# What's New in Angular 6? What Are Improvements In Angular 6?

Let’s start to explore all changes of Angular 6 step by step!

* Added ng update - This CLI commands will update your angular project dependencies to their latest versions. The ng update is normal package manager tools to identify and update other dependencies. *ng update*
* Angular 6 uses RxJS 6 - this is the third-party library (RxJS) and introduces two important changes as compared to RxJS 5.

1. RxJS 6 introduces a new internal package structure

2. Operator concept

Both are requires you to update your existing code. To update to RxJS 6, you simply run *npm install --save rxjs@6*

Simply run the blow command and update your existing Angular project *npm install --save rxjs-compat*

Alternatively, you can use the command - *ng update rxjs* to update *RxJS* and install the *rxjs-compat* package automatically.

* + RxJS 6 Related import paths -

Instead of -

*import { Observable } from 'rxjs/Observable';*

*import { Subject } from 'rxjs/Subject';*

Use a single import -

*import { Observable, Subject } from 'rxjs';*

So all from *rxjs/Something* imports become from one *'rxjs'*

* Operator imports have to change -

Instead of

*import 'rxjs/add/operator/map';*

*import 'rxjs/add/operator/throttle';*

Now you can use -

*import { map, throttle } from 'rxjs/operators';*

And Instead of

*import 'rxjs/add/observable/of';*

*Now you can use -*

*import { of } from 'rxjs';*

* *RxJS 6 Changes - Changed Operator Usage*

*Instead of-*

*import 'rxjs/add/operator/map';*

*import 'rxjs/add/operator/throttle';*

*yourObservable.map(data => data \* 2)*

*.throttle(...)*

*.subscribe(...);*

* You can use the new pipe () method,

*import { map, throttle } from 'rxjs/operators';*

*yourObservable*

*.pipe(map(data => data \* 2), throttle(...))*

*.subscribe(...);*

* CLI update and added a new project config file - Instead of “.angular-cli.json” using “angular.json”

Now in Angular 6 new projects use an “angular.json” file instead of “.angular-cli.json” file. *ng update @angular/cli --from=1 --migrate-only*

The above command helps you to update your existing “.angular-cli.json” file to the new “angular.json” file.  
  
*The “angular.json” file contains the Properties -*

1. *Version* - This is integer file format version and it is currently 1.

2. *newProjectRoot* - This is string path where new projects will be created.

3. *defaultProject* - This is default project name used in commands.

4. *CLI* - This is workspace configuration options for Angular CLI and it contains

a. defaultCollection

b. packageManager

c. warnings

d. And so on.

5. *Schematics* - This is configuration options for Schematics.

6. *Projects* - This is configuration options for each project in the workspace and it contains

a. root

b. sourceRoot

c. projectType

d. prefix

e. schematics

f. Architect - This is the project configuration for Architect targets.

* The <template> deprecated, Now Angular 6 introduce <ng-template> –

Now in Angular 6, you should use *<ng-template>* instead of *<template>*

For example, previously you are using

*<template [ngIf]="IsAdmin">*

*<p>This template renders only if IsAdmin is true.</p>*

*</template>*

Now in Angular 6, you should use <ng-template> instead of <template>

<ng-template [ngIf]="IsAdmin">

<p>This template renders only if IsAdmin is true.</p>

</ng-template>

* Service level changes (the way of marking a service as global) -

In the earlier versions, if you want to provide a service to the entire application –you should add it to *providers []* in the AppModule but in the Angular 6 released you should not add in the *providers []* in the AppModule. Example for marking a service as global -

Instead of

*//my.service.ts*

*export class MyService { }*

*//In app.module.ts*

*//JavaScript imports services*

*import { MyService } from './my-serice.service';*

*//AppModule class with the @NgModule decorator*

*@NgModule({*

*declarations: [],*

*providers: [MyService] //My services instances are now available across the entire app.*

*})*

*export class AppModule {*

*//exporting app module*

*}*

Use with Angular 6 released-

*//my.service.ts*

*@Injectable({providedIn: 'root'})*

*export class MyService { }*

*@NgModule({*

*declarations: [],*

*providers: [] // Service does not need to be added here*

*})*

*export class AppModule {}*

The second one obviously saves you some lines of code as compare to previous code.

* **Angular 6 introduces Angular Elements** -The elements are a feature that allows you to compile Angular components to native web components which you can use in your Angular application. An angular element is a package which is part of the Angular framework @angular/elements.
* **Angular 6 introduces new Ivy Renderer** -The new Ivy renders and it’s not stable for now and it’s only in beta version. It will stable in future for production. Ivy Renderer is new rendering engine which is designed to be backward compatible with existing render and focused to improve the speed of rendering and it optimizes the size of the final package. The main goal of Ivy render is to speed up its loading time and reduce the bundle size of your applications. Also for uses a different approach for rendering Angular components. For Angular, this will not be default renderer, but you can manually enable it in compiler options.
* **Bazel Compiler -**The Bazel Complier is a build system used for nearly all software built at Google. From Angular 6 release, will start having the Bazel compiler support and when you compile the code with Bazel Compiler, you will recompile entire code base, but it compiles only with necessary code. The Bazel Complier uses advanced local and distributed caching, optimized dependency analysis and parallel execution.
* ***Replace Context, Record and Injectors*** *-*
* *Replace* ngOutletContext with ngTemplateOutletContext
* *Replace* CollectionChangeRecord with IterableChangeRecord
* Now use Renderer2, Instead of Renderer
* Now use StaticInjector, Instead of ReflectiveInjector,

Angular 6 Renamed Operators -

The lists of renamed operators are –

1. do() => tap()

2. catch() => catchError()

3. finally() => finalize()

4. switch() => switchAll()

5. throw() => throwError()

6. fromPromise() => from()

Angular 6 introduces multiple validators for array method of FormBuilder –

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

*import {FormsModule, FormBuilder, FormGroup} from '@angular/forms';*

*constructor(private fb: FormBuilder) {}*

*myForm: FormGroup;*

*ngOnInit() {*

*this.myForm = this.fb.group({*

*text: ['', Validators.required],*

*options: this.fb.array([], [MyValidators.minCount, MyValidators.maxCount])*

*});*

*}*

**Addition of navigationSource and restoredState to NavigationStart -**

These two properties help us to handle multiple use cases in routing.

NgModelChange - Now emitted after value and validity is updated on its control. Previously, it was emitted before updated.

As the updated value of the control is available, the handler will become more powerful

Previously -

*<input [(ngModel)]="name" (ngModelChange)="onChange($event)">*

And

*onChange(value) {*

*console.log(value); // would log the updated value, not old value*

*}*

Now Use -

*<input #modelDir="ngModel" [(ngModel)]="name" (ngModelChange)="onChange(modelDir)">*

And

*onChange(NgModel: NgModel) {*

*console.log(NgModel.value);// would log old value, not updated value*

*}*

Form Control statusChanges – Angular 6 emits an event of “PENDING” when we call Abstract Control markAsPending.

New optional generic type ElementRef – This optional generic type will help to get hold of the native element of given custom Element as ElementRef Type.

***Let’s see in depths***-

**1)** Typescript 2.6.x supports

**2)** Added Angular Material and CDK Stable

**3)** Component Dev Kit (CDK) - CDK allows you to build your own library of UI components using Angular Material.

**4)** Improved decorator error messages

**5)** Fix platform-detection example for Universal

**6)** Ivy Renderer - It is a new backward compatible and main focused area - speed improvements, size reduction, and increased flexibility.

**7)** Add afterContentInit and afterContentChecked to render

**8)** Added to supports of nativeElement

**9)** Added Optional generic type for ElementRef

The Example looks like -

@ViewChild('your-element') yourElement:ElementRef;

**10)** Bazel Compiler - Bazel only rebuilds what is necessary.

**11)** Added Test Comment

**12)** Add missing lifecycle tests for projected components

**13)** Closure Compiler - Closure Compiler consistently generates smaller bundles.

**14)** Rename QueryPredicate to LQuery and LQuery to LQueries

**15)** Service Worker - Service worker is a script that runs in the web browser. It also manages caching for an application.

**16)** Added multiple validators for array method of FormBuilder

**17)** Handle string with and without line boundary - Now Handle string with and without line boundary (^ & $) on pattern validators. Previously, it works with string not boundaries.

**18)** AbstractControl statusChanges - Previous version, not emits an event when you called “markAsPending” but now emits an event of "PENDING" when we call AbstractControl markAsPending.

**19)** Updates on NgModelChange - Now emitted after value and validity is updated on its control. Previously, it was emitted before updated.

**20)** Allow HttpInterceptors to inject HttpClient –

Previously, an interceptor attempting to inject HttpClient directly would receive a circular dependency error, as HttpClient was constructed via a factory which injected the interceptor instances. Users want to inject HttpClient into interceptors to make supporting.

Either HttpClient or the user has to deal specially with the circular Dependency. This change moves that responsibility into HttpClient itself. By utilizing a new class HttpInterceptingHandler which lazily Loads the set of interceptors at request time, it's possible to inject HttpClient directly into interceptors as construction of HttpClient no longer requires the interceptor chain to be constructed.

**21)** Add navigationSource and restoredState to NavigationStart – Currently, NavigationStart there is no way to know if navigation was triggered imperatively or via the location change. These two use cases should be handled differently for a variety of use cases (e.g., scroll position restoration). This PR adds a navigation source field and restored navigation id (passed to navigations triggered by a URL change).

**22)** Add type and hooks to directive def

**23)** Enable size tracking of a minimal CLI render3 application

**24)** Add canonical view query

**25)** Language Service – The 2.6 version of Typescript’s “resolveModuleName” started to require paths passed to be separated by '/' instead of being able to handle '\'.

**What's New In Angular 5? What Are the Improvements In Angular 5?**

**Included Key Features - Angular 5**

1) Include Representation of Placeholders to xliff and xmb in the compiler

2) Include an Options Arg to Abstract Controls in the forms controls

3) Include add default updateOn values for groups and arrays to form controls

4) Include updateOn blur option to form controls

5) Include updateOn submit option to form controls

6) Include an Events Tracking Activation of Individual Routes

7) Include NgTemplateOutlet API as stable in the common controls

8) Create StaticInjector which does not depend on Reflect polyfill

9) Include [@.disabled] attribute to disable animation children in the animations

10) Make AOT the default

11) Watch mode

12) Type checking in templates

13) More flexible metadata

14) Remove \*.ngfactory.ts files

15) Better error messages

16) Smooth upgrades

17) Tree-Shakeable components

18) Hybrid Upgrade Application

**Included Performance Improvements - Angular 5**

1) Use of addEventListener for the faster rendering and it is the core functionality.

2) Update to new version of build-optimizer.

3) Include some Improvements on the abstract class methods and interfaces

4) Remove decorator DSL which depends on Reflect for Improve the Performance of Apps and This is the core functionality.

5) Include an option to remove blank text nodes from compiled templates

6) Switch Angular to use Static-Injector instead of Reflective-Injector.

7) Improve the applications testing.

8) Improve the performance of hybrid applications

9) Improvements on Lazy loading for Angular

**Improvement on HttpClient – Included**

1) Improvement on Type-checking the response

2) Improvement on Reading the full response

3) Improvement on Error handling and fetching error details

4) Improvement on Intercepting all requests or responses

5) Improvement on Logging

6) Improvement on Caching

7) Improvement on XSRF Protection

**Angular Router Life Cycle Events -**

Added new router life cycle events for Guards and Resolvers -

1) GuardsCheckStart,

2) GuardsCheckEnd,

3) ResolveStart and

4) ResolveEnd

[**What’s New in Angular 4? AND what are the Improvements in Angular 4?**](https://www.code-sample.com/2017/03/angular-4-vs-angular-2-difference.html)

Angular 4 contains some additional Enhancement and Improvement. Consider the following enhancements.

**1.** Smaller & Faster Apps

**2.** View Engine Size Reduce

**3.** Animation Package

**4.** NgIf and ngFor Improvement

**5.** Template

**6.** NgIf with Else

**7.** Use of AS keyword

**8.** Pipes

**9.** HTTP Request Simplified

**10.** Apps Testing Simplified

**11.** Introduce Meta Tags

**12.** Added some Forms Validators Attributes

**13.** Added Compare Select Options

**14.** Enhancement in Router

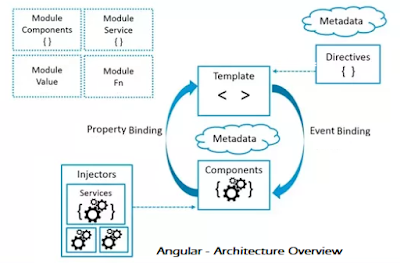
**15.** Added Optional Parameter

**16.** Improvement Internationalization

I hope you are enjoying with this post! Please share with you friends. Thank you so much!

# What Is Architecture Overview of Angular?

The bootstrapping process creates the components listed in the bootstrap array and inserts each one into the browser (**DOM**).



With helps of above architecture overview, you can identify the seven main building blocks of an Angular Application.

1. [Component](https://www.code-sample.com/2018/04/what-are-components-in-angular-6-54-and.html)

2. Templates

3. [Metadata](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html)

4. Data Binding

5. Directives

6. Services

7. [Dependency Injection](https://www.code-sample.com/2018/05/dependency-injection-di-angular-5-6.html)

The basic building blocks of an Angular application are *NgModules*, which provide a compilation context for components. Angular app is defined by a set of *NgModules* and it always has at least a root module that enables bootstrapping, and many more feature modules.

* Components define Template views,
* Components use services

The Angular Module (***NgModules) helps us to organize an application into connected blocks of functionality.***

The NgModule properties for the minimum “**AppModule**” generated by the CLI which are follow as -

* + **Declarations** — Use to declare the application components.
  + **Imports** —Every application must import BrowserModule to run the app in a browser.
  + **Providers** — There are none to start.
  + **Bootstrap** — This is a root AppComponent that Angular creates and inserts into the index.html host web page.

By default Bootstrap file is created in the folder “**src/main.ts**” and “**main.ts**” file is very stable. Once you have set it up, you may never change it again.

What Is Bootstrapping (bootstrap) in Angular?

The Bootstrap is the **root** AppComponent that Angular creates and inserts into the “**index.html**” host web page.

<body>

<app-root></app-root>

</body>

**index.html -**

**<!doctype html>**

**<html lang="en">**

**<head>**

**<meta charset="utf-8">**

**<title>MyApp</title>**

**<base href="/">**

**<meta name="viewport" content="width=device-width, initial-scale=1">**

**<link rel="icon" type="image/x-icon" href="favicon.ico">**

**</head>**

**<body>**

**<app-root></app-root>**

**</body>**

**</html>**

You can put more than one component tree on a host web page, that's not typical. Most of the applications have only one component tree and they bootstrap a single root component and you can call the one root component anything you want but most developers call it AppComponent.

The bootstrapping process creates the components listed in the bootstrap array and inserts each one into the browser (**DOM**).

The Angular Module ([**NgModules**](https://www.code-sample.com/2017/04/angular-4-interview-questions-and.html)**) helps us to organize an application into connected blocks of functionality.**

The NgModule properties for the minimum “**AppModule**” generated by the CLI which are follow as -

**Declarations** — Use to declare the application components.

**Imports** —Every application must import BrowserModule to run the app in a browser.

**Providers** — There are none to start.

**Bootstrap** — This is a root AppComponent that Angular creates and inserts into the index.html host web page.

**app.module.ts -**

**import { BrowserModule } from '@angular/platform-browser';**

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

**import { AppComponent } from './app.component';**

**import { LoginComponent } from './login/login.component';**

**import { SignupComponent } from './signup/signup.component';**

**@NgModule({**

**declarations: [**

**AppComponent,**

**LoginComponent,**

**SignupComponent**

**],**

**imports: [**

**BrowserModule**

**],**

**providers: [],**

**bootstrap: [AppComponent]**

**})**

**export class AppModule { }**

By default Bootstrap file is created in the folder “**src/main.ts**” and “**main.ts**” file is very stable. Once you have set it up, you may never change it again and its looks like -

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

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';

import { environment } from './environments/environment';

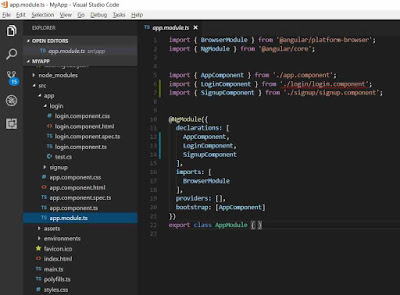
if (environment.production) {

enableProdMode();

}

platformBrowserDynamic().bootstrapModule(AppModule)

.catch(err => console.log(err));



I hope you are enjoying with this post! Please share with you friends. Thank you!!

**What Is a Template Reference Variable?**

A template reference variable is a way of capturing a reference to a specific element, component, directive, and pipe so that it can be used someplace in the same template HTML.

You should declare a reference variable using the hash symbol (#).

Template Reference Variable Syntax –

You can use a template reference variable by two ways.

1. Using hash symbol (#)

2. Using reference symbol (ref-)

The following examples of specifying a template reference variable using Input Text Box –

I have declared a reference variable “cellnumber” using the **hash symbol (#)** and **reference symbol (ref-)**.

**<input type="text" ref-cellnumber> //cellnumber will be a template reference variable.**

And

**<input #cellnumber placeholder="Cell number"> //cellnumber will be a template reference variable.**

I have created a reference to the input element that can be used later on in my template and the scope for “**cellnumber**” variable is the entire HTML template in which the reference is defined.

Here is how I could use that reference to get the value of the input for instance –

**//cellnumber refers to the input element**

**<button (click)="show(cellnumber)">click to see</button>**

In the below line of code, the variable “cellnumber” refer to the HTMLElement object instance for the input -

**show(cellnumber: HTMLInputElement){**

**console.log(cellnumber.value);**

**}**

You can use the ViewChild decorator to reference it inside your component.

**import {ViewChild, ElementRef} from '@angular/core';**

**// Reference cellnumber variable inside Component**

**@ViewChild('cellnumber') cellInputRef: ElementRef;**

And finally, you can use this.nameInputRef anywhere inside your component class.

**show(){**

**this.contactNumber = this.cellInputRef.nativeElement.value**

**}**

Template Reference Variable with NgForm –

Here we will discuss about how to access NgForm directive using template reference variable.

**<form (ngSubmit)="onSubmitEmployee(empForm)" #empForm="ngForm">**

**<label>F-Name </label><input name="f-name" required [(ngModel)]="employee.fname">**

**<label>L-Name </label><input name="l-name" required [(ngModel)]="employee.lname">**

**<label>Age </label><input name="age" required [(ngModel)]="employee.age">**

**<button type="submit" [disabled]="!empForm.form.valid">Submit</button>**

**</form>**

In the above NgForm example contains an ngSubmit event and form directive.

