1. **What is Angular 2? 13 Best Advantages for Angular 2!**

# Introduction to Angular 2 [A Most Popular JS Framework]

# What Are the New Features of Angular 2? Why You Used Angular 2?

**Ans:**

* Angular 2 is Entirely Component Based

       Directives

       Dependency Injection

       Used of TypeScript

       Used of Lambdas or Arrow functions

       Generics

       Forms and Validations

       [And So on.....](https://www.code-sample.com/2016/06/angular-2-vs-angular-1-performance.html)

# 13 Best Advantages for Angular2 - [Angular 2 vs. Angular 1]

Ans: 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

# 5. Angular 2 - What is Lazy Loading and How to enable Lazy Loading?

**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.

5.         And so on.

# What are the Securities Threats should we be Aware of in Angular 2 Applications?

There are some basic guidelines to mitigate the security risks.

1.     Consider using AOT compilation.

2.     Try to avoid using or injecting dynamic HTML content to your component.

3.     Try to avoid using external URLs if not trusted.

4.     Try to prevent XSRF attack by restricting the REST APIs.

# If you are using external resources like HTML, CSS, which is coming from outside the application in case you follow best practice/cleanly your apps.

# 13 Best Advantages for Angular2 - [Angular 2 vs. Angular 1]

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

# 7. Best Key Differences - Constructor Vs. ngOnInit [Angular 2]

**Ans**: **Angular 2 Constructors: -**

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

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

3.      The constructor called first time before the **ngOnInit**().

**Angular 2 ngOnInit**:-

1.      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**.

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

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

**Example as,**

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

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

**constructor**(){

}

ngOnInit(){

}

}

[When will ngInit be called? How would you make use of onNgInit()?](https://www.code-sample.com/2017/02/angular-2-ngoninit-and-ng-init.html)

# What's New in Angular 4? [Angular 4 New Features]

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

# What's New In Angular 5? [Angular 4 vs. Angular 5]

The **Angular 5 Contains** bunch of new features, performance improvements and lot of bug fixes and also some surprises to Angular lovers.

1.     Make AOT the default

2.     Watch mode

3.     Type checking in templates

4.     More flexible metadata

5.     Remove \*.ngfactory.ts files

6.     Better error messages

7.     Smooth upgrades

8.     Tree-Shakeable components

9.     Hybrid Upgrade Application

10. And so on...

# Angular 2 components vs directives

# Ans: Angular 2 components vs directives

|  |  |
| --- | --- |
| [**@Components**](https://www.code-sample.com/2016/04/angular-2-components-vs-directives.html) | [**@Directive**](https://www.code-sample.com/2016/04/angular-2-components-vs-directives.html) |
| 1.       @Component meta-data annotation is used to register the components. | @Directive meta-data annotation is used to register the directives. |
| 2.       The components are used to create UI widgets. | The directives are used to add behavior to existing DOM elements. |
| 3.       The components are used to split to application into smaller parts. | The directives are use to design a reusable components. |
| 4.       Only one component is used per DOM element. | More than one directive are used per DOM element. |
| 5.       In the components, @View, template and templateUrl are mandatory in the components. | The directive do not have @View etc. |

# Angular 2 @Inputs - How to Passing data into Angular 2 components with @Input?

# Ans:

@**Input** allows you to pass data into your controller and templates through html and defining custom properties.  
**@Input**  is used to define an input for a component, we use the @Input decorator.

Angular 2 components is the core components of applications but you must need to know “**how to pass data into components to dynamically**?” and that time you need to define an input component.

You can see the below example for passing the user data in to the components.

**Example 1,**

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

@Component({

selector: “user-info”,

template: “<div> Hello, This is {{ userInfo.name}}</div>”

})

**export** **class** UserInfo {

@Input() userInfo;

**constructor**() { }

}

<user-info [userInfo]="currentUser"></user-info>

The components **<user-info></user-info>** is use to render the user information on the view.

