The core differences and many more advantages on Angular2 vs. Angular 1 as following,

1.     It is entirely component based.

2.     Better change detection

3.     Angular2 has better performance.

4.     Angular2 has more powerful template system.

5.     Angular2 provide simpler APIs, lazy loading and easier to application debugging.

6.     Angular2 much more testable

7.     Angular2 provides to nested level components.

8.     Ahead of Time compilation (AOT) improves rendering speed

9.     Angular2 execute run more than two programs at the same time.

10.  Angular1 is controllers and $scope based but Angular2 is component based.

11.  The Angular2 structural directives syntax is changed like ng-repeat is replaced with \*ngFor etc.

12.  In Angular2, local variables are defined using prefix (#) hash. You can see the below \*ngFor loop Example.

13.  TypeScript can be used for developing Angular 2 applications

14.  Better syntax and application structure  
There are more advantages over performance, template system, application debugging, testing, components and nested level components.  
**Angular 1 Controller:-**

**var** app = angular.module("userApp", []);

app.controller("productController", **function**($scope) {

$scope.users = [{ name: "Anil Singh", Age:**30**, department :"IT" },

{ name: "Aradhya Singh", Age:**3**, department :"MGMT" }];

});

**Angular 2 Components using TypeScript:-**

Here the @Component annotation is used to add the metadata to the class.

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

@Component({

selector: 'usersdata',

template: `<h3>{{users.name}}</h3>`

})

**export** **class** UsersComponent {

users = [{ name: "Anil Singh", Age:**30**, department :"IT" },

{ name: "Aradhya Singh", Age:**3**, department :"MGMT" }];

}

**Bootstrapping in Angular 1 using ng-app,**

angular.element(document).ready(**function**() {

angular.bootstrap(document, ['userApp']);

});

**Bootstrapping in Angular 2,**

**import** { bootstrap } from 'angular2/platform/browser';

**import** { UsersComponent } from './product.component';

bootstrap(UserComponent);

The Angular2 structural directives syntax is changed like **ng-repeat** is replaced with **\*ngFor** etc.

//Angular 1,

<div ng-repeat="user in users">

Name: {{user.name}}, Age : {{user.Age}}, Dept: {{user.Department}}

</div>

//Angular2,

<div \*ngFor="let user of users">

Name: {{user.name}}, Age : {{user.Age}}, Dept: {{user.Department}}

</div>

The **Angular 2** is focusing on **data-binding**, extensible **HTML**and on application test-ability but it is still in design and prototyping stage.

**Angular framework** helps us to build client applications in **HTML**and **JavaScript**.

**Angular 2** is so **simpler**, **faster**, **modular**and instrumented design.

**Angular 2** targeting to modern browsers and it is developing using **ES6**(**The ES6 is called ECMAScript version 6**). It also support to **ECMAScript**version 5(**ES5**).

You don’t worry about the versions of ECMAScript. The compiler manages to the versioning related problems.

All the **Angular 2 framework** code is already being written in ECMAScript 6.

The set of modern browsers are

1.       Chrome

2.       Firefox

3.       Opera

4.       Safari

5.       IE Version10 and 11.

On mobiles, it is supporting to the list of Chrome on **Android**, **iOS**6+, **Windows Phone** 8+ and **Fire-Fox**mobile and also trying to support to older versions of **Android**.

The **Angular 2** is using **Traceur**compiler to generate the nice **ECMAScript**5/6 that runs everywhere you want to do.

**Angular 2** team working with Traceur compiler team to provide the support to build some extensions. This set of extensions called “**ES 6 +A**”.

What is ECMAScript ES5/ES6?

The ECMAScript is a scripting language which is developed by Ecma International Org.

Currently ECMAScript available in multiple versions that are ES5 and ES6 and both of versions fully supported to Chrome,Firefox, Opera, Safari, and IE etc.

What is Traceur Compiler?

The “Traceur” is a JavaScript compiler. The Traceur compiler is very popular now days use to allow use to use the features from the future. This compiler is fully supported to ES5, ES6 and also to vNext. The main goal of Traceur compiler is to inform to design of new JavaScript features and wrote the programming code of new efficient and good manners.

**What is Advantages of Angular 2?**

1.       There is many more advantage of Angular 2.

2.       The Angular 2 has better performance.

3.       The Angular 2 has more powerful template system.

4.       The Angular 2 provide simpler APIs, lazy loading and easier to application debugging.

5.       The Angular 2 much more testable.

6.       The Angular 2 provides to nested level components.

7.       The Angular 2 execute run more than two programs at the same time.

The Angular 2 architecture diagram identifies the eight main building blocks as.

1.       [Module](https://www.code-sample.com/2017/04/angular-2-ngmodel-root-export-module.html)

2.       [Component](https://www.code-sample.com/2016/06/angular-2-template-components.html)

3.       [Template](https://www.code-sample.com/2016/06/angular-2-template-components.html)

4.       [Outpouts](https://www.code-sample.com/2016/06/angular-2-outputs.html" \t "_blank)

5.       [Data Binding](https://www.code-sample.com/2016/06/angular-2-template-components.html)

6.       [Directive](https://www.code-sample.com/2016/06/angular-2-directives-components.html)

7.       [Service](https://www.code-sample.com/2017/05/angular-2-services-singleton-examples.html)

8.       [Dependency Injection](https://www.code-sample.com/2016/04/dependency-injection-in-angular-2.html)

The Angular 2 framework consists of several libraries, the some of them working as core and some are optional.