The ngSubmit – The ngSubmit directive specifies a function to run when the form is submitted. Here on form submit onSubmitEmployee component method will be called.

The NgForm - It is nestable alias of form directive. The main purpose of ngForm is to group the controls, but not a replacement of <form> tag.

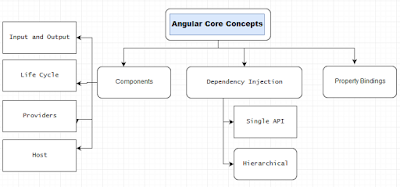
# As you know, the HTML does not allow nesting of form elements. It is very useful to nest forms. What Are Components in Angular?

Components are the most basic building block of a UI in Angular applications and it controls views (HTML/CSS). They also communicate with other components and services to bring functionality to your applications.

Technically components are basically [TypeScript](https://www.code-sample.com/2017/06/typescript-interview-questions-and.html) classes that interact with the HTML files of the components, which get displayed on the browsers.

The component is the core functionality of Angular applications but you need to know to pass the data into the components to configure them.

Angular applications must have a root component that contains all other components.



Components are created using **@Component** decorator that is part of **@angular/core module**.

You can create your own project using Angular CLI, this command allows you to quickly create an Angular application like - generate components, services, pipes, directive, classes, and modules, and so on as per your requirements.

Create your own component (login) using below command line –

ng g component login

After executing the above Angular CLI command in your project directory, the result looks like –

**D:\Angular\DemoApp>ng g component login**

**create src/app/login/login.component.html (24 bytes)**

**create src/app/login/login.component.spec.ts (621 bytes)**

**create src/app/login/login.component.ts (265 bytes)**

**create src/app/login/login.component.css (0 bytes)**

**update src/app/app.module.ts (394 bytes)**

And the application login files are created by default and it looks like –

1. login.component.html

2. login.component.spec.ts

3. login.component.ts

4. login.component.css

5. app.module.ts

And Angular CLI commands also import the Login component in the Angular module.

See the example in details -

login.component.ts -

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

@Component({

selector: 'app-login',

templateUrl: './login.component.html',

styleUrls: ['./login.component.css']

})

export class LoginComponent implements OnInit {

constructor() { }

ngOnInit() {

}

}

The above component class shows some of the most useful **@Component** configuration options –

1. Selector

2. TemplateUrl

3. StyleURLs

The selector – It is a CSS selector that tells Angular to create an instance of this component wherever it finds the corresponding tag in template HTML. For example, it is - <app-login></app-login>

The templateUrl – It is the module-relative address of this component's HTML template and you can also provide the inline HTML template.

The styleUrls - It can be used for CSS rules and it will affect the style of the template elements and you can also provide the inline style CSS.

The components provide you some additional metadata configurations–

@Component({

changeDetection?: ChangeDetectionStrategy

viewProviders?: Provider[]

moduleId?: string

templateUrl?: string

template?: string

styleUrls?: string[]

styles?: string[]

animations?: any[]

encapsulation?: ViewEncapsulation

interpolation?: [string, string]

entryComponents?: Array<Type<any> | any[]>

preserveWhitespaces?: boolean

// inherited from core/Directive

selector?: string

inputs?: string[]

outputs?: string[]

host?: {...}

providers?: Provider[]

exportAs?: string

queries?: {...}

})

app.module.ts –

import { BrowserModule } from '@angular/platform-browser';

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

import { AppComponent } from './app.component';

import { LoginComponent } from './login/login.component';

@NgModule({

declarations: [

AppComponent,

LoginComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

login.component.html –

<p>

Welcome you, Anil!

</p>

login.component.spec.ts –

import { TestBed, async } from '@angular/core/testing';

import { AppComponent } from './app.component';

describe('AppComponent', () => {

beforeEach(async(() => {

TestBed.configureTestingModule({

declarations: [

AppComponent

],

}).compileComponents();

}));

it('should create the app', async(() => {

const fixture = TestBed.createComponent(AppComponent);

const app = fixture.debugElement.componentInstance;

expect(app).toBeTruthy();

}));

it(`should have as title 'app'`, async(() => {

const fixture = TestBed.createComponent(AppComponent);

const app = fixture.debugElement.componentInstance;

expect(app.title).toEqual('app');

}));

it('should render title in a h1 tag', async(() => {

const fixture = TestBed.createComponent(AppComponent);

fixture.detectChanges();

const compiled = fixture.debugElement.nativeElement;

expect(compiled.querySelector('h1').textContent).toContain('Welcome to app!');

}));

});

The detail about Component’s Metadata Properties List -

1. Selector Property – The CSS selector that identifies this component in a template.

2. StyleUrls Property – The list of URLs to style sheets to be applied to this component's view.

3. Styles Property – To be applied the inline styles for the component's view.

4. Template Property – To be applied the inline template for the component's view.

5. TemplateUrl Property – Used the URLs to an external file containing a template for the view

6. Animations Property – Applied the list of animations of this component.

7. ChangeDetection Property – The change detection strategy used by this component.

8. Encapsulation Property - The style encapsulation strategy used by this component.

9. EntryComponents Property – Used the list of components that are dynamically inserted into the view of this component.

10. ExportAs Property – The name under which component instance is exported in a template.

11. Host Property – Used to map the class property to host element bindings for events, properties, and attributes.

12. Inputs Property – The list of class property names to data-bind as component inputs.

13. Interpolation Property – The custom interpolation markers used in this component's template.

14. ModuleId Property – This is the CommonJS module id of the file in which this component is defined.

15. Outputs Property – The list of class property names that expose output events that others can subscribe too.

16. Providers Property – The list of providers available to this component and its children.

17. Queries Property – To configure queries that can be injected into the components.

18. ViewProviders Property – The list of providers available to this component and its view children.

**Summary** -

Components are fundamental building blocks of UI in Angular applications and it communicates with other components and services to bring functionality to your applications.

1. It is a core component of Angular applications.

2. An angular application must have a root component that contains all other components.

3. They have well-defined selector.

4. They have well-defined styles and styleUrls.

5. They have well-defined template and templateUrl.

6. They have well-defined inputs and outputs.

7. They have well-defined encapsulation and animations.

8. They have a well-defined lifecycle.

9. They are self-describing property.

10. And many more...

# What Is an EntryComponent in Angular 5, and 4?

[Anil Singh](https://plus.google.com/116567432705413398625) [5:14 AM](https://www.code-sample.com/2018/04/angular-6-5-4-entry-component.html)

**EntryComponent of Angular -**

The entry component is used to define components and created **dynamically** using the ComponentFactoryResolver.

Firstly, Angular creates a component factory for each of the [bootstrap components](https://www.code-sample.com/2018/04/what-are-components-in-angular-6-54-and.html) with the help of**ComponentFactoryResolver**. And then, at run-time, it will use the factories to instantiate the components.

You specify an entry component by bootstrapping in the **Angular module** or you specify an entry component by routing definition.

All other root components should be listed in the declarations array.

const routes: Routes = [

{ path: '', redirectTo: 'home', pathMatch: 'full'},

{ path: 'login', component: LoginComponent },

{ path: 'dashboard', component: DasboardComponent },

{ path: '\*\*', redirectTo: 'home' }

];

There are two main kinds of entry components which are following -

1. The bootstrapped root component

2. A component you specify in a route

The bootstrapped entry component -

A bootstrapped component is an entry component that Angular loads into DOM at the application launch and the other root components loaded dynamically into entry components.

The angular loads a root dynamically because it is bootstrapped in the Angular Module. In the below example, AppComponent is a root component so that angular loads dynamically.

Example –

import { BrowserModule } from '@angular/platform-browser';

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

import { AppComponent } from './app.component';

import { LoginComponent } from './login/login.component';

@NgModule({

declarations: [

AppComponent,

LoginComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent] // bootstrapped entry component

})

export class AppModule { }

A Routed entry component -

All router components must be entry components because the component would require you to add in two places.

1. Router and

2. EntryComponents

The Angular compiler is so smarter and it is recognizing that this is a router component and it automatically added router components into entry components.

The route definition refers to components by its type i.e.

1. LoginComponent

2. DasboardComponent

There are two components one is Login and another one is Dashboard. These components have the

ability to navigate between the login and dashboard views if passed the authentication and authorization of this app.

Example -

const routes: Routes = [

{ path: '', redirectTo: 'home', pathMatch: 'full'},

{ path: 'login', component: LoginComponent },

{ path: 'dashboard ', component: DasboardComponent },

{ path: '\*\*', redirectTo: 'home' }

];

Stayed Informed - [What Are Components in Angular 5,4 and 2?](https://www.code-sample.com/2018/04/what-are-components-in-angular-6-54-and.html)

Stayed Informed - [Why doesAngular need entryComponents?](https://www.code-sample.com/2018/04/why-does-angular-need-entrycomponents.html)

I hope you enjoyed this post. So please write your thoughts in the below comment box. Thank you so much for reading this post.

# When do I add components to entryComponents?

[Anil Singh](https://plus.google.com/116567432705413398625) [1:34 AM](https://www.code-sample.com/2018/04/angular-5-add-components-to.html)

Most the great application developers would not need to adds components to the entry components and Angular appends few components to entry components automatically.

Entry components are not tied to routes. They are loaded dynamically and are not referenced in component templates.

# Why does Angular need entryComponents?

[Anil Singh](https://plus.google.com/116567432705413398625) [9:03 PM](https://www.code-sample.com/2018/04/why-does-angular-need-entrycomponents.html)

Why does Angular need entryComponents?

The entry components improve the performance, smallest, fastest and reusable code of your production apps.  
  
For example, if you want to load the smallest, fastest and reusable code in your production apps. These codes contain only the classes that you actually need and it should exclude the components that are never used, whether or not those components are declared in the apps.  
  
Stayed Informed - [What Are Components in Angular 5,4 and 2?](https://www.code-sample.com/2018/04/what-are-components-in-angular-6-54-and.html)

As you know, many libraries declare and [export components](https://www.code-sample.com/2018/04/what-are-components-in-angular-6-54-and.html) you will never use in your app. If you do not reference them, the tree shaker drops these libraries and components from the final code package.

@NgModule({

declarations: [

AppComponent

],

imports: [BrowserModule],

providers: [],

bootstrap: [AppComponent] // bootstrapped entry component

})

export class AppModule { }

If a component is not in an entry component, the compiler skips compiling for this component.

I hope you enjoy this post. So please give your thoughts in the below comments box. Thank you!

# What Is Modules (@NgModule decorator)?

[Anil Singh](https://plus.google.com/116567432705413398625) [1:54 AM](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html)

The NgModule is a TypeScript class marked by the @NgModule decorator.

The NgModule is a class and work with the @NgModule decorator function and also takes a metadata object that tells Angular how to compile and run module code.

The [**Angular**](https://www.code-sample.com/2016/11/javascript-jquery-angular-2-nodejs-sql.html) **module helps you to organize an application into associative blocks of functionality.**

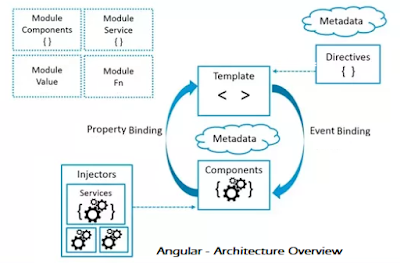
An angular module represents a core concept and plays a fundamental role in structuring Angular applications.

The NgModule is used to simplify the ways you define and manage the dependencies in your applications and also you can consolidate different components and services into associative blocks of functionality.

Every Angular application should have at least one module and it contains the components, service providers, pipes and other code files whose scope is defined by the containing NgModule.

The purpose of the module is to declare everything you create in Angular and group them together.

Every application has at least one Angular module, the root module that you bootstrap to launch the application. The Angular root module is called AppModule.



The module is a way to organize your dependencies for

1. Compiler

2. Dependency Injection

A module can import other modules and can expose its functionality to other modules. The modules can be loaded eagerly when the application starts or lazy loaded asynchronously by the router.

The angular loads a root dynamically because it is bootstrapped in the Angular Module.

An Angular app needs at least one module that serves as the root module.

You can use CLI commands to generate an app, the default AppModule is as follows –

ng new yourApp

The above CLI command is used to create a new Angular project and this CLI command automatically creates several folders and files which are necessary for project development, testing, and configuration and so on.

The Angular CLI (**Command Line Interface**) is a tool to initialize, develop, scaffold and maintain Angular applications. To use this we need to install it first and it should be installed globally on your machine.

npm install -g @angular/cli

The following is an example of specifying a NgModule -

//JavaScript imports

import { BrowserModule } from '@angular/platform-browser';

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

import { AppComponent } from './app.component';

import { LoginComponent } from './login/login.component';

//AppModule class with @NgModule decorator.

@NgModule({

declarations: [

AppComponent,

LoginComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent] // bootstrapped entry component

})

export class AppModule { }

**What are the @NgModule Metadata Properties?**

The @NgModule decorator identifies AppModule as a NgModule class.

The @NgModule takes a metadata object that tells Angular how to compile and launch the application.

The NgModule’s important metadata properties are as follows –

1. providers

2. declarations

3. imports

4. exports

5. entryComponents

6. bootstrap

7. schemas

8. id

The @NgModule class with the decorator and metadata properties -

@NgModule({

providers?: Provider[]

declarations?: Array<Type<any> | any[]>

imports?: Array<Type<any> | ModuleWithProviders | any[]>

exports?: Array<Type<any> | any[]>

entryComponents?: Array<Type<any> | any[]>

bootstrap?: Array<Type<any> | any[]>

schemas?: Array<SchemaMetadata | any[]>

id?: string

})

Let understand in detail about NgModule metadata is as follows-

Providers – A list of dependency injection (DI) providers and it defines the set of injectable objects that are available in the injector of this module.

Declarations - A list of declarable classes, components, directives, and pipes that belong to this module. The compiler throws an error if you try to declare the same class in multiple modules.

Imports - A list of modules and it used to import the supporting modules like FormsModule, RouterModule, CommonModule, or any other custom made feature module.

Exports - A list of declarable components, directives, pipes, and modules that an importing module can be used within a template of any component.

EntryComponents - A list of components that should be compiled when this module is defined. By default, an Angular app always has at least one entry component, the root component, AppComponent.

A bootstrapped component is an entry component that Angular loads into DOM during the application launch and other root components loaded dynamically into entry components.

Bootstrap – A list of components that are automatically bootstrapped and the listed components will be added automatically to entryComponents.

Schemas - Defines a schema that will allow any non-Angular elements and properties.

Id – The Id used to identify the modules in getModuleFactory. If left undefined, the NgModule will not be registered with getModuleFactory.

**What is the difference between declarations, providers, and import in NgModule?**

For more information, see the above question - What Are the @NgModule Metadata Properties?

**Why use multiple NgModules?**

Multiple NgModules provides some potential benefits.

Actually, the modules help you to organize an application into associative blocks of functionality.

First one is organizing an application code. If you are putting around 99 resource files in the default app module and see the happing.

And the second one is - It opens the possibility of lazy loading via the router.

**What Are the Purpose of @NgModule?**

The NgModule is used to simplify the ways you define and manage the dependencies in your applications and also you can consolidate different components and services into cohesive blocks of functionality.

The @NgModule metadata divided into three categories as follows.

1. Static

2. Runtime

3. Composability/Grouping

Static – It is compiler configuration and configured via the declarations array.

Runtime - It is injector configuration and configured via the provider’s array.

Composability/Grouping – Introducing NgModules together and configured via the imports and exports arrays.

The following is an example of specifying a NgModule metadata -

@NgModule({

// Static, This is the compiler configuration

declarations: [], //declarations is used for configure the selectors.

entryComponents: [], //entryComponents is used to generate the host factory.

//Runtime or injector configuration

providers: [], // providers is used for runtime injector configuration.

//Composability and Grouping

imports: [], // imports used for composing NgModules together.

exports: [] //A list of declarations components, directives, and pipes classes that an importing module can use.

})

**What Types of NgModules?**

There are four types of NgModules –

1. Features Module

2. Routing Module

3. Service Module

4. Widget Module

5. Shared Module

Features Module – The feature modules are NgModules for the purpose of organizing an application code.

Routing Module – The Routing is used to manage routes and also enables navigation from one view to another view as users perform application tasks.

Service Module – The modules that only contain services and providers. It provides utility services such as data access and messaging. The root AppModule is the only module that should import service modules. The HttpClientModule is a good example of a service.

Widget Module - The third party UI component libraries are widget modules.

Shared Module – The shared module allows you to organize your application code. You can put your commonly used components, directives, and pipes into the one module and use whenever required to this module.

**What Is Feature Modules?**

**What Are the Types of Feature Modules?**

The feature modules are modules that goal of organizing an application code. It also helps you partition the app into focused areas when you can do everything within the root module.

There are five types of feature modules which are the following-

1. Domain Feature Modules

2. Routed Feature Modules

3. Routing Modules

4. Service Feature Modules

5. Widget Feature Modules

Routed Feature Module - Routed feature modules are domain feature modules that components targets of router navigation routes.

A lazy-loaded routed feature module should not be imported by any module.

Routed feature modules do not export anything because their components never appear in the template of an external component.

Routing Module - A routing module provides routing configuration for another module and the routing module focus on the following.

1. Defines Routes

2. Adds Router Configuration to the module's imports

3. Adds service providers to the module's providers

4. A routing module doesn’t have its own declarations. The components, directives, and pipes are the responsibility of the feature module and not the routing module.

A routing module should only be imported by its companion module.

Service Feature Module - Service modules provide utility services and used to communicate with the server. The HttpClientModule is a great example of a service module.

The root AppModule is the single module that should import service modules.

Domain Feature Module - Domain feature modules deliver a user experience dedicated to a special application domain as like editing a customer and so on.

Widget Feature Module - A widget module makes components, directives, and pipes available to external modules.

The third party UI components and libraries are widget modules.

Import widget modules in any module whose component templates need the widgets.

**Why you use BrowserModule, CommonModule, FormsModule, RouterModule, and HttpClientModule?**

BrowserModule – The browser module is imported from @angular/platform-browser and it is used when you want to run your application in a browser.

CommonModule – The common module is imported from @angular/common and it is used when you want to use directives - NgIf, NgFor and so on.

FormsModule – The forms module is imported from @angular/forms and it is used when you build template driven forms.

RouterModule – The router module is imported from @angular/router and is used for routing RouterLink, forRoot, and forChild.

HttpClientModule –The HttpClientModule is imported from @angular/common/http and it used to initiate HTTP request and responses in angular apps. The HttpClient is more modern and easy to use the alternative of HTTP.

**What are the differences in NgModules and JavaScript Modules?**

NgModules vs. JavaScript Modules -

The NgModule is a TypeScript class decorated with @NgModule Decorator - is a fundamental feature of Angular.

JavaScript also has its own module system for managing collections of JavaScript objects. It is completely different from the NgModule system.

