Contents

[Assumptions 3](#_Toc437595303)

[Solution developed with 3](#_Toc437595304)

[About this Tutorial 3](#_Toc437595305)

[Why Angular 2.0 3](#_Toc437595306)

[**What are the changes?** 4](#_Toc437595307)

[Setting up Sample Project 6](#_Toc437595308)

[Please refer to setup documentation to setup sample project.Walk through with sample application 6](#_Toc437595309)

[Component 7](#_Toc437595310)

[Routing 8](#_Toc437595311)

[Dependency Injection 9](#_Toc437595312)

[Displaying Data 10](#_Toc437595313)

[Http Request (Get & post) 12](#_Toc437595314)

[Other References 13](#_Toc437595315)

# Assumptions

This document is for those who are having fair idea about:

* Angular Version 1.\*
* TypeScript
* MVC Pattern

# Solution developed with

We need to use below technologies to develop the sample solutions for Angular2

* Visual Studio 2015 (pro)
* .NET Framework 4.5.2
* Angular2 Alpha version 46
* Node & Bower
* IIS 7 [Optional]

# About this Tutorial

This tutorial helps developer to learn Angular2 new functionalities and also they will get knowledge on how to develop Angular2 single page application using modern technologies such as TypeScript, Node and bower

Prerequisites for development

1. Install Visual Studio 2015(pro or Community)
2. Install Node, Bower components
3. IIS 7 – optional

# Why Angular 2.0

Mobile

The new Angular version will be focused on the development of mobile apps. The rationale is that it’s easier to handle the desktop aspect of things, once the challenges related to mobile (performance, load time, etc.) have been addressed.

Modular

Various modules will be removed from Angular’s core, resulting in better performance. These will find their way into Angular’s ever-growing ecosystem of modules, meaning you’ll be able to pick and choose the parts you need.

Modern

Angular 2.0 will target ES6 and “evergreen” modern browsers (those automatically updated to the latest version). Building for these browsers means that various hacks and workarounds that make Angular harder to develop can be eliminated allowing developers to focus on the code related to their business domain.

**What are the changes?**

**AtScript**

AtScript is a superset of ES6 and it’s being used to develop Angular 2.0. It’s processed by the Traceur compiler (along with ES6) to produce ES5 code and uses TypeScript’s type syntax to generate runtime type assertions instead of compile time checks. However, AtScript isn’t compulsory—you will still be able to use plain JavaScript/ES5 code instead of AtScript to write Angular apps.

**Improved Dependency Injection (DI)**

Dependency injection (a software design pattern in which an object is passed its dependencies, rather than creating them itself) was one of the factors that initially differentiated Angular from its competitors. It is particularly beneficial in terms of modular development and component isolation, yet its implementation was plagued with problems in Angular 1.x. Angular 2.0 will will address these issues, as well as adding missing features such as child injectors and lifetime/scope control.

**Annotations**

AtScript provides tools for associating metadata with functions. This facilitates the construction of object instances by providing the required information to the DI library (which will check for associated metadata when calling a function or creating an instance of a class). It will be also easy to override parameter data by supplying an Inject annotation.

**Child Injectors**

A child injector inherits all the services of its parent with the capability of overriding them at the child level. According to requirement, different types of objects can be called out and automatically overridden in various scopes.

**Instance Scope**

The improved DI library will feature instance scope control, which will become even more powerful when used with child injectors and your own scope identifiers.

**Templating and Data Binding**

Let’s take a look at templating and data binding as they go hand in hand when developing apps.

**Dynamic Loading**

This is a feature which is missing from the current version of Angular. It will be addressed by Angular 2.0, which will let developers add new directives or controllers on the fly.

**Directives**

In Angular 2.0 there will be three kinds of directive:

* **Component Directives**

These will create reusable components by encapsulating logic in JavaScript, HTML or an optional CSS style sheet.

* **Decorator Directive**

These directives will be used to decorate elements (for example adding a tooltip, or showing/hiding elements using ng-show/ng-hide).

* **Template Directive**

These will turn HTML into a reusable template. The instantiating of the template and its insertion into the DOM can be fully controlled by the directive author. Examples include ng-if and ng-repeat.