**Example 2,**

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

@Component({

selector: 'app-root',

styles: [`

.app {

text-align: **center**;

background: #f5f5f5;

}

`],

template: `

<div **class**="app">

<counter [count]="defaultCount"></counter>

</div>

`

})

**export** **class** AppComponent {

defaultCount: **number** = **20**;

}

# Angular 2 Component Outputs [@Output Property]

# Ans:

**@Output** decorator is used to binds a property of a component to send the data from child component to parent component and this is a one-way communication.

**@Output** decorates output properties and its binds a property of the type of angular EventEmitter.

If you want to bind an event on an element, you can use the new Angular2 events i.e.

@Component(...)

**class** yourComponent {

addUser(event) {

}

}

The method **addUser()** will be called when user clicked on button.

<button (click)="addUser()">Click</button>

<button (click)="addUser($event)"></button>

***What happen if you want to create a custom event?***

Now come to the outputs, if you want to create your custom event in Angular 2 that time we will use to new [**@Output**](https://www.code-sample.com/2016/06/angular-2-outputs.html)**decorator**.

**--------------------------------------------------------------------------------------------------------------------------------------------**

**Examples,**

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

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

@Component({

selector: 'my-app',

providers: [Service],

template: '<div>Hello my name is {{name}}!</div>'

})

**class** MyApp {

**constructor**(service: **Service**) {

**this**.name = service.getName();

setTimeout(() => **this**.name = 'Anil Singh,', **1000**);

}

}

**class** Service {

getName() {

**return** 'Hello';

}

}

bootstrap(App);

**--------------------------------------------------------------------------------------------------------------------------------------------**

**In the above example, we will need to import Output and Event-Emitter to create our new custom event.**

**--------------------------------------------------------------------------------------------------------------------------------------------**

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

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

@Component({

selector: 'my-app',

providers: [Service],

template: '<div>Hello my name is {{name}}!</div>'

})

**class** MyApp {

**constructor**(service: **Service**) {

**this**.userClicked.emit(**this**.user);

**this**.name = service.getName();

setTimeout(() => **this**.name = 'Anil Singh,', **1000**);

}

}

**class** Service {

getName() {

**return** 'Hello';

}

@Output() userClicked = **new** EventEmitter();

}

bootstrap(App);

**--------------------------------------------------------------------------------------------------------------------------------------------**

# How do Components Communicate with Each Other in Angular 2?

**Ans:**

**In Angular 1**, I have some ways to communicate between controllers i.e.

1.      $rootScope,

2.      $scope,

3.      $emit,

4.      $broadcast

Now**In Angular 2**, we have different ways to communicate between components.

A parent component and its children share a service whose interface enables bi-directional communication within the family.

**--------------------------------------------------------------------------------------------------------------------------------------------**

**The following examples for Services communication,**

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

@Injectable()

**export** **class** MyService {

**constructor**() { }

}

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

@Component({

selector: 'my-app',

templateUrl: './myApp.component.html'

})

**export** **class** MyAppComponent { }

**The following example to calling service from any other component,**

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

**import** {MyService} from './app/myService';

@Component({

selector: '<my-component></my-component>',

templateUrl: 'app/component.html',

providers: [MyService]

})

**export** **class** MyComponent **implements** OnInit {

**constructor**(**private** msInstance: **MyService**) {}

ngOnInit() {

**this**.msInstance.getServices();

}

}

**Example for Sibling Component Communication,**

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

**import** { ListComponent } from './list.component';

**import** { DetailComponent } from './detail.component';

@Component({

selector: 'app-component',

template: '<list-component></list-component><detail-component></detail-component>',

directives: [ListComponent, DetailComponent]

})

**class** AppComponent **implements** AfterViewInit {

@ViewChild(ListComponent) listComponent:**ListComponent**;

@ViewChild(DetailComponent) detailComponent: **DetailComponent**;

ngAfterViewInit() {

**// afther this point the children are set, so you can use them**

**this**.detailComponent.doSomething();

}

}

**--------------------------------------------------------------------------------------------------------------------------------------------**

# How do we display errors in a component view with Angular 2?

**Ans:**

**In Angular 1**, the ng-messages modules are used to help us to display error messages and validation to our forms.

**In Angular 2**, the ngModel provides error objects for each of the built-in input validators. You can access these errors from a reference to the ngModel itself then build useful messaging around them to display to your users.