In JavaScript, each file is a module and all objects defined in the file belong to that module. The module declares some objects to be public by marking them with the export keyword.

Other JavaScript modules use import statements to access public objects from other modules.

The following is an example of specifying an export and import statements -

export class AppComponent {

//...

}

After export your class, you can import that file code in another file.

import { AppComponent } from './app.component';

Both the JavaScript and Angular use modules to organize applications code.

# What are the @NgModule Metadata Properties?

[Anil Singh](https://plus.google.com/116567432705413398625) [3:22 AM](https://www.code-sample.com/2018/04/ngmodule-metadata-properties.html)

The @NgModule decorator identifies AppModule as a NgModule class.

The [@NgModule](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html) takes a metadata object that tells Angular how to compile and launch the application.

The NgModule’s important metadata properties are as follows –

1. providers

2. declarations

3. imports

4. exports

5. entryComponents

6. bootstrap

7. schemas

8. id

The @NgModule class with the decorator and metadata properties -

@NgModule({

providers?: Provider[]

declarations?: Array<Type<any> | any[]>

imports?: Array<Type<any> | ModuleWithProviders | any[]>

exports?: Array<Type<any> | any[]>

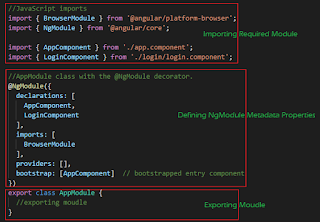
entryComponents?: Array<Type<any> | any[]>

bootstrap?: Array<Type<any> | any[]>

schemas?: Array<SchemaMetadata | any[]>

id?: string

})



Let understand in detail about NgModule metadata is as follows-

Providers – A list of dependency injection (DI) providers and it defines the set of injectable objects that are available in the injector of this module.

Declarations - A list of declarable classes, components, directives, and pipes that belong to this module. The compiler throws an error if you try to declare the same class in multiple modules.

Imports - A list of modules and it used to import the supporting modules like FormsModule, RouterModule, CommonModule, or any other custom made feature module.

Exports - A list of declarable components, directives, pipes, and modules that an importing module can be used within a template of any component.

EntryComponents - A list of components that should be compiled when this module is defined. By default, an Angular app always has at least one entry component, the root component, AppComponent.

A bootstrapped component is an entry component that Angular loads into DOM during the application launch and other root components loaded dynamically into entry components.

Bootstrap – A list of components that are automatically bootstrapped and the listed components will be added automatically to entryComponents.

Schemas - Defines a schema that will allow any non-Angular elements and properties.

Id – The Id used to identify the modules in getModuleFactory. If left undefined, the NgModule will not be registered with getModuleFactory.

# Why use multiple NgModules?

[Anil Singh](https://plus.google.com/116567432705413398625) [3:26 AM](https://www.code-sample.com/2018/04/why-use-multiple-ngmodules.html)

Multiple [NgModules](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html) provides some potential benefits.

Actually, the modules help you to organize an application into associative blocks of functionality.

First one is organizing an application code. If you are putting around 99 resource files in the default app module and see the happing.

And the second one is - It opens the possibility of lazy loading via the router.

# What Are the Purpose of @NgModule?

[Anil Singh](https://plus.google.com/116567432705413398625) [3:28 AM](https://www.code-sample.com/2018/04/purpose-of-ngmodule-angular-5-and-6.html)

What Are the Purpose of @NgModule?

The NgModule is used to simplify the ways you define and manage the dependencies in your applications and also you can consolidate different components and services into cohesive blocks of functionality.

The @NgModule metadata divided into three categories as follows.

1. Static

2. Runtime

3. Composability/Grouping

Static – It is compiler configuration and configured via the declarations array.

Runtime - It is injector configuration and configured via the provider’s array.

Composability/Grouping – Introducing [NgModules](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html) together and configured via the imports and exports arrays.

The following is an example of specifying a NgModule metadata -

@NgModule({

// Static, This is the compiler configuration

declarations: [], //declarations is used for configure the selectors.

entryComponents: [], //entryComponents is used to generate the host factory.

//Runtime or injector configuration

providers: [], // providers is used for runtime injector configuration.

//Composability and Grouping

imports: [], // imports used for composing NgModules together.

exports: [] //A list of declarations components, directives, and pipes classes that an importing module can use.

})

# What Types of @NgModules?

[Anil Singh](https://plus.google.com/116567432705413398625) [3:31 AM](https://www.code-sample.com/2018/04/types-of-ngmodules-angular-4-5-6.html)

What Types of NgModules?

There are four types of NgModules –

1. Features Module

2. Routing Module

3. Service Module

4. Widget Module

5. Shared Module

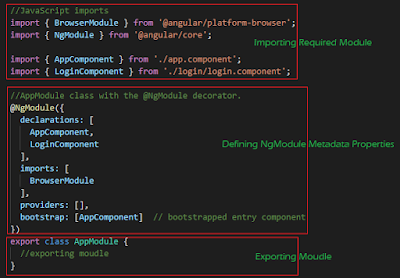
Features Module – The feature modules are [NgModules](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html) for the purpose of organizing an application code.

Routing Module – The Routing is used to manage routes and also enables navigation from one view to another view as users perform application tasks.

Service Module – The modules that only contain services and providers. It provides utility services such as data access and messaging. The root AppModule is the only module that should import service modules. The[HttpClientModule](https://www.code-sample.com/2018/04/ngmodule-metadata-properties.html) is a good example of a service.

Widget Module - The third party UI component libraries are widget modules.

Shared Module – The shared module allows you to organize your application code. You can put your commonly used components, directives, and pipes into the one module and use whenever required to this module.



# What Is Feature Modules? What Are the Types of Feature Modules?

[Anil Singh](https://plus.google.com/116567432705413398625) [3:33 AM](https://www.code-sample.com/2018/04/feature-modules-angular-4-5-6.html)

What Is Feature Modules?

What Are the Types of Feature Modules?

The feature modules are modules that goal of organizing an application code. It also helps you partition the app into focused areas when you can do everything within the root module.

There are five types of feature modules which are the following-

1. Domain Feature Modules

2. Routed Feature Modules

3. Routing Modules

4. Service Feature Modules

5. Widget Feature Modules

Routed Feature Module - Routed feature modules are domain feature modules that components targets of router navigation routes.

A lazy-loaded routed feature module should not be imported by any module.

Routed feature modules do not export anything because their components never appear in the template of an external component.

Routing Module - A routing module provides [routing configuration](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html) for another module and the routing module focus on the following.

1. Defines Routes

2. Adds Router Configuration to the module's imports

3. Adds service providers to the module's providers

4. A routing module doesn’t have its own declarations. The components, directives, and pipes are the responsibility of the feature module and not the routing module.

A routing module should only be imported by its companion module.

Service Feature Module - Service modules provide utility services and used to communicate with the server. The HttpClientModule is a great example of a service module.

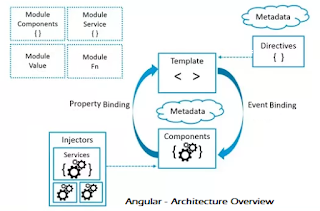
The root AppModule is the single module that should import service modules.

Domain Feature Module - Domain feature modules deliver a user experience dedicated to a special application domain as like editing a customer and so on.

Widget Feature Module - A widget module makes components, directives, and pipes available to external modules.

The third party UI components and libraries are widget modules.

Import widget modules in any module whose component templates need the widgets.



# Why you use BrowserModule, CommonModule, FormsModule, RouterModule, and HttpClientModule?

[Anil Singh](https://plus.google.com/116567432705413398625) [3:38 AM](https://www.code-sample.com/2018/04/browsermodule-commonmodule-formsmodule.html)

Why you use BrowserModule, CommonModule, FormsModule, RouterModule, and HttpClientModule?

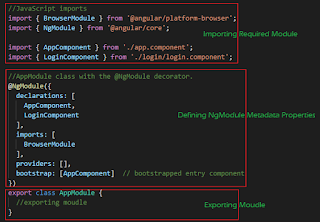
BrowserModule – The browser module is imported from @angular/platform-browser and it is used when you want to run your application in a browser.

CommonModule – The common module is imported from @[angular/common](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html) and it is used when you want to use directives - NgIf, NgFor and so on.

FormsModule – The forms module is imported from @angular/forms and it is used when you build template driven forms.

RouterModule – The router module is imported from @angular/router and is used for routing RouterLink, forRoot, and forChild.

HttpClientModule –The HttpClientModule is imported from @angular/common/http and it used to initiate HTTP request and responses in angular apps. The HttpClient is more modern and easy to use the alternative of HTTP.



# What are the differences in NgModules and JavaScript Modules?

[Anil Singh](https://plus.google.com/116567432705413398625) [3:44 AM](https://www.code-sample.com/2018/04/ngmodules-vs-javascript-modules.html)

What are the differences in NgModules and JavaScript Modules?

NgModules vs. JavaScript Modules -

The NgModule is a TypeScript class decorated with [@NgModule Decorator](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html) - is a fundamental feature of Angular.

JavaScript also has its own module system for managing collections of JavaScript objects. It is completely different from the NgModule system.

In JavaScript, each file is a module and all objects defined in the file belong to that module. The module declares some objects to be public by marking them with the export keyword.

Other JavaScript modules use import statements to access public objects from other modules.

The following is an example of specifying an export and import statements -

export class AppComponent {

//...

}

After export your class, you can import that file code in another file.

import { AppComponent } from './app.component';

Both the JavaScript and Angular use modules to organize applications code.

# What's Angular Elements? - Angular 6 Most Appreciated Features!

[Anil Singh](https://plus.google.com/116567432705413398625) [3:08 AM](https://www.code-sample.com/2018/05/angular-6-7-elements.html)

**Angular Elements** resolve the problems of code reuse across multiple frameworks and provides a great way to use Angular components in non-Angular environments.

So, **where would we use Angular Elements?** **Rob Wormald** covered three main areas with examples in his[**ng-conf 2018 talkon Angular Elements**](https://www.youtube.com/watch?v=Z1gLFPLVJjY)**.**

The list of browsers support for custom Angular elements -

1. Chrome,

2. Opera,

3. Safari,

4. Firefox,

5. Edge

***What's Angular Elements?***

Angular Element is a package which is part of the Angular framework- @angular/elements and it was introduced in[Angular 6](https://www.code-sample.com/2018/01/angular-6-interview-questions-and.html) and hopefully, this will improve with [Angular 7](https://www.code-sample.com/2018/04/angular-7-interview-questions-and.html) or higher versions.

Angular Elements was the brainchild of Angular’s and it one of the most anticipated features of Angular 6 release.

Angular 6 will be the first release that fully supports Angular elements.

Angular elements will give you the ability to use your Angular components in other environments like a jQuery app or [Vue.js](https://www.code-sample.com/2017/11/vuejs-2-interview-questions-and-answers.html) app or anything else.

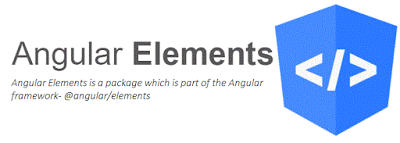
It is very useful, especially when you working with dynamically loaded HTML code.

It also offers functionality that allows you to convert a normal Angular component to a native web component.

Angular Elements give you an easy way to implement a web standard. There is no smoke and mirrors.

Custom Elements, let you create custom tags in a framework-agnostic way. They let you reuse your Angular components in any webpage. Yes, you can embed Angular Elements inside a [React.js](https://www.code-sample.com/2018/03/reactjs-interview-questions-and-answers.html) or [Vue.js](https://www.code-sample.com/2017/11/vuejs-2-interview-questions-and-answers.html) page without any knowledge of Angular.  
  
[How to Create a Custom Angular Elements?](https://www.code-sample.com/2018/07/create-custom-element-angular-6.html)

Custom Elements remove the need to rewrite a widget every-time a new framework pops up.



**How do Angular Elements work?**

1. They are self-Bootstrapping.

2. They actually host the Angular Component inside a Custom Element.

3. They’re a bridge between the DOM APIs and Angular Components.

4. Anyone can use your component without having to know Angular.

**What are the *features of Angular Elements?***

*The features of Angular Elements* -

1. Self-bootstrapping

2. Hosts an Angular Component inside a Custom Elements

3. A bridge between DOM and Angular Components APIs

**What are *Advantages of Angular Elements?***

*Advantages of Angular Elements* -

1. Reusability - Reuse components in across your apps

2. Widgets – You can use Angular components in other environments.

3. CMS Pages

4. And many more.

**How to Install Angular Elements?**

To add support for Angular Elements use the Angular CLI command - **ng add**.

ng add @angular/elements

**What Is “ng add” for Angular Elements?**

The “ng add” command is built on schematics. When you run ng add @angular/elements the CLI scans your project and updates your code to support Angular Elements.

It’s automatic. There’s no manual configuration required.

After successfully added in your project -

Two dependencies are added to the package.json which is the following.

1. @angular/elements: “^6.0.1

2. document-register-element: ^1.7.2

It’s automatic. There’s no manual configuration required.

**How to create an Angular Element?**

**Can you show me an example of an Angular Element?**

Simply you can create a normal Angular component with inputs & outputs and import this component inside your angular module with helps of @angular/elements.

Now, your elements are ready to use inside a simple HTML page-

<my-custom-elmet message="This is my custom element "></my-custom-elmet>

To learn more, you can watch ***Rob Wormald’s* talk** about [Angular Elements](https://www.youtube.com/watch?v=Z1gLFPLVJjY) at **ng-conf**.

***What's Angular Elements?***

Angular Element is a package which is part of the Angular framework- @angular/elements and it was introduced in [Angular 6](https://www.code-sample.com/2018/01/angular-6-interview-questions-and.html) and hopefully, this will improve with [Angular 7](https://www.code-sample.com/2018/04/angular-7-interview-questions-and.html) or higher versions.

Angular Elements was the brainchild of Angular’s and it one of the most anticipated features of Angular 6 release. [Explore in detail about Angular Elements …](https://www.code-sample.com/2018/05/angular-6-7-elements.html)

***Now come to – “How to Create a Custom Angular Elements?”***

There are following steps involving to Creating Custom Elements.

1. Install Angular 6 platforms and Angular CLI, if not installed.

2. Create a new project using Angular CLI.

3. Once successfully created your new project, change the directory of your created.

4. Install Angular elements.

5. Install web components custom elements.

6. Create your reusable component and it will be used as an Angular Element.

7. Register your comments which are required like reusable components, Injector, and createCustomElement.

8. Use the created custom element in your app.

9. Result

**Steps 1** - Install Angular 6, if not install.

**Steps 2** – Create a new project using Angular CLI

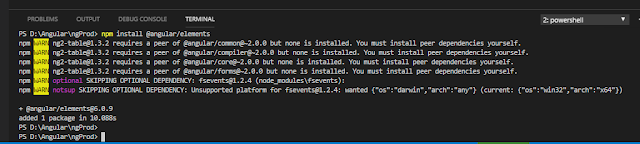
ng new ngprod

In the above command, the “ngprod” is the name of my project which I used to create this demo.

**Steps 3** - Once successfully created your new project, change the directory of your created project just like - ngprod.

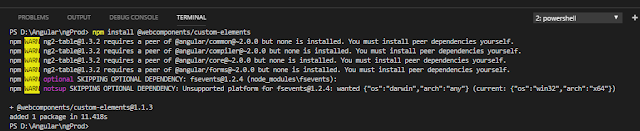
**Steps 4** - Now install Angular elements using the below CLI command.

npm install @angular/elements



**Steps 5** - Once done the web components custom elements, install

npm install @webcomponents/custom-elements



This will work for older browsers and also we will need to open and add below two lines in your project file - polyfills.ts.

\* APPLICATION IMPORTS \*/

import '@webcomponents/custom-elements/src/native-shim';

import '[@webcomponents/custom-elements/custom-elements.min](mailto:@webcomponents/custom-elements/custom-elements.min)';

**Steps 6** - Create your reusable component and it will be used as an Angular Element.

In this example, I have created a custom element for displays the list of active users (active-user.component.ts and active-user.component.html) and my created component contains the list of activeUsers, applications access user name, and title.

**active-user.component.ts** -

import { Component, OnInit, Input } from '@angular/core';

@Component({

selector: 'app-activeuser',

templateUrl: './active-user.component.html',

styleUrls: ['./active-user.component.css']

})

export class ActiveUserComponent implements OnInit {

public activeUsers :any;

public title :any;

@Input() user: string;

constructor() {

this.title ='Angular - Create Custom Elements';

}

ngOnInit() {

this.activeUsers = [

{"name" :"Anil Singh"},

{"name" :"Alok Singh"},

{"name" :"Dilip Singh"},

{"name" :"Sunil Singh"},

{"name" :"Aradhya Singh"},

{"name" :"Reena Singh"}

];

}

}

And **active-user.component.html** -

<h1>{{title}}</h1>

<h2>You are welcome, {{user}}!</h2>

<h4>Active Users - </h4>

<div>

<table \*ngFor="let user of activeUsers; let i = index;" width="40%">

<tr><td width="5%">#{{i +1}}</td><td width="30%">{{user.name}}</td></tr>

</table>

</div>

**Steps 7** - Register the active-user.component in the app.module.ts.

import { ActiveUserComponent } from './active-user/active-user.component';

And comment the line - bootstrap ( //bootstrap: [AppComponent]) from the NgModule class.

And also added the line - entryComponents (entryComponents: [ActiveUserComponent]) in the NgModule.

See in the below code.

import { BrowserModule } from '@angular/platform-browser';

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

import { FormsModule } from '@angular/forms';

import { RouterModule, Routes } from '@angular/router';

import { AppComponent } from './app.component';

import { UserComponent } from './user/user.component';

import { UserListComponent } from './user-list/user-list.component';

import {DataTableModule} from "angular-6-datatable";

import { ActiveUserComponent } from './active-user/active-user.component';

import { createCustomElement } from '@angular/elements';

@NgModule({

declarations: [

AppComponent,

UserComponent,

UserListComponent,

ActiveUserComponent

],

imports: [

BrowserModule,

DataTableModule,

FormsModule,

RouterModule.forRoot([

{ path: 'user-link', component: UserComponent },

{ path: 'user-list', component: UserListComponent }

])

],

providers: [],

// bootstrap: [AppComponent],

bootstrap: [],

entryComponents: [ActiveUserComponent]

})

export class AppModule {

}

**Steps 8** - Register the Injector and createCustomElement in the app.module.ts.

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

import { createCustomElement } from '@angular/elements';

After register the required modules in the app.module.ts, we will need to create custom element in the AppModule.