**Routing Solution**

The initial Angular router was designed to handle just a few simple cases, yet as the framework grew, more and more features were bolted on. The router in Angular 2.0 has been reworked to be simple, yet extensible. It will include the following basic features:

* + Simple JSON-based Route Config
  + Optional Convention over Configuration
  + Static, Parameterized and Splat Route Patterns
  + URL Resolver
  + Query String Support
  + Use Push State or Hashchange
  + Navigation Model (For Generating a Navigation UI)
  + Document Title Updates
  + 404 Route Handling
  + Location Service
  + History Manipulation

**Child Router**

The child router will convert each component of the application into a smaller application by providing it with its own router. It will help encapsulate entire feature sets of an application.

**Logging**

Angular2.0 will contain a logging service called diary.js a super useful feature which measures where time is spent in your application (thus enabling you to identify bottlenecks in your own code).

**Scope**

$scope will be removed in Angular 2.0 in favour of ES6 classes.

# 

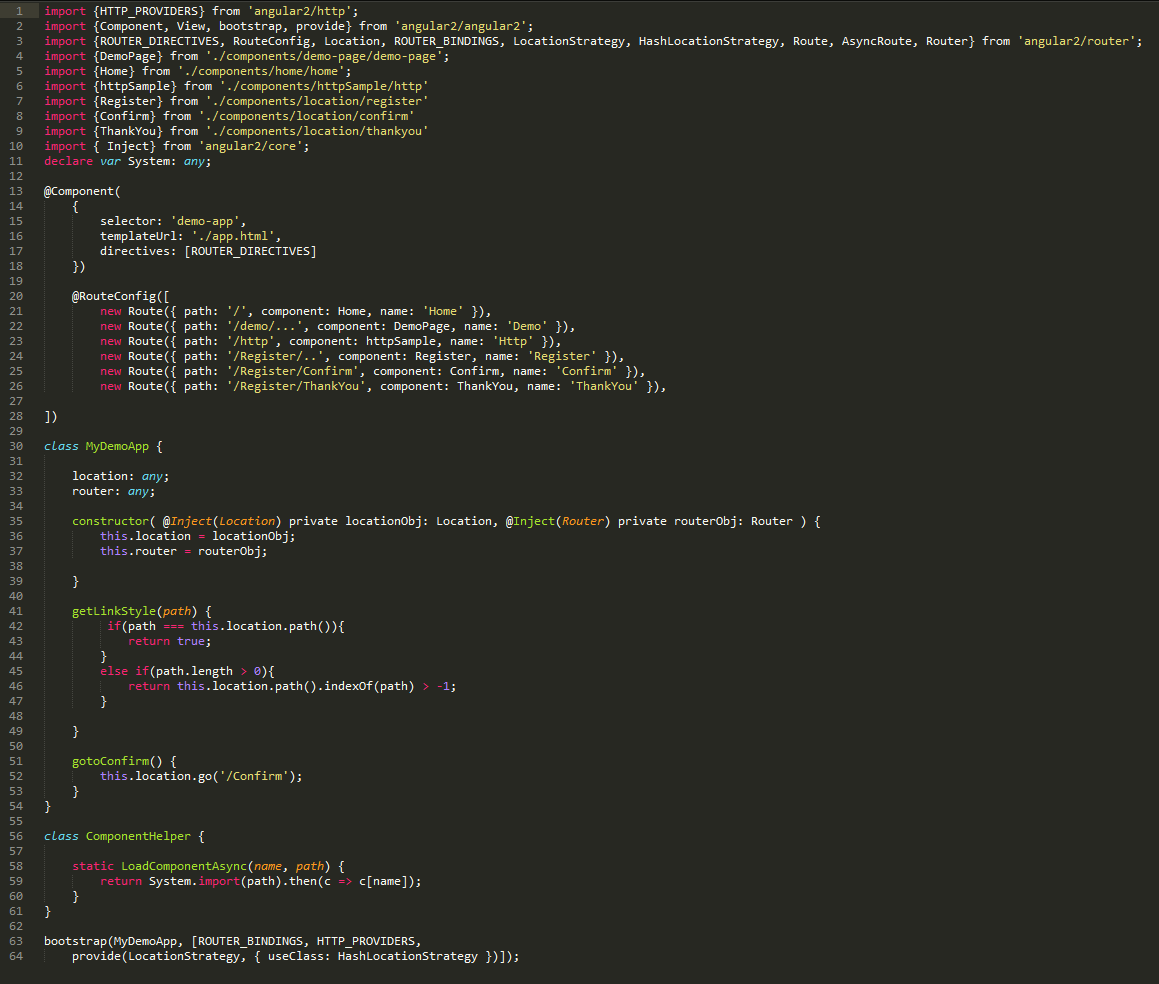
# Setting up Sample Project

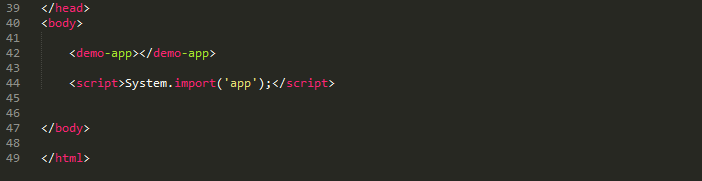