**And also**, we can use the properties “**pristine**” and “**touched**” to display error messages.

1.      If we want to display errors after the user fills something in a field, use the pristine property.

2.      If we want to display errors after the user put the focus on a field, use the touched property.

**Example as,**

<div \*ngIf="(!loginForm.controls.email.valid && !loginForm.controls.email.pristine)">

\*\*Email is required.

</div>

# ElementRef vs Renderer - Angular 2

# Ans: ElementRef vs. Renderer -

In Angular Renderer and ElementRefare used for DOM Manipulation and Renderer and ElementRef are used together to get full platform abstraction.

**Renderer** –

Renderer is a class that is a partial abstraction done the DOM manipulations and the DOM manipulating is not breaking server side rendering or web workers.

**ElementRef –**

ElementRef is a class that is a partial abstraction done the DOM Manipulations without breakable environments and it also can hold a reference to a DOM elements.

If “ElementRef” is injected to a component, the injected instance is a reference to the host element of the current component.

The ways to get an ElementRef instance looks like,

1.     @ViewChild()

2.     @ViewChildren()

3.     @ContentChild()

4.     @ContentChildren()

In this case the ElementRef is a reference to the matching elements in the templates.

**Do notice that you should refrain from using ElementHref as it flagged with a security risk?**

If you allow direct access to the DOM, it can make your application more vulnerable to XSS attacks. So make sure, when you are using to ElementRef in your app code.

**What is the point of calling renderer.invokeElementMethod(rendererEl, methodName)?**

**===============================================**

//our root app component

**import** {Component, ElementRef} from 'angular2/core'

**import** \* **as** browser from 'angular2/platform/browser'

**import** {Ruler, Rectangle} from 'angular2/src/platform/browser/ruler.js'

@Component({

selector: 'my-app',

providers: [ElementRef],

template: `

<div>

<h2>Hello {{name}}</h2>

<p>H2 Height: {{rect.height}}</p>

<p>H2 Width: {{rect.width}}</p>

</div>

`,

directives: []

})

**export** **class** App {

**constructor**(element: **ElementRef**) {

**this**.name = 'Angular2'

**this**.element = element;

**this**.ruler = **new** Ruler(**new** browser.BrowserDomAdapter());

**this**.rect = {};

}

ngOnInit() {

**var** vm = **this**;

**var** measure = **this**.ruler.measure(**this**.element);

measure.then(**function** (rect) {

console.log('Rect', rect);

vm.rect = rect;

});

}

}

**===============================================**

# Services Related Question:

1. **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,

 Services are singleton objects.

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

Service class is decorated with Injectable decorator.

 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.

**3. 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.

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 :-**

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

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

import { UniversalModule } from 'angular2-universal';

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

import { HttpModule } from '@angular/http';

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

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

import { HeaderComponent } from './components/shared/header/header.component';

import { MenuComponent } from './components/menu/menu.component';

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

import { RegistrationComponent } from './components/registration/registration.component';

@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 :-**

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

import { CommonModule } from '@angular/common';

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

//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>          </tr>

                    </tbody>

                </table>

            </div>

        </div>

    </div>

</div>

</div>

**===============================================**

1. **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.

**Q. 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.

**Q. Angular 2 Pipes in Depth [Custom Pipes and Inbuilt Pipes with Examples]**

**Ans:**

**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,**

**Q. 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.

**Q. 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 // Annotation ({

    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.

