```
     </script>
   </head>
<body>
    <my-app>Loading...</my-app>
</body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors that are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files which you need to save under the ***app*** folder.

metadata_main.ts

```
import {bootstrap} from 'angular2/platform/browser';      //importing bootstrap
function
import {AppComponent} from "./app_service.component";     //importing component
function


bootstrap(AppComponent);
```

We will now create a component in **TypeScript(.ts)** file. This will create a view for the component.

app_service.component.ts

```
import {Component} from 'angular2/core';
import {MyListComponent} from "./service-list.component";

@Component({
    selector: 'my-app',
    template: `
    <country-list></country-list>
    `,
    directives: [MyListComponent]
})
export class AppComponent {
}
```

- The ***@Component*** is a decorator that uses configuration object to create the component and its view.
- The ***selector*** creates an instance of the component where it finds the **<my-app>** tag in parent HTML.
- We will also create a directive called ***MyListComponent*** which will be accessed from the ***service-list.component*** file.

service-list.component.ts

```
import {Component} from "angular2/core";
import {CountryService} from "./country.service";
import {Contact} from "./country";
import {OnInit} from "angular2/core";

@Component({
   selector: "country-list",
   template: ` List of Countries<br>
   <ul>
      <li *ngFor="#cntry of countries">{{ cntry.name }}</li>
   </ul>
   `,
   providers: [CountryService]
})

export class MyListComponent implements OnInit {
   public countries : Country[];
   constructor(private _countryService: CountryService) {}


   getContacts(){
      this._countryService.getContacts().then((countries: Country[]) =>
this.countries = countries);
   }


ngOnInit():any{
   this.getContacts();
}
}
```

- The local variable ***cntry*** can be referenced in the template. This will help get the index of the array. Angular 2 will bind the model name from the array with the local variable of the template.
- We have a resource called ***providers***. This registers the **class**, **function** or **value** that exist in the context of dependency injection. The service called ***CountryService*** can be injected using **@Injectable()** in the ***country.service.ts*** file.
- Next you have to implement the ***MyListComponent*** class using ***OnInit*** hook which indicates that Angular is created by that component. Using the constructor, call the ***_countryService*** and populate the ***countries*** list.
- The ***ngOnInit()*** hook is called when the component is created and its inputs are evaluated.

country.service.ts

```
import {Injectable} from "angular2/core";
import {COUNTRIES} from "./country.contacts";


//@Injectable() specifies class is available to an injector for instantiation
and an injector will display an error when trying to instantiate a class that is
not marked as @Injectable()


@Injectable()
//CountryService exposes the getContacts() method that returns the data
export class CountryService {
   getContacts() {
      return Promise.resolve(COUNTRIES); // takes values from country.contacts
typescript file
   }
}
```

country.contacts.ts

```
import {Country} from "./country";


//storing array of data in Country
export const COUNTRIES: Country[] =[
   {name :"India"},
   {name: "Srilanka"},
   {name: "South Africa"},
   {name: "New Zealand"}
];
```

country.ts

```
export interface Country{
   name: string
}
```

Output

Let us now proceed with the following steps to see how the above code works:

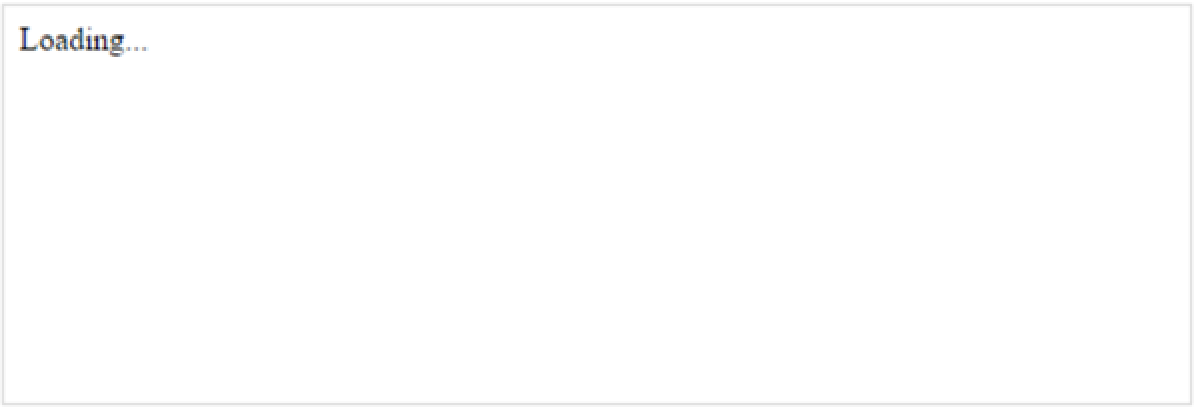
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above ***app*** folder which contains ***.ts*** files.
- Open the terminal window and enter the following command:

```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **angular2_services.html** file in your server root folder.
- Open this HTML file as **http://localhost/angular2_services.html**. You will receive the following output.



```
Loading...
```

14. Angular 2 — Directives

In this chapter, let us discuss **Directives in Angular 2**. Templates of the Angular are ***dynamic***; when these templates are rendered by Angular, it changes the DOM according to the directive fed instructions. The directive is a class that contains metadata, which will be attached to the class by the ***@Directive*** decorator.

Angular has three kinds of directives and is briefly explained in the following table:

Sr. No.	Directives & Description
1	**Component** It is a *directive-with-a-template* and the *@Component* decorator that is indeed a *@Directive* decorator wherein the template-oriented features is extended.
2	**Structural directives** It alters the layout of the DOM by adding, replacing, and removing its elements.
3	**Attribute directives** It changes the appearance or behavior of a DOM element. These directives look like regular HTML attributes in templates.

Angular 2 - Components

Description

The component is a controller class with a template which mainly deals with a view of the application and the logic on the page. It is a bit of code can be used throughout an application. The component knows how to render itself and configure dependency injection.

The component contains two important things; one is ***view*** and the other one is ***some logic***.

Example

The following example shows the use of component in Angular 2:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Angular 2 Component</title>
      <!--Load libraries -->
      <script  src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
      <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
      <script src="https://code.angularjs.org/tools/system.js"></script>
      <script src="https://code.angularjs.org/tools/typescript.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
      <script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>
      <script>
         System.config({
            //transpiler tool converts TypeScript to JavaScript
            transpiler: 'typescript',

            //emitDecoratorMetadata flag used by JavaScript output to create
metadata from the decorators
            typescriptOptions: { emitDecoratorMetadata: true },
            packages: {'app': {defaultExtension: 'ts'}},
            map: { 'app': './angular2/src/app' }
         });
         System.import('app/component_main')
            .then(null, console.error.bind(console));
      </script>
   </head>
   <!--When Angular calls the bootstrap function in main.ts, it reads the
Component metadata, finds the 'app' selector, locates an element tag named app,
and loads the application between those tags.-->
   <body>
      <app>Loading...</app>
   </body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors which are hidden in the browser.

- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files which you need to save under the ***app*** folder.

component_main.ts

```
import {bootstrap} from "angular2/platform/browser";   //importing bootstrap function
import {App} from "./component_app.component"          //importing component function


bootstrap(App);
```

We will now create a component in the **TypeScript(.ts)** file. This will create a component and also a view for the component.

component_app.component.ts

```
// component's metadata can be accessed using this primary Angular library
import {Component, View} from "angular2/core";


//framework recognizes @Component annotation and knows that we are trying to create a new component
@Component({
   selector: 'app'  //specifies selector for HTML element named 'app'
})


@View({
  //template property holds component's companion template that tells Angular how to render a view
  template: '<h2>Welcome to {{name}}</h2>'
})

```

```
export class App {
   name : 'Tutorialspoint!!!'
}
```

Output

When you run the above code, it will display the text specified within the ***template*** option which is defined in the ***component_app.component.ts*** file and holds the component's companion template that tells Angular how to render a view.