# Please refer to setup documentation to setup sample project.Walk through with sample application

## Component

In Angular 2, Components are the main way we build and specify elements and logic on the page.

App.ts

 Index



In Index.html line 42 <demo-app></demo-app> this tag refers to the name of the component, which binds the html template and calls the constructor, and rendered.

## Routing

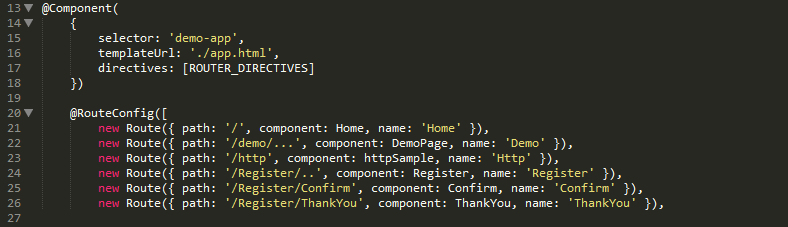
The setup of the router is pretty intuitive and it's really easy to get a simple nav enabled in your application. All you really need is a configuration section pointing to some components that will be loaded by specific routes. See the below example:

In recent version router functionality separated from the angulare2, so when ever if we need to use routing we need to add reference of routing separately as :

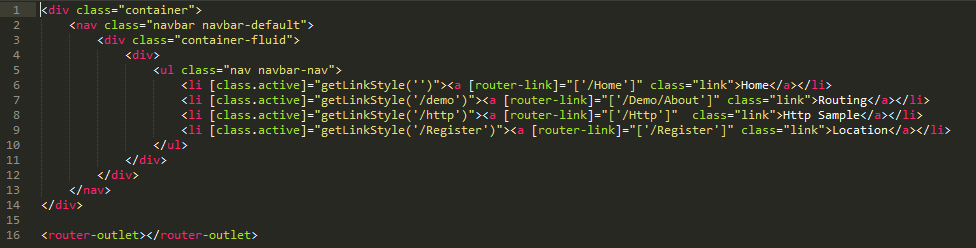


App.ts [router sample code]





App.html



So here need to add reference of router and add directives of router to enable routing in the application and add the route config. Each new rout has path, component and name.

Path: it refers the route path

Component: it is the component name that we going to navigate.

Name: which is alias name of that route, when we need to navigate on particular page we need to use this route name.