“Could anyone tell me about the usage of ngOnInit if we already have a constructor?” but Angular 2 provides life cycle hook ngOnInit by default.

Angular 2 Components and Directives has multiple life-time hooks where we custom logic can be executed.

Angular 2 Constructors:-

The constructor is a default method runs when component is being constructed.

The constructor is a typescript feature and it is used only for a class instantiations and nothing to do with Angular 2.

The constructor called first time before the ngOnInit().

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

**import** {UserService} **from** './userService';

**@Component**({

selector: ‘list-user’,

template: `<ul><li \*ngFor="#user of users">{{user.name}}</li></ul>`

})

**class** **App\_Component** {

users:Array<any>;

constructor(private \_userService: UserService) {

this.users = \_userService.getUsers();

}}

Angular 2 ngOnInit and ngOnChanges:-

The ngOnInit event is an Angular 2 life-cycle event method that is called after the first ngOnChanges and the ngOnInit method is use to parameters defined with @Input otherwise the constructor is OK.

The ngOnInit is called after the constructor and ngOnInit is called after the first ngOnChanges.

The ngOnChanges is called when an input or output binding value changes.

**Examples as**,

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

export **class** **App** implements OnInit{

constructor(){

}

ngOnInit(){

}}

Angular 2 ngOnDestroy :-The ngDestroy directive is called in a component lifecycle just before the instance of the component is finally destroyed.

**Example as,**

@Directive({

selector: '[destroyDirective]'

})

**export** **class** OnDestroyDirective **implements** OnDestroy {

//Call Constructor and set hello Msg.

**constructor**() {

**this**.helloMsg = window.setInterval(() => alert('Hello, I am Anil'), **2000**);

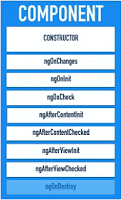
}

//Destroy to the component

ngOnDestroy() {

window.clearInterval(**this**.helloMsg);

}} **Angular 2 Complete lifecycle hook interface inventory**:-

[](https://4.bp.blogspot.com/-L-cC-e3BkgQ/WXMd72Nz29I/AAAAAAAAQVw/fTpqVG8YAjQDyuRStToJRUuSjRu2H1sTQCLcBGAs/s1600/Angular+2+%E2%80%8AComponent+Lifecycle+Hooks+-+Angular+4-min.png)

1.      ngOnChanges - called when an input binding value changes.

2.      ngOnInit - after the first ngOnChanges.

3.      ngDoCheck - after every run of change detection.

4.      ngAfterContentInit - after component content initialized.

5.      ngAfterContentChecked - after every check of component content.

6.      ngAfterViewInit - after component's view(s) are initialized.

7.      ngAfterViewChecked - after every check of a component's view(s).

8.      ngOnDestroy - just before the component is destroyed.

Angular 2 Lifecycle Events Log:-

1.      onChanges

2.      onInit

3.      doCheck

4.      afterContentInit

5.      afterContentChecked

6.      afterViewInit

7.      afterViewChecked

8.      doCheck

9.      afterContentChecked

10. afterViewChecked

11. onChanges

12. doCheck

13. afterContentChecked

14. afterViewChecked

15. onDestroy

When will ngInit be called? How would you make use of ngOnInit()?

In Angular 1.x, ngInit is called when template is re-rendered. In other words “ng-init” is called, when I take turns back to a page.

In Angular2, there is no “ng-init” but we can create a ways like this using the directive and ngOnInit class. Angular 2 provides life cycle hook ngOnInit by default.

The ngOnInit is invoked when the component is initialized and invoked only once when the directive is instantiated. It is a best practice to implement these life-cycle interfaces.  
According to Angular2 Doc, “The ngOnInit is called right after the directive's data-bound properties have been checked for the first time, and before any of its children have been checked. It is invoked only once when the directive is instantiated.”

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

@Directive({

selector: '[ngInit]' })

class NgInit {

@Input() ngInit;

ngOnInit() {

if(this.ngInit) { this.ngInit(); }

}}  
In template as following,

<div \*ngIf="Timer.dateTime === currentDateTime">

<div \*ngIf="Timer.checked" [ngInit]="Start"></div>

<div \*ngIf="!Timer.checked" [ngInit]="Stop"></div>

</div>

Angular2 cookies | angular2 http cookies | angular2 http get set cookie

Angular 2 cookies concept is very similar to the Angular 1.x but Angular2 added only one extra method to remove all cookies i.e. removeAll()

The All cookie methods are

1.                [get()](https://www.code-sample.com/2016/04/angular-2-cookies-npm-example.html)

2.                getObject()

3.                getAll()

4.                put()

5.                putObject()

6.                remove()

7.                removeAll() - This is new one in angular 2

get() :- This method is returns the value of given cookie key.

getObject() :- This method is returns the desterilized value of given cookie key.

getAll() :- This method is returns a key value object with all the cookies.

put() :- This method is use to set a value for given cookie key.

putObject() :- This method is use to serializes and set a value for given cookie key.

remove() :-This method is use to remove given cookie.

**removeAll()** :-This method is use to remove all cookies.  
You can create your own functions to get cookie value, set cookie value and delete cookie value.

Two ways to create cookies -1.     You can inject a service in the components providers.