Let us now proceed with the following steps to see how the above code works:

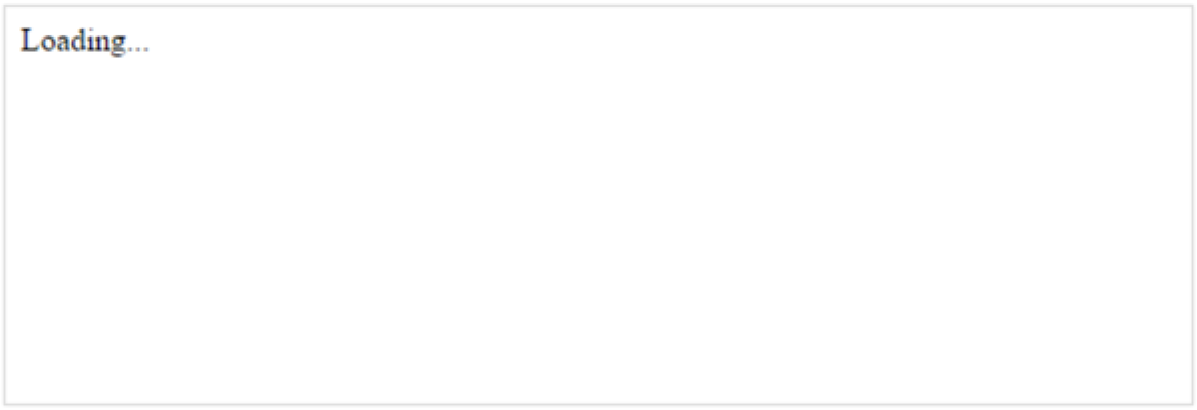
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above ***app*** folder which contains ***.ts*** files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **angular2_components.html** file in your server root folder.
- Open this HTML file as **http://localhost/angular2_components.html**. You will receive the following output.



Loading...

Angular 2 - Structural Directives

Description

The ***structural directives*** alter the layout of the DOM by **adding**, **replacing** and **removing** its elements. The two familiar examples of structural directive are listed below:

- **NgFor:** It is a repeater directive that customizes data display. It can be used to display a list of items.

- **NgIf:** It removes or recreates a part of DOM tree depending on an expression evaluation.

Example

The following example shows the use of **structural directives** in Angular 2:

```
<html>
  <head>
    <title>Contact Form</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">



    <script src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>
    <script>
      System.config({
        transpiler: 'typescript',
        typescriptOptions: { emitDecoratorMetadata: true },
        packages: {'app': {defaultExtension: 'ts'}},
        map: { 'app': './angular2/src/app' }
      });
      System.import('app/structural_main')
            .then(null, console.error.bind(console));
    </script>

  </head>
  <body>
    <my-app>Loading...</my-app>
  </body>
```

```
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the ***typescript*** version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors that are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated. This affects the file size and also has an impact on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files which you need to save under the ***app*** folder.

structural_main.ts

```
import {bootstrap} from 'angular2/platform/browser';       //importing bootstrap
function

import {AppComponent} from './structural_app.component';   //importing component
function

bootstrap(AppComponent);
```

We will now create a component in the **TypeScript(.ts)** file. This will create a view for the component.

structural_app.component.ts

```
import {Component} from 'angular2/core';


@Component({

  selector: 'my-app',

  template: `

    <h2>{{title}}</h2>

    <p class="alert alert-success" *ngIf="names.length > 2">Currently there are
more than 2 names!</p>
```

```
    <p class="alert alert-danger" *ngIf="names.length <= 2">Currently there are
less than 2 names left!</p>
    <ul>
         <li *ngFor="#nam of names"
           (click)="onNameClicked(nam)"
           >{{ nam.name }}</li>
        </ul>
        <input type="text" [(ngModel)]="selectedName.name">
        <button (click)="onDeleteName()">Delete Name</button><br><br>
        <input type="text" #nam>
        <button (click)="onAddName(nam)">Add Name</button>
  `
})

export class AppComponent {
  title = 'Structural Directives';