See in the below code.

export class AppModule {

//An instance of the element is created or upgraded.

constructor(private injector: Injector) {

//The customElements is used for defining a custom element globaly.

const customElement = createCustomElement(ActiveUserComponent, { injector });

//Defining a new element

customElements.define('app-activeuser', customElement);

}

ngDoBootstrap() {}

}

And the app.module.ts looks like –

import { BrowserModule } from '@angular/platform-browser';

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

import { FormsModule } from '@angular/forms';

import { RouterModule, Routes } from '@angular/router';

import { AppComponent } from './app.component';

import { UserComponent } from './user/user.component';

import { UserListComponent } from './user-list/user-list.component';

import {DataTableModule} from "angular-6-datatable";

import { ActiveUserComponent } from './active-user/active-user.component';

import { createCustomElement } from '@angular/elements';

@NgModule({

declarations: [

AppComponent,

UserComponent,

UserListComponent,

ActiveUserComponent

],

imports: [

BrowserModule,

DataTableModule,

FormsModule,

RouterModule.forRoot([

{ path: 'user-link', component: UserComponent },

{ path: 'user-list', component: UserListComponent }

])

],

providers: [],

// bootstrap: [AppComponent],

bootstrap: [],

entryComponents: [ActiveUserComponent]

})

export class AppModule {

//An instance of the element is created or upgraded.

constructor(private injector: Injector) {

//The customElements is used for defining a custom element globaly.

const customElement = createCustomElement(ActiveUserComponent, { injector });

//Defining a new element

customElements.define('app-activeuser', customElement);

}

ngDoBootstrap() {}

}

**Steps 9** - Use the created custom element in your app.

<body>

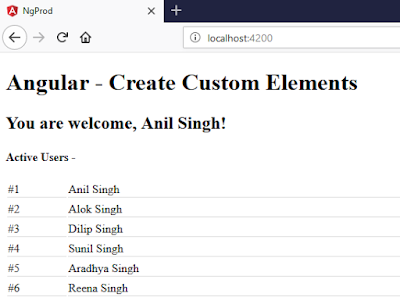
<!-- <app-root></app-root> -->

<app-activeuser user="Anil Singh"></app-activeuser>

</body>

**Note** - I am using created custom element in the - index.html page.

**Result looks like** -



**Conclusion** –

An Angular Custom element gives you a new tool for defining new HTML tags in the browsers and creating reusable components.

# Modular View Engine Architecture - Angular 6

[Anil Singh](https://plus.google.com/116567432705413398625) [10:06 PM](https://www.code-sample.com/2018/02/angular-6-modular-view-engine.html)

Angular View Engine could be implemented, such as -

1. Taking a **multi-threaded** approach to rendering

2. Generating **Web-Assembly** code

3. Generating **SPIR-V** code to exploit the work of the important W3C gpuweb working group

4. Generating Verilog for a **FPGA**

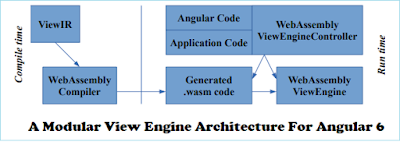
5. Deeper integration with **native platforms**

There are three possible approaches built on [**Angular 6**](https://www.code-sample.com/2018/01/angular-6-interview-questions-and.html) **to facilitate “modular view engines**” and it can be -

ü do nothing

ü do a little

ü do a lot



**Do Nothing**- No changes are needed to be carried out to the existing Angular 6 beta code.

**Do a Little** - Making small changes to **Compiler CLI** so that the shared functionality between the different view compilers can be defined once.

**Do a Lot** - Realizing that supporting multiple view engines is going to be important in the future and investing now the necessary engineering effort to structure on-going development.

I hope you are enjoying with this post! Please share with you friends. Thank you so much!

**What Are Angular Directives?**

Angular Directive is a TypeScript class which is declared as a [@directive decorator](https://www.code-sample.com/2018/05/angular-5-6-7-directives-decorator.html).

The directives allow you to attach behavior to DOM elements and the @directive decorator provide you an additional metadata that determines how directives should be processed, instantiated, and used at run-time.

**What Are decorators?**

The Decorators are functions that modify JavaScript classes and it also used for attaching metadata to classes.

Directive decorator and metadata Properties -

@Directive({

selector?: string

inputs?: string[]

outputs?: string[]

host?: {...}

providers?: Provider[]

exportAs?: string

queries?: {...}

})

**Selector** – It is a CSS selector that tells Angular to create an instance of this component wherever it finds the corresponding tag in template HTML.

For example, it is – <app-login></app-login>

CSS selector also triggers the instantiation of a directive.

The selector may be declared by element name, class name, attribute name, and attribute name & value.

Suppose we have a directive with an *<input type="checkbox">* selector and the HTML looks like this.

<form>

<label>Name -</label> <input type="text">

<label>Are you agree? </label> <input type="checkbox">

<form>

The directive will only be instantiated on the *<input type="checkbox">* element.

**Inputs**– The list of class property names to data-bind as component inputs

**Outputs** - The list of class property names that expose output events that others can subscribe too

**Host**– These properties use to map the class property to host element bindings for properties, events, actions, and attributes.

The host looks like this.

@Directive({

selector: 'button',

host: {'(click)': 'onClick($event.target)'}

})

**Providers** - list of providers available to this component and its children

**Queries**– To configure queries that can be injected into the component

**We have 3 types of Directives in Angular** -

1. Component

2. Attribute Directives

3. Structural Directives

**Components** - The [component is a directive](https://www.code-sample.com/2018/04/what-are-components-in-angular-6-54-and.html) with their own templates and it is responsible for how a component should be processed, instantiated and used at run-time.

**Structural Directives** - The structural directive is a directive and it is responsible for change the DOM layout by adding, removing, and manipulating elements.

The most of the common built-in structural directives are NgIf, NgFor, and NgSwitch.

**Attribute Directives** - The Attribute directive is a directive and it is responsible for change the behavior of a specified element or component.

**What are the differences between @Component and @Directive?**

The components are used, when you want to create new elements in the DOM with their own HTML template.

The attribute directives are used, when you want to change or update the existing elements in the DOM.

**How to Create Custom Directives?**

Let's start to create a simple directive.

I assuming you have installed the Angular CLI and all the necessary configurations are running in your app. Now, go to your project directory and execute the below CLI command for creating your custom directive –

ng g directive myCustom

After execute the above CLI command, created two files in the project - src/app folder

1. src/app/my-custom.directive.spec.ts

2. src/app/my-custom.directive.ts

And update files reference automatically in your project module – “src/app/app.module.ts”

Lest see in the code-sample, how it look like-

my-custom.directive.ts –

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

@Directive({

selector: '[appMyCustom]'

})

export class MyCustomDirective {

constructor() { }

}

And app.module.ts –

import { MyCustomDirective } from './my-custom.directive'

//AppModule class with @NgModule decorator

@NgModule({

//Static, this is the compiler configuration

//declarations is used for configure the selectors.

declarations: [

AppComponent,

MyCustomDirective,

],

//Composability and Grouping

//imports used for composing NgModules together.

imports: [

BrowserModule

],

//Runtime or injector configuration

//providers is used for runtime injector configuration.

providers: [],

//bootstrapped entry component

bootstrap: [AppComponent]

})

export class AppModule { }

# What are the differences between @Component and @Directive?

[Anil Singh](https://plus.google.com/116567432705413398625) [11:01 PM](https://www.code-sample.com/2018/05/angular-5-6-7-component-vs-directive.html)

What are the differences between @Component and @Directive?

The components are used, when you want to create new elements in the DOM with their own HTML template.

The attribute directives are used, when you want to change or update the existing elements in the DOM.

**We have 3 types of Directives in Angular** -

1. Component

2. Attribute Directives

3. Structural Directives

**Components** - The [component is a directive](https://www.code-sample.com/2018/04/what-are-components-in-angular-6-54-and.html) with their own templates and it is responsible for how a component should be processed, instantiated and used at run-time.

**Structural Directives** - The structural directive is a directive and it is responsible for change the DOM layout by adding, removing, and manipulating elements.

The most of the common built-in structural directives are NgIf, NgFor, and NgSwitch.

**Attribute Directives** - The Attribute directive is a directive and it is responsible for change the behavior of a specified element or component.

# What Are Angular Decorators?

[Anil Singh](https://plus.google.com/116567432705413398625) [11:10 PM](https://www.code-sample.com/2018/05/angular-5-6-7-decorators.html)

What Are decorators?

The Decorators are functions that modify JavaScript classes and it also used for attaching metadata to classes.

Directive decorator and metadata Properties -

@Directive({

selector?: string

inputs?: string[]

outputs?: string[]

host?: {...}

providers?: Provider[]

exportAs?: string

queries?: {...}

})

**Selector** – It is a CSS selector that tells Angular to create an instance of this component wherever it finds the corresponding tag in template HTML.

For example, it is – <app-login></app-login>

CSS selector also triggers the instantiation of a directive.

The selector may be declared by element name, class name, attribute name, and attribute name & value.

Suppose we have a directive with an *<input type="checkbox">* selector and the HTML looks like this.

<form>

<label>Name -</label> <input type="text">

<label>Are you agree? </label> <input type="checkbox">

<form>

The directive will only be instantiated on the *<input type="checkbox">* element.

**Inputs**– The list of class property names to data-bind as component inputs

**Outputs** - The list of class property names that expose output events that others can subscribe too

**Host**– These properties use to map the class property to host element bindings for properties, events, actions, and attributes.

The host looks like this.

@Directive({

selector: 'button',

host: {'(click)': 'onClick($event.target)'}

})

**Providers** - list of providers available to this component and its children

**Queries**– To configure queries that can be injected into the component

For more detail kindly [**refer the link....**](https://www.code-sample.com/2018/05/angular-5-6-7-directives-decorator.html)

# How to Create Custom Directives?

[Anil Singh](https://plus.google.com/116567432705413398625) [11:04 PM](https://www.code-sample.com/2018/05/angular-6-7-create-custom-directives.html)

Create a Custom Directives -

Let's start to create a simple directive. For more detail kindly [**refer the link....**](https://www.code-sample.com/2018/05/angular-5-6-7-directives-decorator.html)

I assuming you have installed the Angular CLI and all the necessary configurations are running in your app. Now, go to your project directory and execute the below CLI command for creating your custom directive –

ng g directive myCustom

After execute the above CLI command, created two files in the project - src/app folder

1. src/app/my-custom.directive.spec.ts

2. src/app/my-custom.directive.ts

And update files reference automatically in your project module – “src/app/app.module.ts”

Lest see in the code-sample, how it look like-

my-custom.directive.ts –

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

@Directive({

selector: '[appMyCustom]'

})

export class MyCustomDirective {

constructor() { }

}

And app.module.ts –

import { MyCustomDirective } from './my-custom.directive'

//AppModule class with @NgModule decorator

@NgModule({

//Static, this is the compiler configuration

//declarations is used for configure the selectors.

declarations: [

AppComponent,

MyCustomDirective,

],

//Composability and Grouping

//imports used for composing NgModules together.

imports: [

BrowserModule

],

//Runtime or injector configuration

//providers is used for runtime injector configuration.

providers: [],

//bootstrapped entry component

bootstrap: [AppComponent]

})

export class AppModule { }

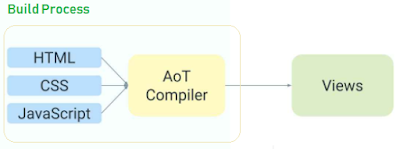
# Ahead-of-Time (AOT) Compiler - What Is the Angular Compiler?

[Anil Singh](https://plus.google.com/116567432705413398625) [1:26 AM](https://www.code-sample.com/2018/04/angular-ahead-of-time-aot-compiler.html)

**What Is the Angular Compiler?**

The **Angular** compiler converts our applications code ([**HTML**](https://www.code-sample.com/2015/05/html5-interview-questions-and-answers.html) **and** [**TypeScript**](https://www.code-sample.com/2017/06/typescript-interview-questions-and.html)**) into JavaScript** code before browser downloads and runs that code.

The @[**NgModule**](https://www.code-sample.com/2017/04/angular-2-ngmodel-root-export-module.html) **metadata plays an important role in guiding the compilation process and also tells the compiler what components to compile for this module and how to link this module with other modules.**



The Angular offers two ways to compile our application code-

1. **Just-in-Time** (**JIT**) - JIT compiles our app in the browser at runtime (compiles before running).

2. **Ahead-of-Time** (**AOT**) - AOT compiles our app at build-time (compiles while running).

The **JIT** compilation is the default when we run the **build** or **serve** CLI commands -

ng build

ng serve

The **AOT** compilation, we append the --**aot** flags to **build** or **serve** CLI commands -

ng build --aot

ng serve --aot

**Why we need Compilation in Angular?**

We need compilation for achieving a higher level of efficiency, performance improvements, faster rendering and also sometimes detect template errors earlier in our Angular applications.

**Why Compile with AOT?**

1. Faster Rendering

2. Asynchronous Requests

3. Detect template errors earlier

4. Smaller Angular frameworks download size

5. Better Security

**What Is the difference between JIT compiler and AOT compiler?**

**JIT (Just-in-Time) -**

1. JIT compiles our app in the browser at runtime.

2. Compiles before running

3. Each file compiled separately

4. No need to build after changing our app code and it automatically reflects the changes in your browser page

5. Highly secure

6. Very suitable for local development

**AOT (Ahead-of-Time) -**

1. AOT compiles our app code at build time.

2. Compiles while running

3. Compiled by the machine itself, via the command line (Faster)

4. All code compiled together, inlining HTML/CSS in the scripts

5. Highly secure

6. Very suitable for production builds

**Angular Compiler Class –**

class Compiler {

// Compiles the given NgModule and all of its components

compileModuleSync<T>(moduleType: Type<T>): NgModuleFactory<T>

//Compiles the given NgModule and all of its components

compileModuleAsync<T>(moduleType: Type<T>): Promise<NgModuleFactory<T>>

//creates ComponentFactories for all components

compileModuleAndAllComponentsSync<T>(moduleType: Type<T>): ModuleWithComponentFactories<T>

////creates ComponentFactories for all components

compileModuleAndAllComponentsAsync<T>(moduleType: Type<T>):Promise<ModuleWithComponentFactories<T>>

//Clears all caches.

clearCache(): void

//Clears the cache for the given component/ngModule.

clearCacheFor(type: Type<any>)

}

I hope you are enjoying with this post! Please share with you friends. Thank you!!

# AOT vs JIT Compiler - What Is the difference between JIT compiler and AOT compiler?

[Anil Singh](https://plus.google.com/116567432705413398625) [2:07 AM](https://www.code-sample.com/2018/04/aot-vs-jit-compiler-angular-4-and-5.html)

*What Is the Angular Compiler?*

The [**Angular** **compiler**](https://www.code-sample.com/2018/04/angular-ahead-of-time-aot-compiler.html)converts our applications code ([**HTML**](https://www.code-sample.com/2015/05/html5-interview-questions-and-answers.html) **and** [**TypeScript**](https://www.code-sample.com/2017/06/typescript-interview-questions-and.html)**) into JavaScript** code before browser downloads and runs that code.

**JIT (Just-in-Time) -**

1. JIT compiles our app in the browser at run-time.

2. Compiles before running

3. Each file compiled separately

4. No need to build after changing our app code and it automatically reflects the changes in your browser page

5. Highly secure

6. Very suitable for local development

**AOT (Ahead-of-Time) -**

1. AOT compiles our app code at build time.

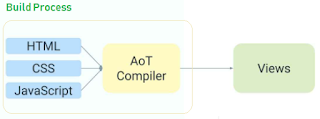
2. Compiles while running

3. Compiled by the machine itself, via the command line (Faster)

4. All code compiled together, inlining HTML/CSS in the scripts

5. Highly secure

6. Very suitable for production builds



I hope you are enjoying with this post! Please share with you friends. Thank you!!

# Angular 6 Ivy Renderer

[Anil Singh](https://plus.google.com/116567432705413398625) [10:51 PM](https://www.code-sample.com/2018/05/angular-6-7-ivy-renderer.html)

What Is Ivy Renderer?

The new Ivy renders and it’s not stable for now and it’s only in beta version. It will stable in future for production.

Ivy Renderer is new rendering engine which is designed to be backward compatible with existing render and focused to improve the speed of rendering and it optimizes the size of the final package.

The main goal of Ivy render is to speed up its loading time and reduce the bundle size of your applications. Also for uses a different approach for rendering Angular components.

For Angular, this will not be default renderer, but you can manually enable it in compiler options.

New Ivy engine in Angular 6 -

1. Smaller builds

2. Faster rebuild times

3. Faster development

4. A simpler, more hack-able pipeline

5. Human readable code

For more detail kindly refer the [**Angular 6**](https://www.code-sample.com/2018/01/angular-6-interview-questions-and.html)

# Angular 6 Bazel Compiler - What Is Bazel Compiler?

[Anil Singh](https://plus.google.com/116567432705413398625) [10:54 PM](https://www.code-sample.com/2018/05/angular-6-7-bazel-compiler.html)

What Is Bazel Compiler?

What Angular is doing with Bazel Compiler?

The Bazel Complier is a build system used for nearly all software built at Google.

From Angular 6 release, will start having the Bazel compiler support and when you compile the code with Bazel Compiler, you will recompile entire code base, but it compiles only with necessary code.

The Bazel Complier uses advanced local and distributed caching, optimized dependency analysis and parallel execution.

Bazel allows us to break an application into distinct build units. In Angular, build units are defined at the NgModule level.

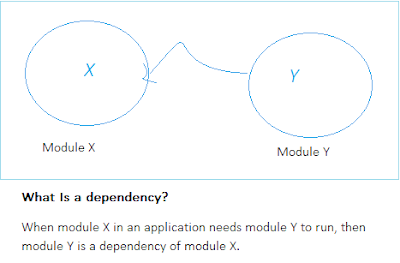
This means the scope of a build can be as granular as a single [**NgModule**](https://www.code-sample.com/2018/04/modules-ngmodule-angular-4-5-6.html)**. If a change is internal to an NgModule, only that module needs to be rebuilt.**

Angular, Angular Universal, NgRx, and Tsickle all switched to Bazel as the build tool, and ship Bazel-built artefacts to npm.

For more detail kindly refer the [**Angular 6**](https://www.code-sample.com/2018/01/angular-6-interview-questions-and.html)

**What Is a Dependency?**

When module X in an application needs module Y to run, then module Y is a dependency of module X.



***What Is Dependency Injection (DI)?***

Dependency Injection is a powerful pattern for managing code dependencies. DI is a way to create objects that depend upon other objects.

Angular has its own DI framework pattern, and you really can't build an Angular application without Dependency injection (DI).

A DI system supplies the dependent objects when it creates an instance of an object.

Let us take an example of a CAR. The CAR consists of following things -

1. Wheel

2. Headlight

3. Outer door

4. Inner door

5. Glass

6. Window

7. Fuel level sensor

So to complete the CAR, we need those eight and so many things.