2.     You can get it via npm.

To install ng2-cookies library, run the below given code i.e.      
npm install angular2-cookie --save  
You can include angular2-cookie library for the same which has given below.

<script src="~/cookie/angular2-cookie.min.js"></script>  
Example - You can get it via npm

The full example for creating your own functions as,

@Component({

selector: 'cookie-consent',

template: cookieconsent\_html,

styles: [cookieconsent\_css]

})

**export** **class** CookieConsent {

**private** isConsented: **boolean** = **false**;

constructor() {

**this**.isConsented = **this**.getCookie(COOKIE\_CONSENT) === "1";

}

**private** getCookie(name: string) {

**let** ca: Array<string> = document.cookie.split(';');

**let** caLen: number = ca.length;

**let** cookieName = name + "=";

**let** c: string;

**for** (**let** i: number = **0**; i < caLen; i += **1**) {

c = ca[i].replace(/^\s\+/g, "");

**if** (c.indexOf(cookieName) == **0**) {

**return** c.substring(cookieName.length, c.length);

} }

**return** "";

}

**private** deleteCookie(name) {

**this**.setCookie(name, "", -**1**);

}

**private** setCookie(name: string, value: string, expireDays: number, path: string = "") {

**let** d:Date = **new** Date();

d.setTime(d.getTime() + expireDays \* **24** \* **60** \* **60** \* **1000**);

**let** expires:string = "expires=" + d.toUTCString();

document.cookie = name + "=" + value + "; " + expires + (path.length > **0** ? "; path=" + path : "");

}

**private** consent(isConsent: **boolean**, e: any) {

**if** (!isConsent) {

**return** **this**.isConsented;

} **else** **if** (isConsent) {

**this**.setCookie(COOKIE\_CONSENT, "1", COOKIE\_CONSENT\_EXPIRE\_DAYS);

**this**.isConsented = **true**;

e.preventDefault();

} }}

Example 2 - You can inject a service in the components providers

//Use of Angular 2 cookies, the example in detail as give below.

**import** {Component} from '@angular2/core';

**import** {Cookie} from '@angular2-cookie/core';

@Component({

selector: 'my-cookie-app',

template: '<div>Cookies in Angular 2</div>',

providers: [Cookie]

})

**export** **class** App\_Component {

constructor(**private** \_cookie:Cookie){}

getCookie(key: string){

**return** **this**.\_cookie.get(key);

}

getCookieObject(key: string){

**return** **this**.\_cookie.getObject(key);

}}

//And other are available methods [put(), putObject(), remove() and removeAll()]

//All methods work similar like above methods.

**What is AOT compilation? Why Use in Angular 2?**

AOT compilation stands for “Ahead of Time compilation” and it are used to compiles the angular components and templates to native JavaScript and HTML during the build time instead of run-time.

The compiled HTML and JavaScript are deployed to the web server so that the compilation and render time can be saved by the browser. It is the big advantage to improve the performance of applications.

Advantages of AOT -

1. Faster download: - The Angular 2 app is already compiled so it is faster.

2.Faster Rendering: - If the app is not AOT compiled and the compilation process happens in the browser once the application is fully loaded. This has a wait time for all necessary components to be downloaded and then the time taken by the compiler to compile the app. With AOT compilation, this is optimized.

3.   Lesser Http Requests: - It is supporting to the lazy loading. Actually, lazy loading is great concepts for sending HTTP request to the server. It is minimise the multiple requests for each associated html and css, there is a separate request goes to the server.

4.  Detect error at build time: - In Angular 2, the compilation happens beforehand and most of the errors can be detected at the compile time and this process providing us a better application’s stability.

Disadvantages of AOT -

1.AOT only works only with HTML and CSS and not for other file types. If required other file types that time we will need to follow the previous build step.

2.  We need to maintain AOT version of bootstrap file.

3.   We need to clean-up step before compiling.

**What is lazy loading and How to enable lazy loading in angular 2?**

Lazy Loading - Lazy loading enables us to load only the module user is interacting and keep the rest to be loaded at run-time on demand.

Lazy loading speeds up the application initial load time by splitting the code into multiple bundles and loading them on demand.

1.         Each and every Angular2 application must have one main module that is called “AppModule” and your code should be splitted into various child modules based on your applications.

2.         We do not require to import or declare lazily loading module in root module.

3.         Add the route to top level routing and takes routes array and configures the router.

4.         Import module specific routing in the child module. and so on.

How would you Optimize the Angular 2 Application for Better Performance?

The optimizations are depends on the size of applications, type and other factors but normally we consider following optimizing points i.e.

1.         Consider AOT compilation.

2.         Consider lazy loading instead of fully bundled app if the app size is more.

3.         Keep in mind, your application is bundled and disfeatured.

4.         Keep in mind, your application doesn’t have un-necessary import statements.

5.         Keep in mind, your application’s 3rd party unused library. If exist and not used, removed from your application.

6.         Remove your application dependencies if not required.

# Pipes

**What is Pipes?**

“Pipes transform displayed values within a template.”

Sometimes, the data is not displays in the well format on the template that time where using pipes.

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

The angular 2 have some additional pipes names that are async, decimal, percept and so on. And also some of pipes not supported in angular 2 that are number, orderBy and filter and these are archiving using “custom pipes”.

**Key Points:-**

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 “@Pipe” decorator allows us to define the pipe name that is globally available for use in any template in the across application.