**Table of Component**

1.      user.component.ts

2.      user.service.ts

3.      custom.barcode.pipe.ts

4.      app.module.ts

5.      user.component.html

**===============================================**

**user.component.ts :-**

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

import { CommonModule } from '@angular/common';

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

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 :-**

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

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

import 'rxjs/add/operator/map';

//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 :-**

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

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

import { UniversalModule } from 'angular2-universal';

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

import { HttpModule } from '@angular/http';

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

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

import { HeaderComponent } from './components/shared/header/header.component';

import { MenuComponent } from './components/menu/menu.component';

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

import { RegistrationComponent } from './components/registration/registration.component';

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

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

import { MyPipePipe } from './components/shared/pipe/test.pipes';

@NgModule({

    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></table></div></div></div></div></div>

**===============================================**

**Directives Related Questions:**

# Q. Angular 2 Directives [Components, Structural, Attribute 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.

**Q. 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.

**Q. 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.

**Q. 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>

**===============================================**

# Q. Angular 2 Router Outlet Directives [Angular 2 Route Params and Config]

**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>

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

**Syntax:-**

<router-link> </router-link>

**Example,**

<a [router-link]="['/AboutMe']">About Me</a>

**The Route-Config: -** The route config is used to map components to URLs.

**Syntax:-**

@RouteConfig([

{path: '/', component: **Home\_Component**, **as**: 'Home'},

{path: '/AboutMe', component: **AboutMe\_Component**, **as**: 'AboutMe' }

{path: '/ContactMe', component: **ContactMe\_Component**, **as**: 'ContactMe' }

])

**The Route Params: -**The routeparameter 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,**

@RouteConfig([

{path: '/employ/:id', component: **employe**, name: 'emp'},

])

**===============================================**

# Q. Angular 2 Structural Directives [How To Write Structural Directives?]

**Ans:**

**What are directives?**

There are two other kinds of Angular directives,

1.      Components

2.      Attribute directives

**What are structural directives?**

The “Structural directives” are responsible for HTML layout. They shape or reshape the DOM structure; it is using for adding, removing and manipulating the elements.

The “Structural directives” is used to enable an element as a template for creating additional elements. If you want to create structural directive that time you should have knowledge of**<template>**elements and structural directives are easy to recognize.

The two familiar examples of structural directive as,

1.      \*ngIf

2.      \*ngfor

An asterisk (**\***) precedes the directive attribute name as

**<div \*ngIf=”user”>{{user.name}}</div>**

**How to creating a structural directive?**

**===========================================================================**

@Directive({

selector: '[appDelay]'

})

**export** **class** DelayDirective {

**constructor**(

**private** templateRef: **TemplateRef**<any>,

**private** viewContainerRef: **ViewContainerRef**

) { }

@Input()

set appDelay(time: **number**): **void** { }

}

**===========================================================================**

**How to create multiple structural directives?**

**===========================================================================**

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

@Component({

selector: 'app-root',

styles: [`

.tabs-sec {

background-color: #cccff;

display: **flex**;

flex-direction: **row**;

width: **100**%;

}

`],

template: `

<div **class**="tabs-sec">

<app-tab

\*ngFor="let tab of tabs; let i = index"

[active]="isSelected(i)"

(click)="setTab(i)">

{{ tab.title }}

</app-tab>

</div>

<div [ngSwitch]="tabNumber">

<template ngFor [ngForOf]="tabs" **let**-tab **let**-i="index">

<app-tab-content \*ngSwitchCase="i">

{{tab.content}}

</app-tab-content>

</template>

<app-tab-content \*ngSwitchDefault>clect to select your tab</app-tab-content>

</div>

`

})

**export** **class** AppComponent {

tabNumber = -**1**;

tabs = [

{ title: 'Blogger1', content: 'Tab Blogger 1' },

{ title: 'Blogger2', content: 'Tab Blogger 2' },

{ title: 'Blogger3', content: 'Tab Blogger 3' },

];

setTab(num: **number**) {

**this**.tabNumber = num;

}

isSelected(num: **number**) {

**return** **this**.tabNumber === i;

}

}

**============================================================================**

**Reference for directives creation.**

<https://egghead.io/lessons/angular-2-write-a-structural-directive-in-angular-2>

**============================================================================**

**Template Related questions**

**Q. Angular 2 Templates - template vs. templateUrl? How to Use Inline and External Templates?**

# Ans:

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;