In this example, we need to require total seven and many more classes to build a fully functional CAR.

1. Car class

2. Wheel class

3. Headlight class

4. Outer door class

5. Inner door class

6. Glass class

7. Window class

8. Fuel level sensor class

***Let’s see what happen, without Dependency Injection (DI)*** –

To complete the CAR class, we need to import all eight classes here and make one fully functional CAR.

Now, here we have created eight classes instance in the constructor of CAR class.

Note that, the CAR class is totally dependent on these eight classes. Otherwise, it will not complete the CAR.

We are creating the instances in the CAR constructor. So Wheel, Headlight, Outer door, Inner door, Glass, Window, and Fuel are not decoupled from the CAR class.

***Let’s see what happen, with Dependency Injection (DI)*** –

If we are using Dependency Injection then, we do not need to create the instances in the constructor.

First, we need to provide all the dependencies to the “*app.module.ts*” class -

import { BrowserModule } from '@angular/platform-browser';

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

//Import App Component

import { AppComponent } from './app.component';

//Import CAR classes.

import {Wheel} from './car/wheel';

import {Headlight} from './car/headlight';

import {Glass} from './car/glass';

import {InnerDoor} from './car/inner-door';

import {OuterDoor} from './car/outer-door';

import {Window} from './car/window';

import {FuelLevel} from './car/fuel-level';

//AppModule class with @NgModule decorator.

@NgModule({

//Static, This is the compiler configuration

//declarations is used for configure the selectors.

declarations: [

AppComponent,

],

//Composability and Grouping

// imports used for composing NgModules together.

imports: [

BrowserModule

],

//Runtime or injector configuration

//providers is used for runtime injector configuration.

providers: [Wheel, Headlight, Glass, InnerDoor, OuterDoor, Window, FuelLevel],

//bootstrapped entry component

bootstrap: [AppComponent]

})

export class AppModule { }

In providers array, we need to provide all eight dependencies.

Then, In the CAR class, inject those dependencies into CAR constructor –

import {Wheel} from './car/wheel';

import {Headlight} from './car/headlight';

import {Glass} from './car/glass';

import {InnerDoor} from './car/inner-door';

import {OuterDoor} from './car/outer-door';

import {Window} from './car/window';

import {FuelLevel} from './car/fuel-level';

export class Car {

constructor(public wheel: Wheel,

public headlight: Headlight,

public glass: Glass,

public innerdoor: InnerDoor,

public outerdoor: OuterDoor,

public window: Window,

public fuellevel: FuelLevel) {}

}

When CAR instance is created at that time, also all the other instances of other classes are also created.

***What Is Dependency Injection pattern?***

DI is an application design pattern and you really cannot build an Angular application without dependency injection (DI).

For more detail, refer the above questions.

***What Is Injectors?***

A service is just a class in Angular until you register with an Angular dependency injector.

The injector is responsible for creating angular service instances and injecting them into classes.

You rarely create an injector yourself and Angular creates automatically during the bootstrap process.

Angular doesn't know automatically how you want to create instances of your services or injector. You must configure it by specifying providers for every service. Actually, providers tell the injector how to create the service and without a provider not able to create the service.

Bootstrap defines the components that should be bootstrapped when this module is bootstrapped. The components listed here will automatically be added to entryComponents.

Explore in detail by using above questions, [*What Is an entryComponents?*](https://www.code-sample.com/2018/04/angular-6-5-4-entry-component.html)

***What Are @Injectable providers?***

The @Injectable decorator identifies services and other classes that are intended to be injected. It can also be used to configure a provider for those services.

To inject the service into a component, Angular provides an Injector decorator: @Injectable().

A provider defines the set of injectable objects that are available in the injector of this module.

The @Injectable decorator marks a class as available to an injector for instantiation. An injector reports an error when trying to instantiate a class that is not marked as @Injectable.

Injectors are also responsible for instantiating components. At the run-time the injectors can read class metadata in the JavaScript code and use the constructor parameter type information to determine what things to inject.

Injectable decorator and metadata -

@Injectable({

providedIn?: Type<any> | 'root' | null

factory: () => any

})

To inject the service into a component, Angular provides an Injector decorator: @Injectable().

Here we configure a provider for CustomerService using the @Injectable decorator on the class.

We have the following steps to create a Service-

1. Create the service class

2. Define the metadata with a decorator

3. Import what we need.

In the above example, providedIn tells Angular that the root injector is responsible for creating an instance of the CustomerService.

The Angular CLI sets up provider automatically when you generating a new service.

***Why @Inject()?***

The @Inject is a special technique for letting Angular knows that a parameter must be injected.

Inject decorator and metadata-

@Inject({

token: any

})

When @Inject () is not present, Injector will use the type annotation of the parameter.

import { Component, OnInit, Inject } from '@angular/core';

import { HttpClient } from '@angular/common/http';

@Component({

selector: 'app-customer',

templateUrl: './customer.component.html',

styleUrls: ['./customer.component.css']

})

export class CustomerComponent implements OnInit {

constructor(@Inject(HttpClient) private http) {

// use this.http which is the Http provider.

}

ngOnInit(){ }

}

At this point, @Inject is a manual way of specifying this lookup token, followed by the lowercase http argument to tell Angular what to assign it against.

***What Is Hierarchical Dependency Injectors?***

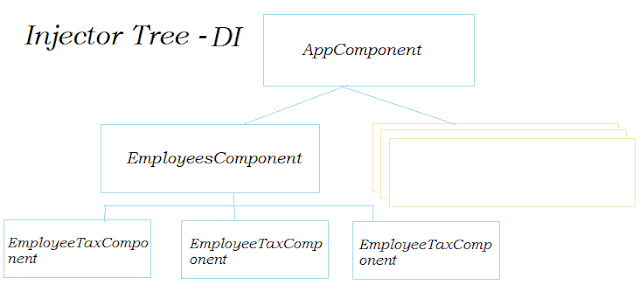
Angular has a Hierarchical Dependency Injection system. There is actually a tree of injectors that parallel an application's component tree. You can reconfigure the injectors at any level of that component tree.

***What Is Injector tree?***

In the Dependency Injection guide, you learned how to configure a dependency injector and how to retrieve dependencies where you need them.

An application may have multiple injectors. An Angular application is a tree of components. Each component instance has its own injector. The tree of components parallels the tree of injectors.

Three level component tree –



***What Is Injector bubbling?***

[***What Is @Injectable, @NgModule or @Component?***](https://www.code-sample.com/2017/08/angular-5-interview-questions-and.html)

Refer the above questions.

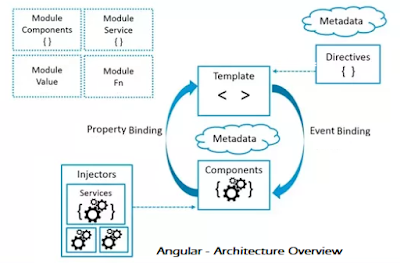
***What Is Injectors?***

A service is just a class in Angular until you register with an Angular dependency injector.

The injector is responsible for creating angular service instances and injecting them into classes.

You rarely create an injector yourself and Angular creates automatically during the bootstrap process.

Angular doesn't know automatically how you want to create instances of your services or injector. You must configure it by specifying providers for every service. Actually, providers tell the injector how to create the service and without a provider not able to create the service.



Bootstrap defines the components that should be bootstrapped when this module is bootstrapped. The components listed here will automatically be added to entryComponents.

Explore in detail by using below questions -

*1.* [*What Is an entryComponents?*](https://www.code-sample.com/2018/04/angular-6-5-4-entry-component.html)

2. What Are @Injectable providers?

3. [What Is Dependency Injection (DI)?](https://www.code-sample.com/2018/05/dependency-injection-di-angular-5-6.html)

I hope enjoy this post. Thank you very much reading this post.

# What Are @Injectable providers? - Injector decorator

[Anil Singh](https://plus.google.com/116567432705413398625) [1:50 AM](https://www.code-sample.com/2018/05/injectable-providers-in-angular-6-5-4.html)

*What Are @Injectable providers?*

The @Injectable decorator identifies services and other classes that are intended to be injected. It can also be used to configure a provider for those services.

To inject the service into a component, Angular provides an Injector decorator: @Injectable().

A provider defines the set of injectable objects that are available in the injector of this module.

The @Injectable decorator marks a class as available to an injector for instantiation. An injector reports an error when trying to instantiate a class that is not marked as @[Injectable](https://www.code-sample.com/2018/05/injectable-providers-in-angular-6-5-4.html).

Injectors are also responsible for instantiating components. At the run-time the injectors can read class metadata in the JavaScript code and use the constructor parameter type information to determine what things to inject.

Injectable decorator and metadata -

@Injectable({

providedIn?: Type<any> | 'root' | null

factory: () => any

})

To inject the service into a component, Angular provides an Injector decorator: @Injectable().

Here we configure a provider for CustomerService using the @Injectable decorator on the class.

We have the following steps to create a Service-

1. Create the service class

2. Define the metadata with a decorator

3. Import what we need.

In the above example, providedIn tells Angular that the root injector is responsible for creating an instance of the CustomerService.

The Angular CLI sets up provider automatically when you generating a new service.

***Why @Inject()?***

The @Inject is a special technique for letting Angular knows that a parameter must be injected.

Inject decorator and metadata-

@Inject({

token: any

})

When @Inject () is not present, Injector will use the type annotation of the parameter.

import { Component, OnInit, Inject } from '@angular/core';

import { HttpClient } from '@angular/common/http';

@Component({

selector: 'app-customer',

templateUrl: './customer.component.html',

styleUrls: ['./customer.component.css']

})

export class CustomerComponent implements OnInit {

constructor(@Inject(HttpClient) private http) {

// use this.http which is the Http provider.

}

ngOnInit(){ }

}

At this point, @Inject is a manual way of specifying this lookup token, followed by the lowercase http argument to tell Angular what to assign it against.

***What Is Hierarchical Dependency Injectors?***

Angular has a Hierarchical Dependency Injection system. There is actually a tree of injectors that parallel an application's component tree. You can reconfigure the injectors at any level of that component tree.

Explore in detail by using above questions -

*1.* [*What Is an entryComponents?*](https://www.code-sample.com/2018/04/angular-6-5-4-entry-component.html)

2. [What Are @Injectable providers?](https://www.code-sample.com/2018/05/injectable-providers-in-angular-6-5-4.html)

3. [What Is Dependency Injection (DI)?](https://www.code-sample.com/2018/05/dependency-injection-di-angular-5-6.html)

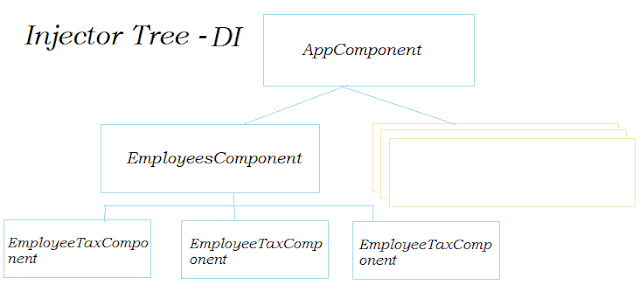
I hope enjoy this post. Thank you very much reading this post.

***What Is Injector tree?***

In the Dependency Injection guide, you learned how to configure a dependency injector and how to retrieve dependencies where you need them.

An application may have multiple injectors. An Angular application is a tree of components. Each component instance has its own injector. The tree of components parallels the tree of injectors.

Three level component tree –



***What Is Injector bubbling?***

***Explore in detail by using below questions -***

*1.* [*What Is an entryComponents?*](https://www.code-sample.com/2018/04/angular-6-5-4-entry-component.html)

2. [What Are @Injectable providers?](https://www.code-sample.com/2018/05/injectable-providers-in-angular-6-5-4.html)

3. [What Is Dependency Injection (DI)?](https://www.code-sample.com/2018/05/dependency-injection-di-angular-5-6.html)

I hope enjoy this post. Thank you very much reading this post.

**What Is Angular Service?**

Services are commonly used for storing data and making HTTP calls.

The main idea behind a service is to provide an easy way to share the data between the components and with the help of dependency injection (DI) you can control how the service instances are shared.

Services use to fetch the data from the RESTful API.

**How to setup and create services?**

Before starting to create a service, you just need to set up a dev environment.

Install **Node.js and npm** if they are not already on your machine.

Then install the **Angular CLI globally**.

npm install -g @angular/cli

Angular CLI tool will allow you to easily and quickly generate services, components, pipes, and many more files for your project.

Now, create a new project using the CLI command -

ng new MyProject

And go to your created project directory and launch the server.

ng serve --open

Now come to service - execute below the command for generating service class.

ng g service MyService

It will create the two files in the folder - src/app/

1. my-service.service.spec.ts

2. my-service.service.ts

Now, import my service file into the angular module - app.module.ts file. It looks like this.

import { AppComponent } from './app.component';

import {MyService} from './my-service.service';//Import Service

//AppModule class with @NgModule decorator

@NgModule({

//Static, This is the compiler configuration

//declarations is used for configure the selectors.

declarations: [

AppComponent

],

//Composability and Grouping

//imports used for composing NgModules together.

imports: [

BrowserModule

],

//Runtime or injector configuration

//providers is used for runtime injector configuration.

providers: [MyService],

//bootstrapped entry component

bootstrap: [AppComponent]

})

export class AppModule { }

And let’s see the created service class -my-service.service.ts

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

@Injectable()

export class MyService {

constructor() { }

}

Now you need to create service methods to get, post, put, and delete the users.

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

@Injectable()

export class MyService {

constructor() { }

//I’m using the static data

//You can also fetch the data using HttpClient service for backend APIs.

users = [

{ "id": 1, "name": "Anil Singh", 'age' :32 },

{ "id": 2, "name": "Aradhya" , 'age' :32},

{ "id": 3, "name": "Reena Singh" , 'age' :32}

]

/\* (method) MyService.getUsers(): {

"id": number;

"name": string;

'age': number;

}[] \*/

getUsers(){

return this.users;

}

}

Now, use this service in the user component for display on UI - my-user.component.ts

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

import {MyService} from '../my-service.service';

@Component({

selector: 'app-my-user',

templateUrl: './my-user.component.html',

styleUrls: ['./my-user.component.css']

})

export class MyUserComponent implements OnInit {

userList =[];

constructor(private service :MyService) {

//Get the user list by using the my service.

this.userList = service.getUsers();

}

ngOnInit() { }

}

And my-user.component.html

<h4>User List -</h4>

<table border="1">

<tr>

<th>ID</th><th>Name</th><th>Age</th>

</tr>

<tr \*ngFor="let user of userList">

<td>{{user.id}}</td><td>{{user.name}}</td><td>{{user.Age}}</td>

</tr>

</table>

Finally, you got the user list on your components HTML.

**What Is Singleton Service?**

In Angular, two ways to make a singleton service -

1. Include the service in the AppModule

2. Declare that the service should be provided in the application root.

The preferred way to create a singleton service - Form beginning to Angular 6 is –

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

@Injectable({

providedIn: 'root',

})

export class CustomerService {

}

Another way to create a singleton service - Include service in the AppModule

customer.service.ts –

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

@Injectable()

export class CustomersService {

constructor() { }

}

And app.module.ts -

import {CustomerService} from './customers.service';

//AppModule class with @NgModule decorator

@NgModule({

//Static, this is the compiler configuration

//declarations is used for configure the selectors.

declarations: [

AppComponent

],

//Composability and Grouping

//imports used for composing NgModules together.

imports: [

BrowserModule

],

//Runtime or injector configuration

//providers is used for runtime injector configuration.

providers: [CustomerService],

//bootstrapped entry component

bootstrap: [AppComponent]

})

export class AppModule { }

For more detail kindly [**refer the link....**](https://www.code-sample.com/2018/05/angular-5-6-7-directives-decorator.html)

**What Is Pipe?**

Pipes transform displayed values within a template.

Use the @Pipe annotation to declare that a given class is a pipe. A pipe class must also implement a PipeTransform interface.

The @Pipe decorator allows you to define the pipe name that is globally available for use in any template in the across Angular apps.

Pipe class implements the “PipeTransform” interfaces transform method that accepts an input value and returns the transformed result.

There will be one additional argument to the transform method for each parameter passed to the pipe.

The CLI commons for generate Pipe -

ng g pipe PipeName

//OR

ng generate pipe PipeName

Pipe decorator and metadata –

@Pipe({

name: string

pure?: boolean

})

The pipe name is used for template bindings.

To use the pipe you must set a reference to this pipe class in the module.

**Why use Pipes?**

Sometimes, the data is not displayed in the well format on the HTML templates that times were using pipes.

You also can execute a function in the HTML template to get its returned value.

For example - If you want to display a credit card number on your web apps - you can't display the whole number on your web app - you should write a custom logic to display card number as like \*\*\*\*-\*\*\*\*-2485 using your custom pipe.

**What Is PipeTransform interface?**

The Pipe class implements the PipeTransform interface that accepts input value (It is optional parameters) and returns the transformed value.

The transform method is an important method to a pipe.

To create a Pipe, you must implement this interface.

Angular invokes the transform method with the value of a binding as the first, and second argument in list form.

The PipeTransform interface looks like -

export interface PipeTransform {

transform(value: any, ...args: any[]): any;

}

And it imported from Angular core -

import {Pipe, PipeTransform} from '@angular/core';

Two Categories of Pipes in Angular –

1) pure

2) impure

Every pipe has been pure by default. If you want to make a pipe impure that time you will allow the setting pure flag to false.

**Angular Impure Pipes** –

**What Is Impure Pipe?**

Angular executes an impure pipe during every component change detection cycle. An impure pipe is called often, as often as every keystroke or mouse-move.

If you want to make a pipe impure that time you will allow the setting pure flag to false.

@Pipe({

name: 'currency',

pure:false

})

The example for impure pipe –

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'currency',

pure:false

})

export class CurrencyPipe implements PipeTransform {

transform(value: any, args?: any): any {

if (!value) {

return '1.00';

}

return value;

}

}

**Angular Pure Pipes –**

**What Is Pure Pipe?**

Angular executes a pure pipe only when it detects a pure change to the input value. A pure change can be primitive or non-primitive.

Primitive data are only single values, they have not special capabilities and the non-primitive data types are used to store the group of values.

@Pipe({

name: 'currency'

})

OR

@Pipe({

name: 'currency',

pure: true

})

And another example for a pure pipe –

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

name: 'currency'

})

export class CurrencyPipe implements PipeTransform {

transform(value: any, args?: any): any {

if (!value) {

return '1.00';

}

return value;

}

}

**The Pipe operator (|)** –

The pipe operator is used to specify a value transformation in an HTML template or view.