**For example as,**

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

@Pipe({

**name: 'barcode',**

    pure: false })

export class BarCodePipe implements PipeTransform {

    transform(value: string, args: any[]): string {

        if (!value) {

            return '';        }

        return "\*\*\*\*-\*\*\*\*\_" + (value.length > 8 ? (value.length - 8): '')

    }}

**Angular 2 Built-in Pipes:-**

1.      DatePipe,

2.      UpperCasePipe,

3.      LowerCasePipe,

4.      CurrencyPipe,

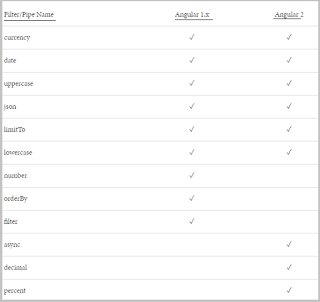
5.      PercentPipe,

6.      JsonPipe,

7.      AsyncPipe,

8.      And so on..

The following table shows a comparison between Angular 1.x and Angular 2.

[](https://3.bp.blogspot.com/-0i_-3dMmKaw/WSfEeUleGcI/AAAAAAAAPUY/6Ov4AjTQYXkDdpbImfCxhqCps2fNvoGaQCLcB/s1600/Angular+2+Async+Pipes+and+Example.png)

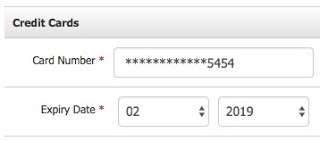
**Why use Pipes?**

Sometimes, the data is not displays in the correct format on the template that time where using pipes.

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

**For example as,**

If you want to display the bank card number on your account detail templates that how to displays this card number?  I think you should display the last four digits and rest of all digits will display as encrypted like (\*\*\*\*-\*\*\*\*-\*\*\*\*\_and your card numbers) that time you will need to create a custom pipe to achieve this.

[](https://4.bp.blogspot.com/-2g_8yna8PuE/WSfEl4HKPFI/AAAAAAAAPUc/0705_voTYSo5gjjoz7M3hJZm8YmJ5PcKgCLcB/s1600/Add-Credit-Card.jpg)

**What is a pure and impure pipe?**

In Angular 2, there are two types of pipes i.e.

1.      pure

2.      impure

The pure pipe is by default. 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.

**Pure Pipes:-**

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.

**For example for pipe pure,**

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

@Pipe({    name: 'barcode'})

export class BarCodePipe implements PipeTransform {

    transform(value: string, args: any[]): string {

        if (!value) {

            return '';

        }

        return "\*\*\*\*-\*\*\*\*\_" + (value.length > 8 ? (value.length - 8): '')

    }}

**Impure Pipes:-**

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.

**For example for pipe impure,**

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

@Pipe({

    name: 'barcode',

    pure: false

})

export class BarCodePipe implements PipeTransform {

    transform(value: string, args: any[]): string {

        if (!value) {

            return '';

        }

        return "\*\*\*\*-\*\*\*\*\_" + (value.length > 8 ? (value.length - 8): '')

    }}

**What is Async Pipe?**

Angular 2 provides us special kinds of pipe that is called Async pipe and the Async pipe subscribes to an Observable or Promise and returns the latest value it has emitted.

The Async pipe allows us to bind our templates directly to values that arrive asynchronously manner and this is the great ability for the promises and observables.

**Example for AsyncPipe with Promise using NgFor,**

@Component({

    selector: 'app-promise',

    template: '<ul> < li \* ngFor="let user of users | async">  Id: {{user.id }}, Name: {{user.name }} </li>< /ul>'

})

export class PromiseComponent {

    //USERS DECLARATIONS.

    users = [];

    //FETCHING JSON DATA FROM REST APIS

    userRestApiUrl: string = 'https://api.github.com/users/hadley/orgs';

    //HOME COMPONENT CONSTRUCTOR

    constructor(private userService: UserService) { }

    //GET USERS SERVICE ON PAGE LOAD.

    ngOnInit() {

        this.userService.getUsers(this.userRestApiUrl).subscribe(data => this.users = data);

    }}

**How to create a custom Pipes?**

**How to create a globally available custom “Pipe”?**

The “@Pipe” decorator allows us to define the pipe name that is globally available for use in any template in the across application.

**Steps for Creating a Custom Pipe:-**

1.      Create a typescript class.

2.      Decorate the class using @Pipe.

3.      Implement PipeTransform interface.

4.      Override transform() method.

5.      Configure the class in application module with @NgModule.

6.      Ready to use our custom pipe anywhere in application.

**In the below example**,

I am using the custom pipe in the user temple to display our custom “Ids” values at the place of Id.