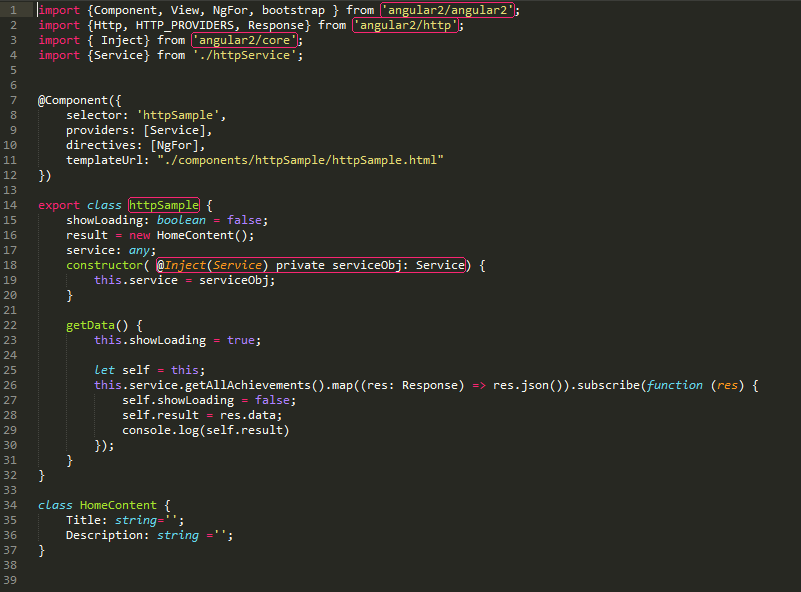
In the above example, we have used both routing like parent and child routing. To define the parent route we need to add route path as “/demo/..” which denotes parent route and thereafter each child route path we need to use “/Register/Confirm” which navigates to the child route.

In html markup if you see we have used [router-link] which configured with the route name.

## Dependency Injection

The main goal of DI is to avoid tightly coupled components by injecting dependencies rather than instantiating them directly in consuming components. The advantages of this might not be obvious at first, but the decoupling makes your components much more flexible since you can “inject” different implementations to the same component – provided the implementations conform to a standard contract. A typical application of this is unit testing since DI allows you to pass in a mock implementation without requiring changes to the components under test.

For Example refer to the Components/httpSample/http.ts

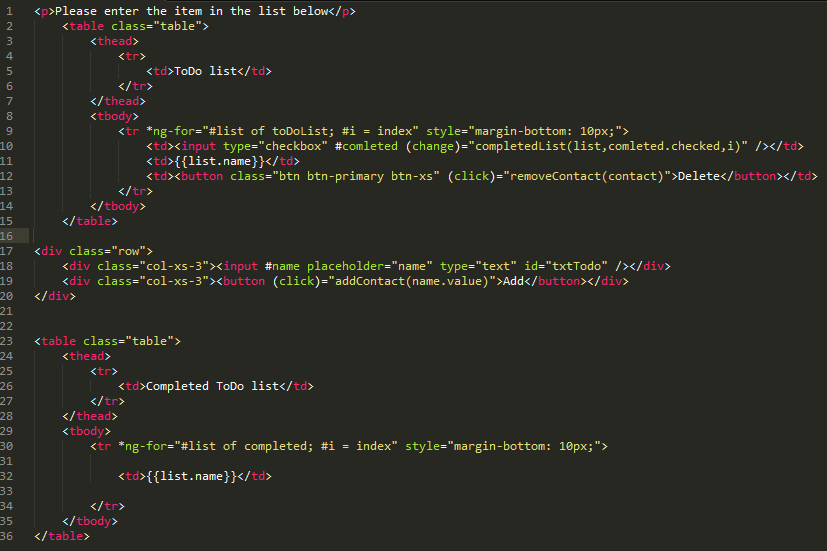


Above example shows how used httpService as a dependency injection.

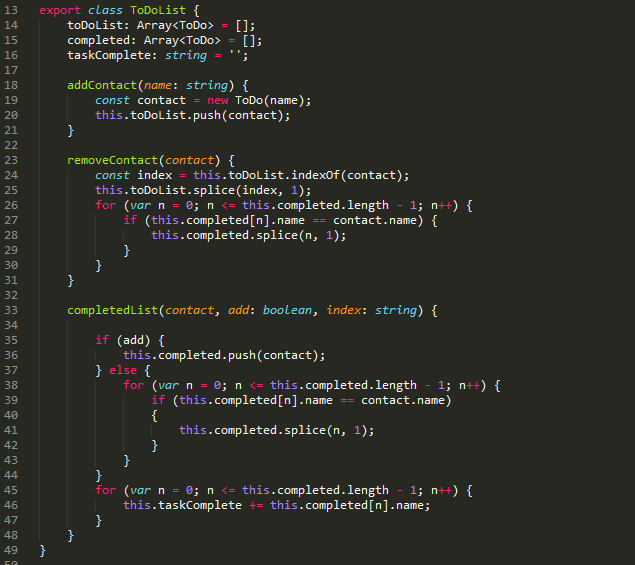
## Displaying Data

Displaying data means getting data from Web API or showing data from input controls to Web UI. In Angular2 the concept of displaying data or play around with UI element has changed little bit. If you refer the sample code todo.html or