}

}

# ===============================================================

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 –**

**===========================================================================**

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

**import** { CommonModule } from '@angular/common';

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

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

**import** { AppGlobals } from '../../shared/app.globals';

@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-**

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

**import** { CommonModule } from '@angular/common';

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

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

**import** { AppGlobals } from '../../shared/app.globals';

@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);

}

}

**=============================================================================**

**Q. 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 –**

**==================================================================================**

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

**import** { CommonModule } from '@angular/common';

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

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

**import** { AppGlobals } from '../../shared/app.globals';

@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-**

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

**import** { CommonModule } from '@angular/common';

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

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

**import** { AppGlobals } from '../../shared/app.globals';

@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 Related Questions:**

**-------------------------------------**

# Q. Angular 2 Routing Concepts and Examples

Ans: “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**,

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

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

import { UniversalModule } from 'angular2-universal';

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

import { HttpModule } from '@angular/http';

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

import { HomeComponent } from './components/home/home.component';

import { HeaderComponent } from './components/shared/header/header.component';

import { MenuComponent } from './components/menu/menu.component';

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

import { RegistrationComponent } from './components/registration/registration.component'

@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>

=============================================================  
**References-**

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

[http://www.code-sample.com/2015/07/angularjs-2-documentation-with-example.html](https://www.code-sample.com/2015/07/angularjs-2-documentation-with-example.html)

# Q. Angular 2 - What is Routes?

**Ans:** 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' }

        ])

# Q. What is router-outlet directive in Angular 2?

**Ans:** The Router-Link, RouterLink-Active and Router-Outlet are directives provided by the Angular RouterModule package. It is Provides the navigation and URLs manipulation capabilities.

**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 as,**

<div **class**='container'>

<div **class**='row'>

<router-outlet></router-outlet>

</div>

</div>

=================================================================================

# Q. Is it possible to have a multiple router-outlet in the same template?

**Ans:**

Yes! We can use multiple router-outlets in same template by configuring our routers and simply add the router-outlet name. You can see in the example.

**Syntax-**

<div class="row"><div class="user"> <router-outlet name="userList"></router-outlet>

</div> <div class="userInfo"> <router-outlet name="userInfo"></router-outlet>

</div></div>

And setup your route config:

**const** routes: **Routes** = [

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

{ path: 'home', component: **HomeComponent** },

{ path: 'user', component: **userComponent**, children: [

{ path: 'userList', component: **userListComponent**, outlet: 'userList' },

{ path: ':id', component: **userInfoComponent**, outlet: 'userInfo' }]

}];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule],

providers: []

})

**export** **class** RoutingModule { }

# ==================================================================

# State management Questions:

# =========================

# Q. Angular2 cookies | angular2 http cookies | angular2 http get set cookie

# Ans:

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

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

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

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

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

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

7.     **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.

**=======================================================================**

**Dependency Injection Questions:**

**Q. Dependency Injection (DI) in Angular 2 [Why @Injectable()?]**

**Ans:** Dependency Injection is a powerful pattern for managing code dependencies.

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; };

**===================================================================**

**Q. Angular 2 Injectable - Why @Injectable ()?**

**Ans:**

**Why @Injectable ()?**

@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.

**Example as,**

**import** { Injectable, InjectionToken } from '@angular/core';

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

@Injectable()

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

**constructor**(**private** \_http: **Http**) {

}

getAPIUsers(apiUrl) {

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

}

}

**====================================================================Q. Angular 2 Inject - Why @Inject()?**

**Ans:** Angular 2 @Inject() is a special technique for letting Angular know that a parameter must be injected.

**Example as,**

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

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

**class** UserService {

users: **Array**<any>;

**constructor**( @Inject(Http) \_http: **Http**) {

}

=======================================================================

**Q. Angular 2 @Injectable() vs. @Inject() ?**

**Ans:**

**Why @Injectable ()?**

@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.