**user.component.ts :-**

import { UserService } from '../shared/service/user.service';

import { BarCodePipe } from '../shared/pipe/custom.barcode.pipe';

@Component({

    selector: 'user',

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

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

})

export class UserComponent {

    //USERS DECLARATIONS.

    users = [];

    //FETCHING JSON DATA FROM REST APIS

    userRestApiUrl: string = 'https://api.github.com/users/hadley/orgs';

    //HOME COMPONENT CONSTRUCTOR

    constructor(private userService: UserService) {  }

    //GET USERS SERVICE ON PAGE LOAD.

    ngOnInit() {

        this.userService.getUsers(this.userRestApiUrl).subscribe(data => this.users = data);

    }

}

//END BEGIN - USERCOMPONENT

**user.service.ts :-**

//BEGIN-REGION - USERSERVICE

@Injectable()

export class UserService {

    constructor(private \_http: Http) {

    }

    getUsers(apiUrl) {

        return this.\_http.get(apiUrl).map((data: Response) => data.json());

    }}

//END BEGIN – USERSERVICE

**custom.barcode.pipe.ts :-**

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

@Pipe({

    name: 'barcode',

    pure: false

})

export class BarCodePipe implements PipeTransform {

    transform(value: string, args: any[]): string {

        if (!value) {

            return '';

        }

        return "....-" + (value.length > 2 ? (value.length - 2) : '')

    }

}

**app.module.ts :-**

    bootstrap: [ AppComponent ],

    declarations: [

        AppComponent,

        UserComponent,

        HeaderComponent,

        MenuComponent,

        LoginComponent,

        RegistrationComponent,

        BarCodePipe,

        MyPipePipe

    ],

    imports: [

        UniversalModule, // MUST BE FIRST IMPORT. THIS AUTOMATICALLY IMPORTS BROWSERMODULE, HTTPMODULE, AND JSONPMODULE TOO.

        RouterModule.forRoot([ //RouterModule.forRoot method in the module imports to configure the router.

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

            { path: 'user/:id', component: UserComponent }, //HERE ID IS A ROUTE PARAMETER.

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

            { path: 'registration', component: RegistrationComponent },

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

        ]),

        FormsModule,

        ReactiveFormsModule

    ],

    providers: [UserService]

})

export class AppModule {

}

**user.component.html :-**

<div class="row">

<div class="col-lg-12">

    <div class="ibox float-e-margins">

        <div class="ibox-title">

            <h2>Angular 2 - User Services</h2>

        </div>

        <hr />

        <div class="ibox-content">

            <div class="table-responsive">

                <table class="table table-striped">

                    <thead>

                        <tr>

                            <th>ID</th>

                            <th>Name </th>

                            <th>Description </th>

                            <th>URls </th>

                        </tr>

                    </thead>

                    <tbody>

                   <tr \*ngFor="let user of users; let i = index" class="tbl-row-border">

                           <td>**{{**user.id | barcode: true**}}**</td>

                            <td>**{{**user.login**}}**</td>

                            <td>**{{**user.description**}}**</td>

                            <td><a href="**{{**user.public\_members\_url**}}**" target="\_blank">**{{**user.public\_members\_url**}}**</a></td>

                        </tr>

                    </tbody>

# Directives

 There are 3 types of directives in Angular 2.

1.     Components Directives - directives with a template

2.     Structural Directives - change the DOM layout by adding and removing DOM elements.

3.     Attribute Directives - change the appearance or behavior of an element, component, or other directive.

What are components directives?

A component is a directive with a template and the @Component decorator is actually a @Directive decorator extended with template oriented features.

1.     To register a component, we use @Component meta-data annotation.

2.     The directives are used to add behavior to existing DOM elements.

3.     The directives are used to design a reusable component.

4.     Only one component can be present per DOM element.

5.     Multiple directives are used per DOM element.

6.     The directive does not have @View etc.

What are structural directives?

The Structural directives are responsible for HTML layout and It is using Angular 2 for reshape the DOM's structure and also removing, or manipulating elements.

What are attribute directives?

Attribute directives are used to change the behavior, appearance or look of an element on a user input or via data from the service.

For example as,

**import** {Component, View} **from** 'angular2/core'';

**@Component**({

selector: 'user-detail'

})

**@View**({

template: "<div> <h1>{{userName}}</h1> <p>{{phone}}</p>"

})

**class** **userDetail** {

constructor(public userName: string, public phone: string) {}

}

<user-detail></user-detail>

# Template

A template is a HTML view that tells Angular 2 for render your components in the views.

The Angular 2 templates are very similar to Angular 1 but Angular 2 has some small syntactical changes.                           
You can see the changes as below,

1.         {}: Is use to rendering the HTML elements.

2.         []: Is use to binding properties.

3.         (): Is use to handling your events.

4.         [()]: Is use to data binding.

5.         \*:  Is use to asterisk Operations like \*ngFor="let item of items; let i=index;”

The templates can be inline or external separate files.  
**How to use {}, [], [] and [()] in Angular2 Template?**

Here, I am using “Inline Template” in the user components i.e.

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

@Component({

selector: 'Users',

template: `<div>

<input (keyup)="onKey($event)" (click)="onClick()"/>

<div [hidden]="isActive" **class**="info">

<h2>Active element or Not?</h2>

<div>{{values}}</div>

</div>

</div>`,

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

})

**export** **class** UsersComponent {

values: **string**;

isActive: **boolean** = **false**;

onKey(event) {

**this**.isActive = **true**;

**this**.values += event.target.value;

}}

What are differences of using template and templateUrl in Angular 2 Component?

Angular 2 template vs. templateUrl?

When using template vs. templateUrl?

Inline templates are specified directly in the component using template and it is more complex for bigger templates. As per expert suggestions, use templates and styles into a separate file, when your code more than 5 to 10 lines.

External templates define the HTML in a separate file and reference this file in templateUrl.

To use a relative path in the templateUrl we must include import component form @angular/core

Some befits for template Urls i.e.

1.               Separations of code

2.               Easy debugging

The upcoming offline template compiler will inline templates linked by templateUrl.