**Angular Parameterizing a Pipe** –

**What Is Parameterizing Pipe?**

A pipe can accept any number of optional parameters to achieve output. The parameter value can be any valid template expressions. To add optional parameters follow the pipe name with a colon (:). Its looks like- *currency: 'INR'*

In the following example –

<h2>The birthday is - {{ birthday | date:"MM/dd/yy" }} </h2>

<!-- Output - The birthday is - 10/03/1984 -->

**What Is Chaining Pipe?**

The chaining Pipe is used to perform the multiple operations within the single expression. This chaining operation will be chained using the pipe (I).

In the following example, to display the birthday in the upper case- will need to use the inbuilt date-pipe and upper-case-pipe.

In the following example –

{{ birthday | date | uppercase}}

<!-- The output is - MONDAY, MARCH 10, 1984 -->

**What Are Inbuilt Pipes in Angular?**

**Angular defines various Pipes API lists** – That is called Inbuilt Pipes.

1) DatePipe

2) CurrencyPipe

3) AsyncPipe

4) DecimalPipe

5) PercentPipe

6) UpperCasePipe

7) LowerCasePipe

8) TitleCasePipe

9) JsonPipe

10) SlicePipe

11) I18nSelectPipe

12) And many more

Similarly, you can also create a custom pipe (as per your needs) and configure in a module that is globally available in across angular apps.

**Angular DatePipe** -

The DatePipe is used to format a date with the help of locale rules.

{{ value\_expression | date [ : format [ : timezone [ : locale ] ] ] }}

The Example for date pipe –

The full date provides you full date for the date. The short date converts the date to a short date and the long date provides you long date for the date.

<h3>{{TodayDate}}</h3>

<h3>{{TodayDate | date:'shortDate'}}</h3>

<h3>{{TodayDate | date:'longDate'}}</h3>

<h3>{{TodayDate | date:'fullDate'}}</h3>

**Angular CurrencyPipe** –

The CurrencyPipe is used to format a currency with help of locale rules.

{{ value\_expression | currency [ : currencyCode [ : display [ : digitsInfo [ : locale ] ] ] ] }}

The CurrencyPipe formats a number as a currency of a specific country. It takes country currency type as a parameter.

The example for the currency pipe –

<tr>

<td>{{employee.salary | currency}}</td>

<td>{{employee.salary | currency : 'INR'}}</td>

<td>{{employee.salary | currency : 'INR' : true : '6.2'}}</td>

</tr>

**Angular AsyncPipe** –

Angular provide a special kind of pipe that are called AsyncPipe and the AsyncPipe subscribes to an observable or promise and returns the latest value it has emitted.

The AsyncPipe allows you to bind your HTML templates directly to values that arrive asynchronously manner that is a great ability for the promises and observables.

The expression with Async pipe-

{{ obj\_expression | async }}

OR

<ul><li \*ngFor="let account of accounts | async">{{account.ACNo }}</li></ul>

The object expression can be observable, promise, null, or undefined.

The example for AsyncPipe -

@Component({

selector: 'app-async-pipe',

template:`<ul><li \*ngFor="let account of accounts | async"> A/C No- {{account.ACNo }} </li></ul>`,

styleUrls: ['./async-pipe.component.css']

})

export class AsyncPipeComponent implements OnInit {

accounts = [];//accounts declarations

apiURL: string = '<https://api.github.com/anilsingh/accounts/>'; //fetching json data from Rest API

//AsyncPipe Component constructor

constructor(private accountService: AccountService) { }

//Load the account list

ngOnInit() {

this.accountService.getAccount(this.apiURL)

.subscribe(data => this.accounts = data);

}

}

**Angular PercentPipe** -

Angular provides a PercentPipe and it is used to format a number as a percentage according to below rules.

The expression rule with percent -

{{ value\_expression | percent [ : digitsInfo [ : locale ] ] }}

The input value to be formatted as a percentage and it can be any type.

The digitsInfo is optional string parameters and by default is undefined.

The locale is optional string parameters and by default is undefined.

The example as,

<h2>Result- {{marks | percent}}</h2>

<!-- output result is - '98%'-->

**Angular LowerCasePipe** -

Angular provides a LowerCasePipe and it is used to transforms given a text to lowercase.

The expression with lowercase -

{{ value\_expression | lowercase }}

The example as,

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

@Component({

selector: 'lowercase-pipe',

template: `<div>

<input type="text" #name (keyup)="changeLowerCase(name.value)">

<p>LowerCase - <h2>'{{value | lowercase}}'</h2>

</div>`

})

export class LowerCasePipeComponent {

value: string;

changeLowerCase(value: string) {

this.value = value;

}

}

**Angular UpperCasePipe** –

Angular provides an UpperCasePipe and it is used to transforms given a text to uppercase.

The expression with uppercase -

{{ value\_expression | uppercase }}

The example as,

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

@Component({

selector: 'uppercase-pipe',

template: `<div>

<input type="text" #name (keyup)="changeUpperCase(name.value)">

<p>UpperCase - <h2>'{{value | uppercase}}'</h2>

</div>`

})

export class UpperCasePipeComponent {

value: string;

changeUpperCase(value: string) {

this.value = value;

}

}

**Angular TitleCasePipe** –

The TitleCasePipe is used to converts the text (string type data) in which the first alphabet of each word is made capital latter and the rest will be in the small case letter.

The expression with titlecase -

{{ value\_expression | titlecase }}

The example as,

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

@Component({

selector: 'titlecase-pipe',

template: `<div>

<input type="text" #name (keyup)="changetitlecase(name.value)">

<p>titlecase - <h2>'{{value | titlecase}}'</h2>

</div>`

})

export class titlecasePipeComponent {

value: string;

changetitlecase(value: string) {

this.value = value;

}

}

# Routing and Navigation FAQs- Angular 4, 5, and 6

[Anil Singh](https://plus.google.com/116567432705413398625) [2:56 AM](https://www.code-sample.com/2018/05/angular-5-6-7-routing-and-navigation.html)

Stayed Informed – [Angular 2, 4, 5, 6 and 7 documentations including Examples](https://www.code-sample.com/2018/05/angular-6-7-documentation-and-examples.html)

**What is Angular Router?**

An Angular Router is a tool, library that configures navigations between states and views within your Angular app.

The Routing library is written and maintained by the [Angular Core Team](https://angular.io/about?group=Angular).

Angular router has own library package - @angular/router.

import {Routes, RouterModule,} from '@angular/router';

The basic concept of Angular Router and It allows you to -

1. Redirect a URL to another URL

2. Resolve data before a page is displayed

3. Run scripts when a page is activated or deactivated

4. Lazy load parts of our application

The router supports both styles with two LocationStrategy providers -

1. PathLocationStrategy— this is the default style.

2. HashLocationStrategy— adds the route path to the hash (#) in the browser’s URL.

**What is Router module?**

The Router module is a module that provides the necessary service providers and directives for navigating one view to other in the application.

**What is Routes?**

Angular Route is an array of route configurations. The “RouterModule.forRoot” method in the module imports to configure the router.

type Routes = Route[];

Each Route has the following properties -

interface Route {

path?: string

pathMatch?: string

matcher?: UrlMatcher

component?: Type<any>

redirectTo?: string

outlet?: string

canActivate?: any[]

canActivateChild?: any[]

canDeactivate?: any[]

canLoad?: any[]

data?: Data

resolve?: ResolveData

children?: Routes

loadChildren?: LoadChildren

runGuardsAndResolvers?: RunGuardsAndResolvers

}

List of properties and it has the following order -

1. path - It uses the route matcher DSL

2. pathMatch - It uses to specifies the matching strategy

3. matcher - It uses to defines a custom strategy for path matching

4. component - It is a component type

5. redirectTo - It is the URL fragment and it will replace the current matched segment

6. outlet - It is the name of the outlet the component should be placed into

7. canActivate - It is an array of DI tokens and used to handle the CanActivate handlers

8. canActivateChild - It is an array of DI tokens and used to handle the CanActivateChild handlers

9. canDeactivate - It is an array of DI tokens and used to handle the CanDeactivate handlers

10. canLoad - It is an array of DI tokens and used to handle the CanLoad handlers

11. data - It is additional data provided to the component by using the ActivatedRoute

12. resolve - It is a map of DI tokens used to look up data resolvers

13. runGuardsAndResolvers - It is defined when guards and resolvers will be run and by default, they run only when the matrix parameters of the route change.

14. children - it is an array of child route definitions

15. loadChildren - It is a reference to lazily loaded child routes.

**The following example** help you to understand the Router, Router module, and Routes -

In this example, the array of appRoots describes how to navigate from one view to other views and pass it into RouterModule.forRoot method to configure the router.

import { BrowserModule } from '@angular/platform-browser';

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

import {Routes, RouterModule,} from '@angular/router';

//Import Components

import { AppComponent } from './app.component';

import { DashboardComponent } from './dashboard/dashboard.component';

import { UserComponent } from './user/user.component';

import { UserDetailComponent } from './user-detail/user-detail.component';

import { PageNotFoundComponent } from './page-not-found/page-not-found.component'

//Apps roots

const appRoots = [

{ path: '', redirectTo: '/dashboard', pathMatch: 'full' },

{ path: 'user/:id', component: UserDetailComponent }, //Id is a Roots parameter.

{ path: 'users', component: UserComponent, data:{ title:'User List'} },

{ path: '\*\*', redirectTo: 'PageNotFoundComponent' } //Wild Cards (\*\*), the router will instantiate the PageNotFound component

];

//AppModule class with @NgModule decorator

@NgModule({

//Static, this is the compiler configuration

//declarations is used for configure the selectors

declarations: [

AppComponent,

DashboardComponent,

UserComponent,

UserDetailComponent,

PageNotFoundComponent,

],

//Composability and Grouping

//imports used for composing NgModules together

imports: [

BrowserModule,

//enableTracing is used for debugging purposes only

RouterModule.forRoot(appRoots, { enableTracing: true })

],

//Runtime or injector configuration

//providers is used for runtime injector configuration

providers: [],

//bootstrapped entry component

bootstrap: [AppComponent]

})

export class AppModule { }

**How Angular Router Works?**

Angular Router performs the following steps in order -

1. The router reads the browser URL the user wants to navigate to

2. The router applies a URL redirect (if one is defined otherwise page not found the error)

3. It figures out which router state corresponds to the URL

4. It runs the guards that are defined in the router state

5. It resolves the required data for the router state

6. It activates the Angular components to display the page

7. It manages navigations and repeats the steps above when a new page is requested.

Angular Router introduces the following terms and concepts -

1. <base href>

2. Router imports

3. Configuration

4. Router outlet

5. Router links

6. Router state

7. Activated route

8. Router events

**What Is <base href>?**

Most of all Angular routing apps must have the <base> element to the index.html or layout page in the <head> tag.

When using the PathLocationStrategy, need to tell the browsers what will be prefixed to the requested path to generate the URL.

You can specify a base URL like this –

<base href="/">

OR

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>My Demo Apps</title>

<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" type="image/x-icon" href="favicon.ico">

</head>

<body>

<app-root></app-root>

</body>

</html>

So if you had an Angular route defined like this –

{ path: 'users', component: UserComponent, data:{ title:'User List'} }

If <base href="/" >, the route become “/users”.

If <base href="/angular" >, the route become “/angular/users”.

**How to Append Base URL to HTTP requests?**

We can append base URL to HTTP requests using –

1. Dependency Injection

2. Using HttpInterceptors

The following example for append base URL using DI -

Firstly, we register a base URL provider in the NgModule and after register this BASE\_URL, it is available universally in your Apps.

//Runtime or injector configuration

//providers is used for runtime injector configuration.

providers: [{ provide: 'BASE\_URL', useFactory: getBaseUrl }],

Now provide factory method which gets the base URL from <base> element.

export function getBaseUrl() {

return document.getElementsByTagName('base')[0].href;

}

Finally, we can get the base URL injected and add it to URL-

export class GetUserComponent {

constructor(http: Http, @Inject('BASE\_URL') baseUrl: string) {

http.get(baseUrl + 'api/users').subscribe(data => {

this.users = data.json();

}, error => console.error(error));

}

}

The following example for append base URL using HttpInterceptors –

If we wants to create an interceptor, we must create an Injectable class which implements HttpInterceptor.

Firstly, register interceptor in the module provider -

//Runtime or injector configuration

//providers is used for runtime injector configuration.

providers: [{ provide: HTTP\_INTERCEPTORS, useClass: ApiInterceptor, multi: true } ],

And after register interceptor, to create –

@Injectable()

export class ApiInterceptor implements HttpInterceptor {

//Intercepts HttpRequest and handles them.

intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

const baseUrl = document.getElementsByTagName('base')[0].href;

const apiReq = req.clone({ url: `${baseUrl}${req.url}` });

return next.handle(apiReq);

}

}

Now we can access the base URL in your across apps.

**What Is PathLocationStrategy?**

A LocationStrategy used to configure the Location service that represents its state in the path of the browser's URL and the PathLocationStrategy is a default routing strategy.

When you are using the PathLocationStrategy, we must provide *APP\_BASE\_HREF* in the module or base element in the app document.

**What Is HashLocationStrategy?**

To enable HashLocationStrategy in an Angular app you pass {useHash: true} when you providing routes with router module and it like this.

//Composability and Grouping

//imports used for composing modules together.

imports: [

BrowserModule,

//enableTracing enables debugging purposes only

//useHash enables the location strategy that uses the URL fragment instead of the history API.

RouterModule.forRoot(appRoots, { enableTracing: true, useHash:true })

],

The HashLocationStrategy add the route path to the hash (#) in the browser’s URL.

The hash (#) part of the URL is called the hash fragment.

When using HashLocationStrategy for routing and providing a base Href, it always placed after the hash (#) e.g.

<http://localhost:8080/#/UserDetail/1>

The Hash style routing using the anchor tags technique to achieve client side routing and URL never sent to the server.

The anchor tag, when used along with the hash (#) allows us to jump to a place, within apps.

The URL would look like this -

1. <http://localhost:8080>

2. <http://localhost:8080/#/Users>

3. <http://localhost:8080/#/UserDetail/1>

In the above URLs “#/Users” and “#/UserDetail/1” never sent to the server.

**How do you change the base URL dynamically?**

Instead of setting the base element's href value, you can set the base URL programmatically, by providing for APP\_BASE\_HREF with your custom operation.

For more detail, referee to the above questions.

**What Is Router imports?**

It is an optional service that presents a special component view for a given URL. It has own library package- @angular/router and It is not a part of an Angular core.

The Angular package looks like this.

import {Routes, RouterModule,} from '@angular/router';

**How to configure Angular routes?**

A router has no routes until you configure it. So you are configuring the Angular router for accessing your apps URLs.

//Composability and Grouping

//imports used for composing NgModules together.

imports: [

BrowserModule,

//enableTracing is used for debugging purposes only

//Enables the location strategy that uses the URL fragment instead of the history API.

RouterModule.forRoot(appRoots, { enableTracing: true, useHash:false })

]

For the detail, you can refer Angular Routes example for the same.

**What Is Router outlet?**

The Router-Link, RouterLink-Active and the Router outlet is directive provided by the Angular RouterModule package. It’s provides the navigation and URLs manipulation capabilities. It also renders the components for specific location of your applications.

Both the template and templateUrl render the components where you use this directive.

<router-outlet> </router-outlet>

**Is it possible to have a multiple router-outlet in the same template?**

Yes, why not! We can use multiple router-outlets in the same template by configuring our routers and simply adds the router-outlet name.

<div class="row">

<div class="user">

<router-outlet name="users"></router-outlet>

</div>

<div class="detail">

<router-outlet name="userDetail"></router-outlet>

</div>

</div>

And setups your route config and it looks like this.

//Apps roots

const appRoots = [

{ path: '', redirectTo: '/dashboard', pathMatch: 'full' },

{ path: 'userDetail', component: UserDetailComponent }, //Id is a Roots parameter.

{ path: 'users', component: UserComponent, data:{ title:'User List'} },

{ path: '\*\*', redirectTo: 'PageNotFoundComponent' } //Wild Cards, the router will instantiate the PageNotFound component.

];

And

//AppModule class with @NgModule decorator

@NgModule({

//Composability and Grouping

//imports used for composing NgModules together

imports: [

BrowserModule,

//enableTracing is used for debugging purposes only

//Enables the location strategy that uses the URL fragment instead of the history API.

RouterModule.forRoot(appRoots)

],

//bootstrapped entry component

bootstrap: [AppComponent]

})

export class AppModule { }

**What Is Router link?**

The Router-link is a directive and it used to link a specific part of your applications.

@Directive({ selector: ':not(a)[routerLink]' })

Let explain the route configuration using the -

{ path: 'user/:id', component: UserDetailComponent

I the above rote configuration, when linking to this user/:id route, you use the RouterLink directive.

If the link is static, you can use the directive as follows.

<a routerLink="/user/id"> See the User detail</a>

If you using dynamic values to generate the router link that you can pass an array of path segments.

You can specify a route parameter like this.

<a [routerLink]="['/user', user.id]">

<span class="text-align">{{ user.id }}</span>{{ user.name }}

</a>

You can set query params and fragment as follows.

<a [routerLink]="['/user/id']" preserveQueryParams preserveFragment>

See the user component

</a>

You can specify optional route parameters like this.

<a [routerLink]="['/user-detail', { id: '102348014' }]">User Detail</a>

And

@Component({

selector: 'app-user',

template: `<nav>

<a [routerLink]="['/users']">User List</a>

<a [routerLink]="['/userDetail/101', { Id: '102348014' }]">User Detail</a>

</nav>

<router-outlet></router-outlet>`,

styleUrls: ['./user.component.css']

})

**What Is RouterLinkActive?**

The RouterLinkActive is a directive. To add the active CSS class to the element when the associated RouterLink becomes active (visually look like selected anchors). It is also works for both parent and child elements.

@Directive({

selector: '[routerLinkActive]',

exportAs: 'routerLinkActive'

})

Consider the following example for active a link –

<a routerLink="/user/detail" routerLinkActive="active-link">User Detail</a>

You can also set more than one class and it look like this.

<a routerLink="/user/detail" routerLinkActive="active-class1 active-class2">User detail</a>

<a routerLink="/user/detail" [routerLinkActive]="['active-class1', 'active-class2']">User detail</a>

**What Is RouterState?**

RouterState is interface and it represents the state of the router.

It looks like this.

interface RouterState extends Tree {

snapshot: RouterStateSnapshot

toString(): string

}

It is also a tree of activated routes.

We can access the current RouterState from anywhere in the Angular app using the Router service and the routerState property.

**What Is ActivatedRoute?**

ActivatedRoute is an interface and it contains the information about a route associated with a component loaded into an outlet and it can also be used to traverse the router state tree.