  public names = [
    { name: "Kamal"},
    { name: "Mitchel"},
    { name: "Yoon"},
    { name: "Johnson"},
    { name: "Jet Li"}
  ];
  public selectedName = {name : ""};
  onNameClicked(nam) {
    this.selectedName = nam;
  }
  onAddName(nam) {
    this.names.push({name: nam.value});
  }
  onDeleteName() {
    this.names.splice(this.names.indexOf(this.selectedName), 1);
      this.selectedName.name = "";
  }
}
```

- The ***@Component*** is a decorator that uses the configuration object to create the component and its view.
- The ***selector*** creates an instance of the component where it finds the **<my-app>** tag in parent HTML.
- Further, the ***ngFor*** directive creates the view exports which we bind to, in the template. The ***** is a shorthand for using Angular 2 template syntax with the template tag.
- The local variable ***nam*** can be referenced in the template and get the index of the array. When you click on the item value, the ***onNameClicked()*** event will get activated and Angular 2 will bind the model name from the array with the local variable of template.
- The methods ***onAddName()*** and ***onDeleteName()*** are used to add and delete the items from the list. The ***onNameClicked()*** method uses the local variable **'nam'** as parameter and displays the selected item by using the ***selectedName*** object.

Output

Let us now proceed with the following steps to see how the above code works:

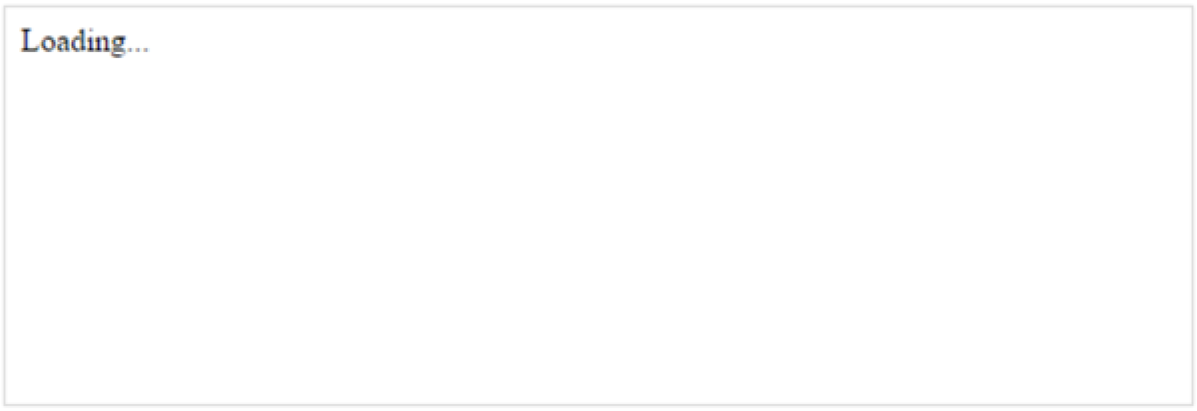
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above *app* folder which contains *.ts* files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **structural_directives.html** file in your server root folder.
- Open this HTML file as **http://localhost/structural_directives.html**. You will receive the following output.



Loading...

Angular 2 - Attribute Directives

Description

The attribute directive changes the appearance or behavior of a DOM element. These directives look like regular HTML attributes in templates. The ***ngModel*** directive which is used for two-way binding is an example of an attribute directive. Some of the other attribute directives are listed below:

- **NgSwitch:** It is used whenever you want to display an element tree consisting of many children. The Angular places only selected element tree into the DOM based on some condition.
- **NgStyle:** Based on the component state, dynamic styles can be set by using ***NgStyle***. Many inline styles can be set simultaneously by binding to ***NgStyle***.
- **NgClass:** It controls the appearance of elements by adding and removing CSS classes dynamically.

Example

The following example shows the use of attribute directives in Angular 2:

```
<html>
  <head>
    <title>Contact Form</title>

    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet"
 href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">

    <script     src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
    <script               src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
    <script                                src="https://code.angularjs.org/2.0.0-
beta.6/angular2.dev.js"></script>
    <script>
      System.config({
        transpiler: 'typescript',
        typescriptOptions: { emitDecoratorMetadata: true },
        packages: {'app': {defaultExtension: 'ts'}},
        map: { 'app': './angular2/src/app' }
      });
      System.import('app/attribute_main')
            .then(null, console.error.bind(console));
    </script>
  </head>
  <body>
    <my-app>Loading...</my-app>
  </body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the *typescript* version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors that are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impacts on the application runtime.
- Angular 2 includes the packages from the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

To run the code, you need the following ***TypeScript(.ts)*** files. These files need to be saved under the ***app*** folder.