Todo.html



Todo.ts



In sample code getting a textbox value by **#name** where it holds the value and passed in the parameter of change event, we can also use instead [(ng-model)]=””

The syntax also has changed for ng-for, how to trigger click events and all etc..

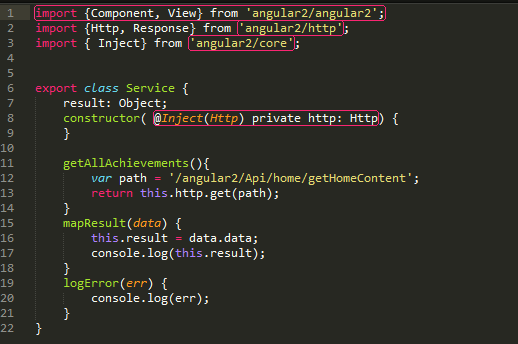
## Http Request (Get & post)

Performs http requests using XMLHttpRequest as the default backend. To perform xmlhttp request need to import Http Class and inject in our component of angular.

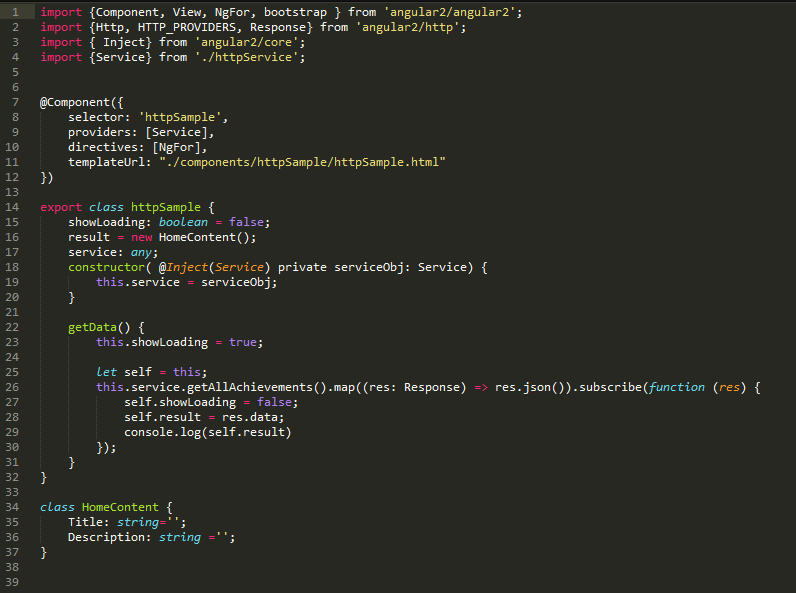
Http is available as an injectable class, with methods to perform http requests. Calling request returns an Observable which will emit a single Response when a response is received.

In below sample code used MVC controller service as web api which returns the Json data and binds it on the UI

httpService.ts



http.ts



## Other References

Forms and validations

<http://www.syntaxsuccess.com/viewarticle/forms-and-validation-in-angular-2.0>

Dependency Injection

<http://www.syntaxsuccess.com/viewarticle/dependency-injection-in-angular-2.0>

Using JQuery and Angular2

<http://www.syntaxsuccess.com/viewarticle/using-jquery-with-angular-2.0>

Angular2 and HTTP

<http://www.syntaxsuccess.com/viewarticle/angular-2.0-and-http>

Angular Quick Start

<https://angular.io/docs/ts/latest/quickstart.html>

TypeScript Hand Book

<http://www.typescriptlang.org/Handbook>