And it contains the list of Properties –

1. Snapshot – It is the current snapshot of this route.

2. URL – It is an observable of the URL segments and it matched by this route

3. Params – It is an observable of the matrix parameters scoped to this route

4. QueryParams - it is an observable of the query parameters shared by all the routes

5. Fragment- It is an observable of the URL fragment shared by all the routes

6. Data- It is an observable of the static and resolved data of this route.

7. Outlet. It's a constant and outlet name of the route

8. Component- It's a constant and a component of the route

9. RouteConfig- This configuration used to match this route

10. Root- This is the root of the router state

11. Parent - The parent of this route in the router state tree

12. FirstChild- The first child of this route in the router state tree

13. Children- The children of this route in the router state tree

14. pathFromRoot- The path from the root of the router state tree to this route

15. paramMap- It is read-only

16. queryParamMap- It is read-only

**What Is Router events?**

Whenever the root navigations, the router emits navigation events using Router.events property.

The sequence of router events is -

1. NavigationStart

2. RouteConfigLoadStart

3. RouteConfigLoadEnd

4. RoutesRecognized

5. GuardsCheckStart

6. ChildActivationStart

7. ActivationStart

8. GuardsCheckEnd

9. ResolveStart

10. ResolveEnd

11. ActivationEnd

12. ChildActivationEnd

13. NavigationEnd

14. NavigationCancel

15. NavigationError

The Router events are also logged in the console when enableTracing option is enabled.

The NavigationStart event is triggered when navigation starts.

The RoutesRecognized event triggered when the routes are recognized.

The RouteConfigLoadStart event triggered before the Router lazy loads.

The RouteConfigLoadEnd event triggered after a route has been lazily loaded.

The NavigationEnd event triggered when navigation ends successfully.

The NavigationCancel event triggered when navigation is canceled.

The NavigationError event triggered when router navigation fails due to an error.

Stayed Informed – [Angular 4, 5, 6 and 7 documentations including Examples](https://www.code-sample.com/2018/05/angular-6-7-documentation-and-examples.html)

# What Is HttpClient in Angular?

[Anil Singh](https://plus.google.com/116567432705413398625) [11:57 PM](https://www.code-sample.com/2018/05/angular-5-6-httpclient-requests-and.html)

**What Is HttpClient in Angular?**

**What Is the role and responsibility of HttpClient?**

HttpClient is performing HTTP requests and responses.

Most of all web applications communicate with backend services over the HTTP protocol and all modern browsers support two different APIs for making HTTP requests i.e.

1. XMLHttpRequest interface

2. fetch() APIs

The HttpClient is more modern and easy to use the alternative of HTTP.

HttpClient is an improved replacement for HTTP. They expect to deprecate http in Angular 5 and remove it in a later version.

The new HttpClient service is included in the HTTP Client Module that used to initiate HTTP request and responses in angular apps. The HttpClientModule is a replacement of HttpModule.

HttpClient also gives us advanced functionality like the ability to listen for progress events and interceptors to modify requests or responses.

Before using the HttpClient, you must need to import the Angular HttpClientModule and the HttpClientModule is imported from *@angular/common/http*.

You must import HttpClientModule after BrowserModule in your angular apps.

First, you’ll need to imported HttpClientModule from @angular/common/http in your app module and it must be import HttpClientModule after BrowserModule in your angular apps.

The following example as given below -

import { BrowserModule } from '@angular/platform-browser';

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

import { HttpClientModule } from '@angular/common/http';

//Import App Component

import { AppComponent } from './app.component';

//AppModule class with @NgModule decorator

@NgModule({

imports: [

BrowserModule,

//import HttpClientModule after BrowserModule

HttpClientModule,

],

//Static, compiler configuration

//declarations is used for configure the selectors

declarations: [

AppComponent,

],

//Runtime or injector configuration

providers: [],

//bootstrapped entry component

bootstrap: [ AppComponent ]

})

export class AppModule {}

After imported HttpClientModule into the AppModule, you can inject the HttpClient into your created service.

The following example as give below–

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

import { HttpClient } from '@angular/common/http';

@Injectable()

export class CustomerService {

//Inject HttpClient into your components or services

constructor(private httpClient: HttpClient) { }

}

HttpClient supports mutating a request, which is, sending data to the server with HTTP methods such as PUT, POST, and DELETE.

HttpClient is use the XMLHttpRequest browser API to execute HTTP request.

HttpClient Perform HTTP requests –

1. GET method – get()

2. POST method – post()

3. PUT method – put()

4. DELETE method – delete()

5. HEAD method – head()

6. JSONP method – jsonp()

7. OPTIONS method – options()

8. PATCH method – patch()

And the HttpClient class looks like –

class HttpClient {

constructor(handler: HttpHandler)

request(first: string | HttpRequest<any>, url?: string, options: {...}): Observable<any>

delete(url: string, options: {...}): Observable<any>

get(url: string, options: {...}): Observable<any>

head(url: string, options: {...}): Observable<any>

jsonp<T>(url: string, callbackParam: string): Observable<T>

options(url: string, options: {...}): Observable<any>

patch(url: string, body: any | null, options: {...}): Observable<any>

post(url: string, body: any | null, options: {...}): Observable<any>

put(url: string, body: any | null, options: {...}): Observable<any>

}

The options contain the list of parameters –

1. headers

2. observe,

3. params,

4. reportProgress,

5. responseType,

6. withCredentials

The following options parameters look like -

options: {

headers?: HttpHeaders | {

[header: string]: string | string[];

};

observe?: 'body';

params?: HttpParams | {

[param: string]: string | string[];

};

reportProgress?: boolean;

responseType: 'blob';

withCredentials?: boolean;

}

Benefits of HttpClient -

1. HttpClient include the testability features

2. HttpClient include typed request

3. HttpClient include response objects

4. HttpClient include request and response interception

5. HttpClient include Observable APIs

6. HttpClient include error handling

# What Is Angular HttpInterceptor?

[Anil Singh](https://plus.google.com/116567432705413398625) [11:59 PM](https://www.code-sample.com/2018/05/angular-5-6-7-httpinterceptor.html)

HttpInterceptor - HttpInterceptors is an interface which uses to implement the intercept method.

Intercepts HttpRequest and handles them.

Intercepts is an advanced feature that allows us to intercept each request/response and modify it before sending/receiving.

Interceptors capture every request and manipulate it before sending and also catch every response and process it before completing the call.

Firstly, we can implement own interceptor service and this service will “catch” each request and append an Authorization header.

You can see in the following example,

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

import {HttpEvent, HttpInterceptor, HttpHandler, HttpRequest} from '@angular/common/http';

@Injectable()

export class MyInterceptor implements HttpInterceptor {

//Intercepts HttpRequest and handles them.

intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

const reqHeader = req.clone({headers: req.headers.set('Authorization', 'MyAuthToken')});

return next.handle(reqHeader);

}

}

After that you can configure your own interceptor service (MyInterceptor) and HTTP\_INTERCEPTORS in the app Module.

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

import {HTTP\_INTERCEPTORS} from '@angular/common/http';

@NgModule({

providers: [{

provide: HTTP\_INTERCEPTORS,

useClass: MyInterceptor,

multi: true,

}],

})

export class AppModule {}

Following this logic, Authorization token will be appended to each request. It’s also possible to override request’s headers by using **set**() method.

# What’s the difference between HTTP and HttpClient?

[Anil Singh](https://plus.google.com/116567432705413398625) [12:07 AM](https://www.code-sample.com/2018/05/angular-5-67-http-vs-httpclient.html)

Angular HTTP vs. HttpClient -

The HttpClient is used to perform HTTP requests and it imported form @angular/common/http.

The HttpClient is more modern and easy to use the alternative of HTTP.

HttpClient is an improved replacement for Http. They expect to deprecate Http in Angular 5 and remove it in a later version.

It's an upgraded version of http from @angular/http module with the following improvements –

1. Immutable request and response objects

2. Interceptors allow middleware logic to be inserted into the pipeline

3. Progress events for both request and response

4. Typed

5. Event firing

6. Synchronous response body access

7. Support JSON body types and JSON by default and now, no need to be explicitly parsed

8. Automatic conversion from JSON to an object

9. Post request verification

10. A flush based testing framework

11. Simplified syntax for headers

The example with HttpClient -

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

import { HttpClient } from '@angular/common/http';

@Injectable()

export class CustomerService {

//Inject HttpClient into your components or services

constructor(private http: HttpClient) { }

}

And other example with Http –

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

import { Http } from '@angular/http';

@Injectable()

export class CustomerService {

//Inject HttpClient into your components or services

constructor(private http: Http) { }

}

For more detail kindly[**refer the link....**](https://www.code-sample.com/2018/05/angular-5-6-httpclient-requests-and.html)

# What's the difference between HttpModule and HttpClientModule?

[Anil Singh](https://plus.google.com/116567432705413398625) [12:01 AM](https://www.code-sample.com/2018/05/angular-6-httpmodule-vs-httpclientmodule.html)

HttpModule vs. HttpClientModule -

*HttpClientModule* -

The HttpClientModule imported form -

import { HttpClientModule } from '@angular/common/http';

NgModule which provides the HttpClient and associated with components services and the interceptors can be added to the chain behind HttpClient by binding them to the multi-provider for HTTP\_INTERCEPTORS.

*HttpModule* –

Http deprecate @angular/http in favour of @angular/common/http.

HttpModule imported from –

import { HttpModule } from '@angular/http';

They both support HTTP calls but HTTP is the older API and will eventually be deprecated.

The new HttpClient service is included in the HttpClientModule that used to initiate HTTP request and responses in angular apps. The HttpClientModule is a replacement of HttpModule.

For more detail kindly[**refer the link....**](https://www.code-sample.com/2017/08/angular-5-interview-questions-and.html)

# What Are Angular HttpHeaders?

[Anil Singh](https://plus.google.com/116567432705413398625) [12:11 AM](https://www.code-sample.com/2018/05/angular-5-6-7-httpheaders.html)

What Are HttpHeaders?

The Http Headers is immutable Map and each and every set() returns a new instance and applies the changes with lazy parsing.

An immutable set of Http headers, with lazy parsing.

HttpHeaders Constructor -

constructor(headers?: string | { [name: string]: string | string[];});

Imports HttpHeaders from -

import {HttpHeaders } from '@angular/common/http';

HttpHeaders class contains the list of methods -

1. has() - Checks for existence of header by given name.

2. get() - Returns the first header that matches given name.

3. keys() - Returns the names of the headers

4. getAll() - Returns list of header values for a given name.

5. append() - Append headers by chaining.

6. set() - To set a custom header on the request for a given name

7. delete() - To delete the header on the request for a given name

For more detail kindly[**refer the link....**](https://www.code-sample.com/2017/08/angular-5-interview-questions-and.html)

# How to set a custom header on the request?

[Anil Singh](https://plus.google.com/116567432705413398625) [12:14 AM](https://www.code-sample.com/2018/05/angular-5-6-7-set-custom-header.html)

How to set a custom header on the request?

To set a custom header on the request, firstly we need to instantiate HttpHeaders() object and pass ('header', 'value') into a function.

let headers = new HttpHeaders().set('Content-Type', 'text');

In the above example we set “Content-Type” header value to be “text” and the default header “Content-Type” is – “application/json”

It is of type immutable Map so if you assign a new value it will reinitialize the object.

let requestHeaders = new HttpHeaders().set('Content-Type', 'application/json');

requestHeaders = requestHeaders.set('authorization', 'Bearer ' + token);

We can also append headers by chaining HttpHeaders() constructor and will look like this-

let requestheaders = new HttpHeaders().set('Content-Type', 'application/json')

.set('authorization', 'Bearer ' + token);

And final request with custom headers will look like this –

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

import { HttpClient, HttpHeaders } from '@angular/common/http';

@Injectable()

export class CustomerService {

//Inject HttpClient into your components or services

constructor(private http: HttpClient) { }

//Set Headers

requestHeaders = new HttpHeaders().set('Content-Type', 'text')

.append('Authorization', 'CustomToke\_AFA96A3429A9524');

//Get Customer list

getCustomers() {

this.http.get('https://code-sample.com/customerjson', {

headers: this.requestHeaders

}).map((data:HttpEvent<object>) => { console.log(data) })

}

}

For more detail kindly[**refer the link....**](https://www.code-sample.com/2017/08/angular-5-interview-questions-and.html)

# How to catch and log specific Angular errors in your app?

[Anil Singh](https://plus.google.com/116567432705413398625) [12:16 AM](https://www.code-sample.com/2018/05/try-catch-and-log-angular-errors.html)

How to catch and log specific Angular errors in your app?

The default implementation of ErrorHandler log error messages, status, and stack to the console.

To intercept error handling, we write a custom exception handler that replaces this default as appropriate for your app.

import { Injectable, ErrorHandler } from '@angular/core';

import {ErrorLoggService} from './error-logg.service';

// Global error handler for logging errors

@Injectable()

export class GlobalErrorHandler extends ErrorHandler {

constructor(private errorLogService: ErrorLoggService) {

//Angular provides a hook for centralized exception handling.

//constructor ErrorHandler(): ErrorHandler

super();

}

handleError(error) : void {

this.errorLogService.logError(error);

}

}

For more detail kindly[**refer the link....**](https://www.code-sample.com/2017/08/angular-5-interview-questions-and.html)

# How to Create a Custom ErrorHandler?

[Anil Singh](https://plus.google.com/116567432705413398625) [12:19 AM](https://www.code-sample.com/2018/05/angular-5-6-7-custom-errorhandler.html)

How to create a custom ErrorHandler?

The best way to log exceptions is to provide a specific log message for each possible exception. Always ensure that sufficient information is being logged and that nothing important is being excluded.

The multiple steps involved in creating custom error logging in Angular -

1. Create a constant class for global error messages.

2. Create an error log service – [ErrorLoggService](https://www.code-sample.com/2018/05/angular-5-6-7-custom-errorhandler.html).

3. Create a global error handler for using the error log service for logging errors.

Steps 1 – In the first step, we will create a constant class for logging global error messages and its look like this.

export class AppConstants {

public static get baseURL(): string { return '<http://localhost:4200/api>'; }

public static get httpError(): string { return 'There was an HTTP error.'; }

public static get typeError(): string { return 'There was a Type error.'; }

public static get generalError(): string { return 'There was a general error.'; }

public static get somethingHappened(): string { return 'Nobody threw an Error but something happened!'; }

}

Steps 2 – In the second steps, we will create an error log service (ErrorLoggService) for error logging and it look like this.

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

import { HttpErrorResponse } from '@angular/common/http';

import{ AppConstants} from '../app/constants'

//#region Handle Errors Service

@Injectable()

export class ErrorLoggService {

constructor() { }

//Log error method

logError(error: any) {

//Returns a date converted to a string using Universal Coordinated Time (UTC).

const date = new Date().toUTCString();

if (error instanceof HttpErrorResponse) {

//The response body may contain clues as to what went wrong

console.error(date, AppConstants.httpError, error.message, 'Status code:',

(<HttpErrorResponse>error).status);

}

else if (error instanceof TypeError) {

console.error(date, AppConstants.typeError, error.message, error.stack);

}

else if (error instanceof Error) {

console.error(date, AppConstants.generalError, error.message, error.stack);

}

else if(error instanceof ErrorEvent){

//A client-side or network error occurred. Handle it accordingly

console.error(date, AppConstants.generalError, error.message);

}

else {

console.error(date, AppConstants.somethingHappened, error.message, error.stack);

}

}

}

//#endregion

Steps 3 – In the 3rd steps, we will create a global error handler for using the error log service for logging errors and its look like this.

import { Injectable, ErrorHandler } from '@angular/core';

import {ErrorLoggService} from './error-logg.service';

// Global error handler for logging errors

@Injectable()

export class GlobalErrorHandler extends ErrorHandler {

constructor(private errorLogService: ErrorLoggService) {

//Angular provides a hook for centralized exception handling.

//constructor ErrorHandler(): ErrorHandler

super();

}

handleError(error) : void {

this.errorLogService.logError(error);

}

}

Steps 4 – In the 4th step, we will Import global handler and error handler services in the NgModule and its look like this.

import {ErrorLoggService} from './error-logg.service';

import {GlobalErrorHandler} from './global-error.service';

//AppModule class with @NgModule decorator

@NgModule({

//declarations is used for configure the selectors

declarations: [

AppComponent,

],

//Composability and Grouping

imports: [

BrowserModule,

HttpClientModule

],

//Runtime or injector configuration

//Register global error log service and error handler

providers: [ErrorLoggService, GlobalErrorHandler],

//bootstrapped entry component

bootstrap: [AppComponent]

})

export class AppModule { }

Steps 5 – Finally, we got a custom error handler for log error in your application errors.

**What Happens If the Request fails on the Server Due to Poor Network Connection?**

HttpClient will return an error instead of a successful response.

# Angular 4, 5, 6 Form, Template, and Validations

[Anil Singh](https://plus.google.com/116567432705413398625) [10:50 PM](https://www.code-sample.com/2018/05/angular-5-6-7-form-validations.html)

**What are the Validator functions?**

There are two types of validator functions which are the following -

1. Async validators

2. Sync validators

**Async** validator functions that take a control instance and return an **observable** that later emits a set of validation errors or null.

**Sync** validator functions that take a control instance and return a set of validation errors or null.

Angular runs only [Async validators](https://www.code-sample.com/2018/05/angular-5-6-7-form-validations.html) due to some performance issues.

**What Is a Template Reference variable?**

A template reference variable is a way of capturing a reference to a specific element, component, directive, and pipe so that it can be used someplace in the same **template HTML**.

You should declare a reference variable using the **hash symbol (#)**.

The Angular components and directives only match selectors for classes that are declared in the Angular module.

Template Reference Variable Syntax –

You can use a template reference variable by two ways.

1. Using hash symbol (#)

2. Using reference symbol (ref-)

The following examples of specifying a template reference variable using Input Text Box –

I have declared a reference variable “cellnumber” using the **hash symbol (#)** and **reference symbol (ref-)**.

<input type="text" ref-cellnumber> //cellnumber will be a template reference variable.

And

<input #cellnumber placeholder="Cell number"> //cellnumber will be a template reference variable.

I have created a reference to the input element that can be used later on in my template and the scope for “**cellnumber**” variable is the entire HTML template in which the reference is defined.

Here is how I could use that reference to get the value of the input for instance –

//cellnumber refers to the input element

<button (click)="show(cellnumber)">click to see</button>

In the below line of code, the variable “cellnumber” refer to the HTMLElement object instance for the input -

show(cellnumber: HTMLInputElement){

console.log(cellnumber.value);

}

You can use the ViewChild decorator to reference it inside your component.

import {ViewChild, ElementRef} from '@angular/core';

// Reference cellnumber variable inside Component

@ViewChild('cellnumber') cellInputRef: ElementRef;

And finally, you can use this.nameInputRef anywhere inside your component class.

show(){

this.contactNumber = this.cellInputRef.nativeElement.value

}

Template Reference Variable with NgForm –

Here we will discuss about how to access NgForm directive using template reference variable.