Example for Inline Template -

@Component({

selector: 'users-app',

template: `<div \*ngFor="let user of users; let i = index">

<div>{{user.id }}</div>

<div>{{user.id | barcodepipe:**true**}}</div>

<div>{{user.login}}</div>

<div>{{user.description}}</div>

<div><a href="{{user.public\_members\_url}}" target="\_blank">{{user.public\_members\_url}}</a></div>

</div>`,

styleUrls: ['./user.component.css'],

providers: [UserService, AppGlobals],

})

**export** **class** UsersApp {

//USERS DECLARATIONS.

users = [];

//USER COMPONENT CONSTRUCTOR.

**constructor**(**private** \_userService: **UserService**,

**private** \_global: **AppGlobals**) { }

//GET USERS SERVICE ON PAGE LOAD and BIND UI GRID.

ngOnInit() {

**this**.\_userService.getAPIUsers(**this**.\_global.baseAPIUrl + 'users/api/GetUsers').subscribe(data => **this**.users = data);

}}

**Example for external templates - Separate file-**

@Component({

selector: 'users-app',

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

styleUrls: ['./user.component.css'],

providers: [UserService, AppGlobals],})

**export** **class** UsersApp {

//USERS DECLARATIONS.

users = [];

//USER COMPONENT CONSTRUCTOR.

**constructor**(**private** \_userService: **UserService**,

**private** \_global: **AppGlobals**) { }

//GET USERS SERVICE ON PAGE LOAD and BIND UI GRID.

ngOnInit() {

**this**.\_userService.getAPIUsers(**this**.\_global.baseAPIUrl + 'users/api/GetUsers').subscribe(data => **this**.users = data); }}

# **Routing**

“The Router is use to map applications URLs to application components. There are three main components that you are using to configure routing.”

1.           **Routes**: - It uses to describe our application's Routes.

2.           **Router Imports: -**It uses to import our application's Routes.

3.           **RouterOutlet**: - It is a placeholder component and use to get expanded to each route's content.

4.           **RouterLink**: - It is use to link to application's routes.

**Routes: -** The Routes is uses to describe our application's Routes. The “RouterModule.forRoot” method in the module imports to configure the router.

Five concepts that need Routes Representation

1.           Path (a part of the URL)

2.           Route Parameters

3.           Query/Matrix Parameters

4.           Name outlets

5.           A tree of route segments targeting outlets

**Syntax:-**

RouterModule.forRoot([

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

            { path: 'home/:id', component: HomeComponent }, //HERE ID IS A ROUTE PARAMETER.

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

            { path: 'registration', component: RegistrationComponent },

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

        ])

**Example,**

@NgModule({

    bootstrap: [ AppComponent ],

    declarations: [ AppComponent,  HomeComponent,  HeaderComponent, MenuComponent,        LoginComponent,        RegistrationComponent    ],

    imports: [        UniversalModule, // MUST BE FIRST IMPORT. THIS AUTOMATICALLY IMPORTS BROWSERMODULE, HTTPMODULE, AND JSONPMODULE TOO.

        RouterModule.forRoot([ //RouterModule.forRoot method in the module imports to configure the router.

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

            { path: 'home/:id', component: HomeComponent }, //HERE ID IS A ROUTE PARAMETER.

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

            { path: 'registration', component: RegistrationComponent },

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

        ]),

        FormsModule,      ReactiveFormsModule    ]})

**Router Imports** - The Angular Router is an optional service that presents a particular component view for a given URL i.e.

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

Example,

@NgModule({

    bootstrap: [ AppComponent ],

    declarations: [

        AppComponent, HomeComponent,HeaderComponent,MenuComponent,LoginComponent, RegistrationComponent],

    imports: [UniversalModule, // MUST BE FIRST IMPORT. THIS AUTOMATICALLY IMPORTS BROWSERMODULE, HTTPMODULE, AND JSONPMODULE TOO.

        RouterModule.forRoot([ //RouterModule.forRoot method in the module imports to configure the router.

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

            { path: 'home/:id', component: HomeComponent }, //HERE ID IS A ROUTE PARAMETER.

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

            { path: 'registration', component: RegistrationComponent },

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

        ]),

        FormsModule,

        ReactiveFormsModule

    ]})

Router-outlet directive: - Router-outlet directive is used to render the components for specific location of your applications. Both the template and templateUrl render the components where you use this directive.

Syntax :- <router-outlet></router-outlet>

Example

<div class='container'>

    <div class='row'>

        <router-outlet></router-outlet>

    </div></div>

The Route Params: - The route parameter is used to map given URL's parameters based on the rout URLs and it is an optional parameters for that route.

Syntax: -   params: {[key: string]: string}

Example

@NgModule({

    bootstrap: [ AppComponent ],

    declarations: [    AppComponent,

        HomeComponent,

        HeaderComponent,

        MenuComponent,

        LoginComponent,

        RegistrationComponent

    ],

    imports: [

        UniversalModule, // MUST BE FIRST IMPORT. THIS AUTOMATICALLY IMPORTS BROWSERMODULE, HTTPMODULE, AND JSONPMODULE TOO.

        RouterModule.forRoot([ //ROUTERMODULE.FORROOT METHOD IN THE MODULE IMPORTS TO CONFIGURE THE ROUTER.

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

            { path: 'home/:id', component: HomeComponent }, //HERE ID IS A ROUTE PARAMETER.