attribute_main.ts

```
import {bootstrap} from 'angular2/platform/browser';      //importing bootstrap
function
import {AppComponent} from "./attribute_app.component";   //importing component
function
```

```
bootstrap(AppComponent);
```

We will now create a component in the **TypeScript(.ts)** file. This will create a view for the component.

```
attribute_app.component.ts
import {Component} from 'angular2/core';
import {ShoppingListComponent} from './shopping-item.component';

@Component({
    selector: 'my-app',
    template: `
    <shopping></shopping>
    `,
    directives: [ShoppingListComponent]
})
export class AppComponent {

}
```

- The ***@Component*** is a decorator that uses the configuration object to create the component and its view.
- The ***selector*** creates an instance of the component where it finds the **<my-app>** tag in parent HTML.
- Next, we create a directive called ***ShoppingListComponent*** which will be accessed from the ***shopping-item.component*** file.

shopping-item.component.ts

```
import {Component} from "angular2/core";
import {NgSwitch} from "angular2/core";
import {NgSwitchWhen} from "angular2/core";
import {NgSwitchDefault} from "angular2/core";
import {NgClass} from 'angular2/common';

@Component({
   selector: "shopping",
   template: `
```

```
        <h2>Shopping Items</h2>
        <ul>

           <li
           *ngFor="#shopItem of shopItems"
           (click)="onItemClicked(shopItem)"
           >{{ shopItem.name }}</li>
        </ul>

        <span [ngSwitch]=selectedItem.name>
          <p>You selected :
             <span *ngSwitchWhen="'Shirt'">Shirt</span>
             <span *ngSwitchWhen="'Pant'">Pant</span>

             <span *ngSwitchWhen="'Sarees'">Sarees</span>
             <span *ngSwitchWhen="'Jeans'">Jeans</span>
             <span *ngSwitchWhen="'T-Shirt'">T-Shirt</span>
             <span *ngSwitchDefault>Nothing</span>
          </p>
        </span>

        <div [ngStyle]="setStyles(selectedItem.name)" class="text-success">
           Thank you for Selecting an item!!
        </div>

        <button [ngClass]="{active: isActive}" (click)="isActive = !isActive">Buy
Items</button>
    `,
    styles: [`
        .button {
           width: 120px;
           border: medium solid black;
        }
        .active {
           background-color: red;
        }
        p{
            font-weight: bold;
         }
    `]
    directives: [NgClass]
})

export class ShoppingListComponent {
    public shopItems = [
       {name: "Shirt"},
       {name: "Pant"},
       {name: "Sarees"},
       {name: "Jeans"},
       {name: "T-Shirt"},
    ];
    public selectedItem = {name: ""};
    onItemClicked(shopItem) {
       this.selectedItem = shopItem;
    }
    setStyles(item) {
       let styles = {
         'font-size'  : item? '24px'   : 'none',
         'visibility' : !item? 'hidden' : 'visible'
       }
       return styles;
    }
}
```

- The local variable ***shopItem*** can be referenced in the template. This will help get the index of the array. Angular 2 will bind the model name from the array with the local variable of template.

- When you click on the item value, the ***onItemClicked()*** event will get activated and Angular 2 will bind the model name from the array with the local variable of the template.

- The ***NgSwitch*** directive inserts the nested elements based on which match expression matches the value obtained from the evaluated switch expression. It displays an element whose ***ngSwitchWhen*** matches the current ***ngSwitch*** expression value.

- The ***onItemClicked()*** method uses the local varaible **'shopItem'** as parameter and displays the selected item by using the ***selectedItem*** object.

Output

Let us now proceed with the following steps to see how the above code works:

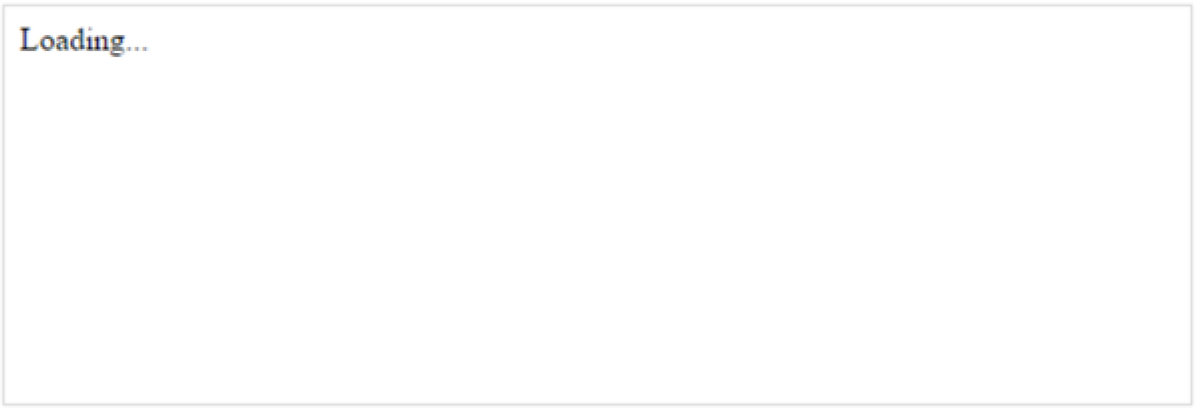
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above *app* folder which contains the ***.ts*** files.
- Open the terminal window and enter the following command:

```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **attribute_directive.html** file in your server root folder.
- Open this HTML file as **http://localhost/attribute_directive.html**. You will receive the following output.



```
Loading...
```

15. Angular 2 — Dependency Injection

Dependency Injection is a design pattern that passes an object as dependencies in different components across the application. It creates a new instance of class along with its required dependencies. The Dependency Injection is stimulated into the framework and can be used everywhere.

The following points need to be considered while using Dependency Injection:

- The ***injector*** mechanism maintains service instance and can be created using a ***provider***.
- The ***provider*** is a way of creating a service.
- You can register the ***providers*** along with injectors.

Example

The following example shows the use of dependency injection in Angular 2:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Angular 2 Dependency Injection</title>
    <script    src="https://cdnjs.cloudflare.com/ajax/libs/es6-shim/0.33.3/es6-
shim.min.js"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/systemjs/0.19.20/system-
polyfills.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/angular2-
polyfills.js"></script>
    <script src="https://code.angularjs.org/tools/system.js"></script>
    <script src="https://code.angularjs.org/tools/typescript.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/Rx.js"></script>
    <script src="https://code.angularjs.org/2.0.0-beta.6/angular2.dev.js"></script>
    <script>
      System.config({
        transpiler: 'typescript',
        typescriptOptions: { emitDecoratorMetadata: true },
        packages: {'app': {defaultExtension: 'ts'}},
        map: { 'app': './angular2/src/app' }
      });
      System.import('app/dependency_injection_main')
            .then(null, console.error.bind(console));
```

```
     </script>
   </head>
<body>
   <my-app>Loading...</my-app>
</body>
</html>
```

The above code includes the following configuration options:

- You can configure the ***index.html*** file using the *typescript* version. The SystemJS transpiles the TypeScript to JavaScript before running the application by using the ***transpiler*** option.
- If you do not transpile to JavaScript before running the application, you can see the compiler warnings and errors which are hidden in the browser.
- The TypeScript generates metadata for each and every class of the code when the ***emitDecoratorMetadata*** option is set. If you don't specify this option, large amount of unused metadata will be generated which affects the file size and impacts on the application runtime.
- Angular 2 includes the packages form the ***app*** folder where the files will have the ***.ts*** extension.
- It will further load the main component file from the ***app*** folder. If there is no main component file found, then it will display the error in the console.
- When Angular calls the bootstrap function in **main.ts**, it reads the Component metadata, finds the **'app'** selector, locates an element tag named app, and loads the application between those tags.