<form (ngSubmit)="onSubmitEmployee(empForm)" #empForm="ngForm">

<label>F-Name </label><input name="f-name" required [(ngModel)]="employee.fname">

<label>L-Name </label><input name="l-name" required [(ngModel)]="employee.lname">

<label>Age </label><input name="age" required [(ngModel)]="employee.age">

<button type="submit" [disabled]="!empForm.form.valid">Submit</button>

</form>

In the above NgForm example contains a ngSubmit event and form directive.

The ngSubmit – The ngSubmit directive specifies a function to run when the form is submitted. Here on form submit onSubmitEmployee component method will be called.

The NgForm - It is a nestable alias of form directive. The main purpose of ngForm is to group the controls, but not a replacement of <form> tag.

As you know, the HTML does not allow nesting of form elements. It is very useful to nest forms.

**How to bind to user input events to component event handlers?**

Most of the DOM events are triggered by user input and bind to these events provides a way to get inputs from a user.

The following example shows a click event binding – [on-click.component.ts]

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

@Component({

selector: 'app-on-click',

templateUrl: './on-click.component.html',

styleUrls: ['./on-click.component.css']

})

export class OnClickComponent implements OnInit {

welcomeMsg = '';

constructor() { }

ngOnInit() { }

onClick() {

this.welcomeMsg = 'Welcome you, Anil!';

}

}

And on-click.component.html -

<div class="msg">

<button (click)="onClick()">Click Me!</button>

<p>

{{welcomeMsg}}

</p>

</div>

OR

<!-- Canonical form, the (on-) prefix alternative -->

<div class="msg">

<button on-click="onClick($event)">Click Me!</button>

<p>

{{welcomeMsg}}

</p>

</div>

When the user clicks the button, Angular calls the onClick method from OnClickComponent.

**How to get user input from the $event object?**

The DOM events carry all information that is useful to the component.

The following example shows to get user input from the $event – key-up.component.ts

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

@Component({

selector: 'app-key-up',

templateUrl: './key-up.component.html',

styleUrls: ['./key-up.component.css']

})

export class KeyUpComponent implements OnInit {

values = '';

constructor() { }

ngOnInit() { }

//KeyUp events.

onKeyUp(event: any) {

this.values += event.target.value + ' : ';

}

}

And key-up.component.html –

<div class="event">

<button (click)="onKeyUp($event)">KeyUp Event!</button>

<p>

{{values}}

</p>

</div>

**How to get user input from a template reference variable?**

This is the other way to get the user data. It is also called #var.

“A template reference variable is mostly a reference to a DOM element within a template. It can also be a reference to Angular components or directives and others.”

It looks like this.

<input #name placeholder="Enter Name">

The following example shows to get user input from a template reference variable - template-reference.component.ts

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

@Component({

selector: 'app-template-reference',

templateUrl: './template-reference.component.html',

styleUrls: ['./template-reference.component.css']

})

export class TemplateReferenceComponent implements OnInit {

constructor() { }

ngOnInit() {

}

}

And template-reference.component.html –

<div class="event">

<button #keydownVal (keydown)="0"></button>

<p>

{{keydownVal.value}}

</p>

</div>

**How to create a custom validator for both model driven and template driven forms?**

There are two types of Validators –

1. Built-in Validators

2. [Custom Model Form Validators](https://www.code-sample.com/2018/04/angular-5-custom-form-validations.html)

a. Email Validator

b. Password Validator

c. Secure Site Validator

d. Credit card validator

Built-in Validators -

1. Validators .required - Requires a form control to have a non-empty value

2. Validators .minlength - Requires a form control to have a value of a min length

3. Validators .maxlength - Requires a form control to have a value of a max length

4. Validators .pattern - Requires a form control’s value to match a given regex

5. And so on

Built-in validator looks like –

this.empForm = new FormGroup({

'email': new FormControl(this.employee.email,[Validators.required, ValidationService.emailValidator]),

'name': new FormControl(this.employee.name, [Validators.required,Validators.minLength(4)]), 'Dep': new FormControl(this.employee.Dep, [Validators.required, Validators.minLength(10)]), 'Desc': new FormControl(this.employee.Desc, [Validators.required, Validators.minLength(100),Validators.minLength(500)]),

});

Custom Model Form Validators – Validators are core functions, they take as input a FormControl instance and returns either null if it’s valid or flag for errors.

You can use the custom validator to validate a specific requirement like -

1. Email Validator

2. Password Validator

3. Secure Site Validator

4. Credit card validator

5. And may more

The Following Steps involve CREATING custom validators -

**Steps 1**- Create validation service using the CLI command.

ng g service validation

**Steps 2** - import validation service in your app NgModule –

import { BrowserModule } from '@angular/platform-browser';

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

import {FormsModule, FormGroup} from '@angular/forms';

import {RouterModule} from '@angular/router';

import {HttpClientModule} from "@angular/common/http";

//MY COMPONENTS

import { AppComponent } from './app.component';

import { LoginComponent } from './login/login.component';

import { RegisterComponent } from './register/register.component';

import { EmployeeComponent } from './employee/employee.component';

//My Services

import { AuthServiceService } from './auth-service.service';

import { AuthGuard } from './auth.guard';

import { EmployeeService} from './employee.service';

import { ValidationService } from './validation.service';

@NgModule({

declarations: [

AppComponent,

LoginComponent,

RegisterComponent,

EmployeeComponent

],

imports: [

BrowserModule,

FormsModule,

HttpClientModule,

RouterModule.forRoot([

{ path: '', component: AppComponent, pathMatch: 'full' },

{ path: 'register', component: RegisterComponent },

{ path: 'employee', component: EmployeeComponent},

{ path: 'login', component: LoginComponent}])

],

providers: [EmployeeService, ValidationService],

bootstrap: [AppComponent]

})

export class AppModule { }

**Steps 3** - Write the customer validation method in your validation.service.ts -

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

@Injectable()

export class ValidationService {

constructor() { }

//Check Site contains SSL Security protocol or Not.

static secureSiteValidator(control){

if (!control.value.startsWith('https') || !control.value.includes('.in')) {

return { IsSecureSite: true };

}

return null;

}

//Email Validator

static emailValidator(control) {

if (control.value.match(/[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*@(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?/)) {

return null;

}

else {

return { 'InvalidEmail': true };

}

}

//Password Validator

static passwordValidator(control) {

if (control.value.match(/^(?=.\*[0-9])[a-zA-Z0-9!@#$%^&\*]{6,100}$/)) {

return null;

}

else {

return { 'InvalidPassword': true };

}

}

}

Steps 4 - Use of validation service in your components and its looks like –

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

import {Employee } from '../employee'

import { Validators, FormGroup, FormControl } from '@angular/forms';

import {EmployeeService} from '../employee.service'

import { ValidationService } from '../validation.service';

@Component({

selector: 'app-employee',

templateUrl: './employee.component.html',

styleUrls: ['./employee.component.css']

})

export class EmployeeComponent implements OnInit {

constructor( public \_empService: EmployeeService) { }

empForm:any;

ngOnInit() {

this.empForm = new FormGroup({

'email': new FormControl(this.employee.email,[Validators.required, ValidationService.emailValidator]),

'name': new FormControl(this.employee.name, [Validators.required,Validators.minLength(4)]),

'Dep': new FormControl(this.employee.Dep, [Validators.required, Validators.minLength(10)]),

'Desc': new FormControl(this.employee.Desc, [Validators.required, Validators.minLength(100),Validators.minLength(500)]),

});

}

employee = new Employee(0,'','','','','');

submitted = false;

//Add new Employee

onSubmit() {

this.submitted = true;

let isSuccess = this.\_empService.addEmployee(this.employee);

if(isSuccess){

//handle success

console.log(isSuccess);

}else{

//handle errors

}

}

}

And

<div class="container">

<h1>Employee Form</h1>

<form #empForm="ngForm" (ngSubmit)="onSubmit()">

<div class="form-group">

<label for="name">Email</label>

<input type="text" class="form-control" id="email” required [(ngModel)]="employee.email" name="email">

</div>

<div class="form-group">

<label for="name">Name</label>

<input type="text" class="form-control" id="name” required [(ngModel)]="employee.name" name="name">

</div>

<div class="form-group">

<label for="Dep">Department</label>

<input type="text" class="form-control" id="Dep” required [(ngModel)]="employee.Dep" name="Dep">

</div>

<div class="form-group">

<label for="Desc">Desc</label>

<input type="text" class="form-control" id="Desc” required [(ngModel)]="employee.Desc" name="Desc">

</div>

<button type="submit" class="btn btn-success" [disabled]="!empForm.form.valid">Submit</button>

</form>

<div [hidden]="!submitted">

<h4 style="color:green;">Record Added Successfully!</h4>

</div>

</div>

**For more detail kindly refer the link** - [https://www.code-sample.com/2018/05/angular-6-7-documentation-and-examples.html](https://www.code-sample.com/2018/05/angular-6-7-documentation-and-examples.html%C2%A0)

# Angular Cookies QAs

[Anil Singh](https://plus.google.com/116567432705413398625) [10:01 PM](https://www.code-sample.com/2018/05/angular-5-6-7-cookies-examples.html)

**What is a Cookie?**

A cookie is a small piece of data sent from a website and stored on the user's machine by the user's web browsers while the user is browsing.

OR

Cookies are small packages of information that are typically stored in your browsers and websites tend to use cookies for multiple things.

Cookies persist across multiple requests and browser sessions should you set them to and they can be a great method for authentication in some web apps.

**How to install a cookie in Angular?**

Install cookie -

npm install ngx-cookie-service –save

If you do not want to install this via NPM, you can run npm run compile and use the \*.d.ts and \*.js files in the dist-lib folder

After installed successfully, add the cookie service in the Angular module - app.module.ts

import {CookieService} from 'ngx-cookie-service'

//AppModule class with @NgModule decorator

@NgModule({

//Static, this is the compiler configuration

//declarations is used for configure the selectors.

declarations: [

AppComponent

],

//Composability and Grouping

//imports used for composing NgModules together.

imports: [

BrowserModule

],

//Runtime or injector configuration

//providers is used for runtime injector configuration.

providers: [CookieService],

//bootstrapped entry component

bootstrap: [AppComponent]

})

export class AppModule { }

Then, import and inject it into a component -

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

import {CookieService} from 'ngx-cookie-service'

@Component({

selector: 'app-on-click',

templateUrl: './on-click.component.html',

styleUrls: ['./on-click.component.css']

})

export class OnClickComponent implements OnInit {

cookieValue ="";

constructor(private cookie:CookieService) { }

ngOnInit() {

this.cookie.set('cookie', 'demoApp' );

this.cookieValue = this.cookie.get('cookie');

}

}

**What are the cookies methods?**

Angular cookies concept is very similar to the Angular 1.x but Angular added only one extra method to delete all cookies.

The All cookie methods are

1. Check – This method is used to check the cookie existing or not.

2. Get - This method returns the value of given cookie name.

3. GetAll - This method returns a value object with all the cookies

4. Set – This method is used to set the cookies with a name.

5. Delete – This method used to delete the cookie with the given name

6. deleteAll – This method is used to delete all the cookies

7. and so on

**Cookie Methods** –

The Angular cookies service contains the following methods.

export declare class CookieService {

private document;

private documentIsAccessible;

constructor(document: any);

/\*\*

\* @param name Cookie name

\* @returns {boolean}

\*/

check(name: string): boolean;

/\*\*

\* @param name Cookie name

\* @returns {any}

\*/

get(name: string): string;

/\*\*

\* @returns {}

\*/

getAll(): {};

/\*\*

\* @param name Cookie name

\* @param value Cookie value

\* @param expires Number of days until the cookies expires or an actual `Date`

\* @param path Cookie path

\* @param domain Cookie domain

\* @param secure Secure flag

\*/

set(name: string, value: string, expires?: number | Date, path?: string, domain?: string, secure?: boolean): void;

/\*\*

\* @param name Cookie name

\* @param path Cookie path

\* @param domain Cookie domain

\*/

delete(name: string, path?: string, domain?: string): void;

/\*\*

\* @param path Cookie path

\* @param domain Cookie domain

\*/

deleteAll(path?: string, domain?: string): void;

/\*\*

\* @param name Cookie name

\* @returns {RegExp}

\*/

private getCookieRegExp(name);

}

**How to set in Angular cookies, type number values?**

**Why is Token Based Authentication more preferable Then Cookie based?**

The cookie-based authentication has been the default and the cookie-based authentication is stateful.

**What is Stateful?**

Keep and track the previously stored information which is used for a current transaction.

A stateful service based on HTTP cookies uses the HTTP transport protocol and its ability to convey cookies, used as session context.

**What are the Cookies** **Limitations?**

We can only store around 20 cookies per web server and not more than 4KB of information in each cookie and they can last indefinitely should you choose to specify the max-age attribute.

**Token Based Authentication -**

The Token-based authentication has received expansion over last few years due to RESTful Web APIs, SPA and so on.

The Token based authentication is stateless.

**What is Stateless?**

Every transaction is performed as if it was being done for the very first time and there is no previously stored information used for the current transaction.

**Token Based Authentication steps -**

A user enters their login credentials and the server verifies the entered credentials. Validating to the entered credentials, It’s correct or not. If the credentials are correct, returns a signed token.

This token is stored in local storage on the client side. We can also store in session storage or cookie.

**Advantages of Token-Based Authentication -**

1. Stateless,

2. Scalable

3. Decoupled

4. JWT is placed in the browsers local storage

5. Protect Cross Domain and CORS

6. Store Data in the JWT

7. Protect XSS and XSRF Protection

**Where to Store Tokens?**

It does depend on you, where you want to store the JWT. The JWT is placed in your browsers local storage.

# Install cookies in Angular

[Anil Singh](https://plus.google.com/116567432705413398625) [10:04 PM](https://www.code-sample.com/2018/05/install-cookies-angular-5-6-7.html)

How to install a cookie in Angular?

Install cookie -

npm install ngx-cookie-service –save

If you do not want to install this via NPM, you can run npm run compile and use the \*.d.ts and \*.js files in the dist-lib folder.

After installed successfully, add the cookie service in the Angular module - app.module.ts

import {CookieService} from 'ngx-cookie-service'

//AppModule class with @NgModule decorator

@NgModule({

//Static, this is the compiler configuration

//declarations is used for configure the selectors.

declarations: [

AppComponent

],

//Composability and Grouping

//imports used for composing NgModules together.

imports: [

BrowserModule

],

//Runtime or injector configuration

//providers is used for runtime injector configuration.

providers: [CookieService],

//bootstrapped entry component

bootstrap: [AppComponent]

})

export class AppModule { }

Then, import and inject it into a component -

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

import {CookieService} from 'ngx-cookie-service'

@Component({

selector: 'app-on-click',

templateUrl: './on-click.component.html',

styleUrls: ['./on-click.component.css']

})

export class OnClickComponent implements OnInit {

cookieValue ="";

constructor(private cookie:CookieService) { }

ngOnInit() {

this.cookie.set('cookie', 'demoApp' );

this.cookieValue = this.cookie.get('cookie');

}

}

For more detail kindly refer this link. <https://www.code-sample.com/2018/05/angular-5-6-7-cookies-examples.html>

# Angular Cookies methods

[Anil Singh](https://plus.google.com/116567432705413398625) [10:09 PM](https://www.code-sample.com/2018/05/angular-5-6-7-cookies-methods.html)

**What are the cookies methods?**

Angular cookies concept is very similar to the Angular 1.x but Angular added only one extra method to delete all cookies.

The All cookie methods are

1. Check – This method is used to check the cookie existing or not.

2. Get - This method returns the value of given cookie name.

3. GetAll - This method returns a value object with all the cookies

4. Set – This method is used to set the cookies with a name.

5. Delete – This method used to delete the cookie with the given name

6. deleteAll – This method is used to delete all the cookies

7. and so on

**Cookie Methods** –

The Angular cookies service contains the following methods.

export declare class CookieService {

private document;

private documentIsAccessible;

constructor(document: any);

/\*\*

\* @param name Cookie name

\* @returns {boolean}

\*/

check(name: string): boolean;

/\*\*

\* @param name Cookie name

\* @returns {any}

\*/

get(name: string): string;

/\*\*

\* @returns {}

\*/

getAll(): {};

/\*\*

\* @param name Cookie name

\* @param value Cookie value

\* @param expires Number of days until the cookies expires or an actual `Date`

\* @param path Cookie path

\* @param domain Cookie domain

\* @param secure Secure flag

\*/

set(name: string, value: string, expires?: number | Date, path?: string, domain?: string, secure?: boolean): void;

/\*\*

\* @param name Cookie name

\* @param path Cookie path

\* @param domain Cookie domain

\*/

delete(name: string, path?: string, domain?: string): void;

/\*\*

\* @param path Cookie path

\* @param domain Cookie domain

\*/

deleteAll(path?: string, domain?: string): void;

/\*\*

\* @param name Cookie name

\* @returns {RegExp}

\*/

private getCookieRegExp(name);

}

**For more detail kindly refer this link.** [**https://www.code-sample.com/2018/05/angular-5-6-7-cookies-examples.html**](https://www.code-sample.com/2018/05/angular-5-6-7-cookies-examples.html)

# Advantages of Token-Based Authentication - Angular 5 and 6

[Anil Singh](https://plus.google.com/116567432705413398625) [10:12 PM](https://www.code-sample.com/2018/05/advantages-of-token-based-authentication.html)

**How to set in Angular cookies, type number values?**

**Why is Token Based Authentication more preferable Then Cookie based?**

The cookie-based authentication has been the default and the cookie-based authentication is stateful.

**What is Stateful?**

Keep and track the previously stored information which is used for a current transaction.

A stateful service based on HTTP cookies uses the HTTP transport protocol and its ability to convey cookies, used as session context.

**What are the Cookies** **Limitations?**

We can only store around 20 cookies per web server and not more than 4KB of information in each cookie and they can last indefinitely should you choose to specify the max-age attribute.

**Token Based Authentication -**

The Token-based authentication has received expansion over last few years due to RESTful Web APIs, SPA and so on.

The Token based authentication is stateless.

**What is Stateless?**

Every transaction is performed as if it was being done for the very first time and there is no previously stored information used for the current transaction.

**Token Based Authentication steps -**

A user enters their login credentials and the server verifies the entered credentials. Validating to the entered credentials, It’s correct or not. If the credentials are correct, returns a signed token.

This token is stored in local storage on the client side. We can also store in session storage or cookie.

**Advantages of Token-Based Authentication -**

1. Stateless,

2. Scalable

3. Decoupled

4. JWT is placed in the browsers local storage

5. Protect Cross Domain and CORS

6. Store Data in the JWT

7. Protect XSS and XSRF Protection

**Where to Store Tokens?**

It does depend on you, where you want to store the JWT. The JWT is placed in your browsers local storage.