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

            { path: 'registration', component: RegistrationComponent },

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

        ]),

        FormsModule,

        ReactiveFormsModule    ]})

Router-link directive: - Router-link directive is used to link a specific part of your applications.

Syntax :- <router-link></router-link>

Example,

<ul class='nav navbar-nav'>

    <li [routerLinkActive]="['link-active']">

        <a [routerLink]="['/login']">

            <span class='glyphicon glyphicon-Login'></span> Login

        </a>

    </li>

    <li [routerLinkActive]="['link-active']">

        <a [routerLink]="['/registration']">

            <span class='glyphicon glyphicon-Register'></span> Register

        </a>

    </li>

    <li [routerLinkActive]="['link-active']">

        <a [routerLink]="['/Billing']">

            <span class='glyphicon glyphicon-Billing'></span> Billing

        </a>

    </li>

</ul>

Angular 2 Dependency Injection consists of three things.

**1.       Injector**

**2.       Provider**

**3.       Dependency**

**Injector** :- The injector object use to create instances of dependencies.

**Provider** :- A provider is help to injector for create an instance of a dependency. A provider takes a token and maps that to a factory function that creates an object.

**Dependency** :- A dependency is the type of which an object should be created.

**@Injectable**() 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.

**How to use Dependency Injection (DI) correctly in Angular 2?**

The basics Steps of Dependency injection,

1.      A class with **@Injectable()** to tell angular 2 that it’s to be injected “UserService”.

2.      A class with a constructor that accepts a type to be injected.

Example, UserService marked as @Injectable as,

**import** {Injectable, bind} from 'angular2/core';

**import** {Http} from 'angular2/http';

@Injectable() /\* This is #Step 1 \*/

**export** **class** UserService {

**constructor**(http: **URL** /\* This is #Step 2 \*/ ) {

**this**.http = URL;

}}

Example as,

**import** {Injectable} from "@angular/core";

@Injectable()

**export** **class** InjectToService {

id: **string**;

**constructor**() {

**this**.resetPasscode();

}

resetPasscode(): **void** {

**this**.id = **this**.generatePasscode();

}

**private** generatePasscode(): **string** {

**var** date = **new** Date().getTime();

**var** pascode = '00X000-00000-7000-Z0000-00000'.replace(/[xy]/, **function**(f) {

**var** random = (date + Math.random() \* **16**) % **16** | **0**;

date = Math.floor(date / **16**);

**return** (f == '0' ? random : (random & **0x3** | **0x8**)).toString(**16**);

});

**return** pascode;

};}

# **NgModel**

The @NgModule is a new decorator. This module is recently added in Angular 2.

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

The @NgModules page guides you from the most elementary @NgModule to a multi-facetedsample with lazy modules.

Stayed Informed – [Angular 2 vs. Angular 1](https://www.code-sample.com/2016/06/angular-2-vs-angular-1-performance.html)

Stayed Informed - [Angular 4 vs. Angular 2](https://www.code-sample.com/2017/03/angular-4-vs-angular-2-difference.html)

The @NgModule main use to simplify the way you define and manage the dependencies in your applications and using @NgModule you can consolidate different components and services into cohesive blocks of functionality.

The Basic Example of @NgModule as,

@NgModule({

imports: [BrowserModule],

declarations: [YourComponent],

bootstrap: [YourComponent]

})

class YourAppModule {}

The @NgModule is a way to organize your dependencies for

1.      Compiler

2.      Dependency Injection

The declarations of @NgModule.declarations as,

@NgModule({

declarations: [ AppComponent, YourComponent, YourDirective, YourPipe,

...OTHER DIRECTIVES AND SO ON.

]})

The @NgModule providers as,

@NgModule({

providers: [ YourService, SomeLibraryService, ],})

The @NgModule exporting as,

@NgModule({

declarations: [YourComponent, YourPipe]

exports: [YourComponent, YourPipe],

providers: [YourService]

})

**export** **class** YourModule { }

Why Angular 2 modules needed?

An Angular @NgModule allows us to define a context for compiling templates.

Why @NgModule?

1.      Easy to use Components

2.      Easy to use Directives

3.      Easy to use Pipes

4.      Providers’ Inheritance

5.      Library Architecture

6.      Easy to migrate from angular.module()

7.      So on

What is a Root Module?

Each application only has one root module and each component, directive and pipe should only be associated to a single module. This one is the main reason.

How Should We Organize Modules?

There are no standard ways to group modules, but the recommendations are,

1.      Features as a module

2.      Shared utilities as a module

Module’s Features:-

For example, suppose that your application has customer, product and feature. Each module has some components, directives and pipes.

Module’s Utility:-

For the functions or features that can be shared across modules and application, consider creating a shared module.

How to declaration Module?

@NgModule({

declarations: [AppComponent],

imports: [BrowserModule, CommonModule, FormsModule, MaterialModule],

entryComponents: [AppComponent]

})

**class** AppModule {

**constructor**(appRef: **ApplicationRef**) {

appRef.bootstrap(AppComponent);

}}

//Bootstrapping

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

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

platformBrowserDynamic().bootstrapModule(AppModule);

@NgModule

**class** NgModule {

declarations: **Array**<ComponentType | DirectiveType | PipeType>;

imports: **Array**<ModuleType | ModuleWithProviders>;

exports: **Array**<ComponentType | DirectiveType | PipeType | ModuleType>;

providers: **Array**<Providers | Array<any> >;

entryComponents: **Array**<ComponentType>;

schemas: **Array**<any>;

}

What is One Root Module?