Let's create the ***TypeScript(.ts)*** files and save them in the ***app*** folder.

dependency_injection_main.ts

```
import {bootstrap} from 'angular2/platform/browser';  //importing bootstrap function
import {AppComponent} from "./fruit-component";     //importing component function

bootstrap(AppComponent);
```

fruit-component.ts

```
import {Component} from 'angular2/core';
import {MyListComponent}  from './play-component';

//@Component is a decorator that uses configuration object to create the component
and its view.
@Component({
  selector: 'my-app',  //The selector creates an instance of the component where it finds 'my-app' tag in parent HTML

  template:`<my-list></my-list>`,

  directives:[MyListComponent]  //'MyListComponent' is the root component of fruits that governs child components

})

export class AppComponent { }
```

play-component.ts

```
import {Component} from "angular2/core";
import {FruitService} from "./fruits.service";
import {Fruits} from "./fruits";
import {OnInit} from "angular2/core";

@Component({
   selector: "my-list",
   template: ` List of Fruits<br>

   <ul>

      <li *ngFor="#list of fruits">  //The NgFor directive instantiates a template once per item from an iterable
         {{list.id}} - {{ list.name }}
      </li>
   </ul>
   `,
   providers: [FruitService]  //providers are part of @Component metadata
})

//The 'MyListComponent' should get list of fruits from the injected 'FruitService'
export class MyListComponent implements OnInit {
   public fruits : Country[];

   //Using constructor, call the _fruitService and populate the fruits list
   constructor(private _fruitService: FruitService) {}

   getContacts(){
      this._fruitService.getContacts().then((fruits: Country[]) => this.fruits = fruits);
   }

   //The 'ngOnInit()' hook is called when done with creating the component and evaluated the inputs
   ngOnInit():any{
      this.getContacts();
   }
}
```

fruits.service.ts

```
import {Injectable} from "angular2/core";
import {COUNTRIES} from "./fruits.lists";

//It is used for meta data generation and specifies that class is available to an injector for instantiation
@Injectable()

//'FruitService' exposes 'getContacts()' method that returns list of data
export class FruitService {
   getContacts() {
      return Promise.resolve(COUNTRIES); // takes values from fruits.lists.ts file
   }
}
```

fruits.lists.ts

```
import {Fruits} from "./fruits";

//storing array of data in Fruits
export const COUNTRIES: Fruits[] =[
   {"id": 1, name :"Apple"},
   {"id": 2, name: "Grapes"},
   {"id": 3, name: "Orange"},
   {"id": 4, name: "Banana"}
```

fruits.ts

```
export interface Fruits{
   id: number;
   name: string;
}
```

Output

Let us now proceed with the following steps to see how the above code works:

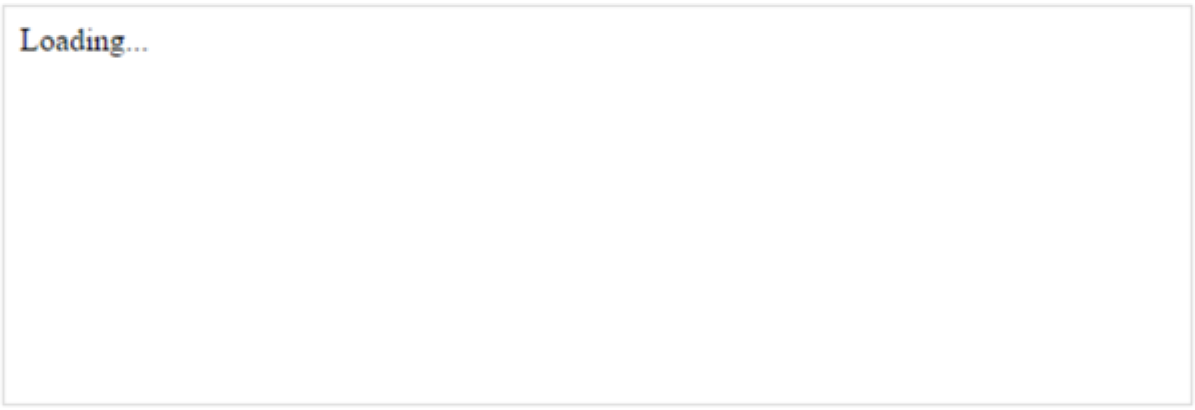
- Save the above HTML code as **index.html** file the way we created in the environment chapter and use the above ***app*** folder which contains ***.ts*** files.
- Open the terminal window and enter the command as given below:

```
npm start
```

- After a few moments, a browser tab should open. This tab displays the output as shown below.

OR you can run this file in another way:

- Save the above HTML code as **angular2_dependency_injection.html** file in your server root folder.
- Open this HTML file as **http://localhost/angular2_dependency_injection.html**. You will receive the following output.



```
Loading...
```