When we create an Angular 2 app, we define a root module. We learned about this in the previous post. This root module is defined with @NgModule and works quite well for small apps.

// app.module.ts

@NgModule({

imports: [BrowserModule, FormsModule, HttpModule],

declarations: [ AppComponent, VehicleListComponent, VehicleSelectionDirective, VehicleSortingPipe ],

providers: [ LoggerService, VehicleService, UserProfileService ],

bootstrap: [AppComponent],

})

**export** **class** AppModule { }

Our root module declares our components, pipes and directives.

Our root module imports common features from the Angular 2 BrowserModule, FormsModule, and HttpModule.

Final Conclusions are,

1.      The Use of NgModule.providers

a.     Remove Component.providers

2.      Use NgModule.declarations

a.     Remove Component.directives/pipes

3.      Keep a single scope

4.      Use modules

a.     Http, Forms, Router, and so on.

5.      Make modules

6.      Module as a Library

# Services

What is an Angular 2 Service?

Angular 2 service is a class that encapsulates some methods (GET/POST/PUT) and provides it result as a service for across your application.

What are the features of Angular 2 Service?

The Angular 2 is using services concept and it provide the multiple features to us that are,

1.      Services are singleton objects.

2.      Services are capable of returning the data in the form promises or observables.

3.      Service class is decorated with Injectable decorator.

4.      The Injectable decorator is required only if our service class is making use of some Angular injectable like Http, Response and HttpModule service within it.

What are the differences between Observables & Promises?

1.      Promise:- Promises are only called once and It can return only a single value at a time and the Promises are not cancellable.

2.      Observables:- Observables handle multiple values over time and it can return multiple values and the Observables are cancellable.

3.      The Observables are more advanced than Promises.

Steps for creating an Angular 2 Service:-

There are four steps as,

1.      Import the injectable member i.e.

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

2.      Import the HttpModule, Http and Response members’ i.e.

import { HttpModule, Http, Response } from '@angular/http';

3.      Add the @Injectable Decorator i.e. @Injectable()

4.      Export to the Service class i.e.

export class UserService {

       constructor(private \_http: Http) {  }

   }

Steps for Calling an Angular 2 Service in the Angular 2 Component class:-

There are four steps to calling a service in component as,

1.      Create or Import the Service to the component class.

2.      Add it as a component provider.

3.      Include it through Dependency Injection.

4.      Use the Service function in the component.

In the below Example,

I hope this will help you to understand and create the basic of Angular 2 service. I am creating a user service and this user service returns the list of users.

After creating user service, I will use the user service “getUsers()” method in the user component’s ngOnInit() method to load the returns user collections on user screen.

I am also using the REST API Url (<https://api.github.com/users/hadley/orgs>) and this RESTful API will returns the users.

app.module.ts :-

@NgModule({

    bootstrap: [ AppComponent ],

    declarations: [ AppComponent, UserComponent, HeaderComponent,MenuComponent,

        LoginComponent,RegistrationComponent ],

    imports: [UniversalModule, // MUST BE FIRST IMPORT. THIS AUTOMATICALLY IMPORTS BROWSERMODULE, HTTPMODULE, AND JSONPMODULE TOO.

        RouterModule.forRoot([ //RouterModule.forRoot method in the module imports to configure the router.

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

            { path: 'user/:id', component: UserComponent }, //HERE ID IS A ROUTE PARAMETER.

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

            { path: 'registration', component: RegistrationComponent },

            { path: '\*\*', redirectTo: 'user' }        ]),

        FormsModule,        ReactiveFormsModule           ]

})

export class AppModule {

}

user.component.ts and user.service.ts :-

//BEGIN-REGION - USERSERVICE

@Injectable()

export class UserService {

     constructor(private \_http: Http) {  }

    getUsers(apiUrl) {

        return this.\_http.get(apiUrl).map((data: Response) => data.json());

    }}

//END BEGIN - USERSERVICE

//BEGIN-REGION - USERCOMPONENT

@Component({

    selector: 'user',

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

    styleUrls: ['./user.component.css'],

    providers: [UserService]

})

export class UserComponent {

    //USERS DECLARATIONS.

    users = [];

    //FETCHING JSON DATA FROM REST APIS

    userRestApiUrl: string = 'https://api.github.com/users/hadley/orgs';

    //HOME COMPONENT CONSTRUCTOR

    constructor(private userService: UserService) {  }

    //GET USERS SERVICE ON PAGE LOAD.

    ngOnInit() {

        this.userService.getUsers(this.userRestApiUrl).subscribe(data => this.users = data);

    }}

//END BEGIN – USERCOMPONENT

user.component.html :-

<div class="row">

<div class="col-lg-12">

    <div class="ibox float-e-margins">

        <div class="ibox-title">

            <h2>Angular 2 - User Services</h2>

        </div>        <hr />

        <div class="ibox-content">            <div class="table-responsive">

                <table class="table table-striped">             <thead>

                        <tr>   <th>ID</th>   <th>Name </th>   <th>Description </th>  <th>URls </th>            </tr>

                    </thead>                  <tbody>

                        <tr \*ngFor="let user of users; let i = index">

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

                            <td>{{user.login}}</td>

                            <td>{{user.description}}</td>

                            <td><a href="{{user.public\_members\_url}}"> {{user.public\_members\_url}}</a></td