**Example as,**

**import** { Injectable, InjectionToken } from '@angular/core';

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

@Injectable()

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

**constructor**(**private** \_http: **Http**) {

}

getAPIUsers(apiUrl) {

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

}

}

**Why @Inject()?**

Angular 2 @Inject() is a special technique for letting Angular know that a parameter must be injected.

**Example as,**

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

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

**class** UserService {

users: **Array**<any>;

**constructor**( @Inject(Http) \_http: **Http**) {

}

}

**===============================================================**

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

**Ans:**

The basics Steps of Dependency injection,

1.     A class with @Injectable () to tell Angular2 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, InjectionToken } from '@angular/core';

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

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

//BEGIN-REGION - USERSERVICE

@Injectable()

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

**constructor**(**private** \_http: **Http**) {

}

getAPIUsers(apiUrl) {

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

}

getAppUsers(apiUrl) {

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

}

}

//END BEGIN - USERSERVICE

====================================================================

**Ng-Modules Questions:**

**=======================**

**Q. Angular 2 @NgModel [Purpose of Root Module & Export Module]**

**Ans:**

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-faceted**sample with **lazy** modules.

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 { }

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

**Q. Angular 2 Modules vs. JavaScript Modules vs. Angular 1 Modules**

**Ans:**

**Angular 2 Modules -**

The Angular module — a class decorated with @NgModule — is a fundamental feature of Angular.

JavaScript also has its own module system for managing collections of JavaScript objects. It's completely different and unrelated to the Angular module system.

Angular Modules are the unit of reusability.

Angular modules represent a core concept and play a fundamental role in structuring Angular applications.

Every Angular app has at least one module, the root module, conventionally named AppModule.

Important features such as lazy loading are done at the Angular Module level.

Angular Modules logically group different Angular artifacts such as components, pipes, directives, and so on.

Angular Modules help to organize an application into cohesive blocks of functionalities and extend it with capabilities from external libraries.

**App module looks like below,**

---------------------------------------------------

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

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

**import** { HttpModule } from '@angular/http';

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

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

**import** { NavMenuComponent } from './components/navmenu/navmenu.component';

**import** { HomeComponent } from './components/home/home.component';

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

**import** { UserService } from './components/service/user.service';

**import** { BarCodePipe } from './components/pipe/custom.pipe';

**export** **const** sharedConfig: **NgModule** = {

bootstrap: [ AppComponent ],

declarations: [

AppComponent,

NavMenuComponent,

HomeComponent,

UserComponent,

BarCodePipe

],

imports: [

RouterModule.forRoot([

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

{ path: 'home', component: **HomeComponent** },

{ path: 'user', component: **UserComponent** },

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

])

],

providers: [UserService]

};

**Angular 1 Module -**

1.     Services

2.     Directives

3.     Controllers

4.     Filters

5.     Configuration information

6.     And so on…

**JavaScript Modules -**

In JavaScript Modules, every file is one module. In Angular 2, one component is normally a file.

JavaScript also has its own module system for managing collections of JavaScript objects. It's completely different and unrelated to the Angular module system.

Avoid leaking code to the global namespace and thus to avoid naming collisions.

Encapsulate code to hide implementation details and control what gets exposed to the outside.

Structure our applications and we can’t use a single file.

Manage dependencies and code reuse.

==========================================================================

# Q. What are Zones? What is Change Detection? What is NgZone run outside ?

# Ans:

**What are Zones? What is NgZone in Angular 2?**

Angular 2 runs inside of its own special zone called NgZone and this special zone extends the basic functionality of a zone to facilitate change detection.

It is Running inside a zone allows to detect when asynchronous tasks.

If you change or update your internal application code or view and it is detecting the applications changes with help of NgZone.

Application state change by following things as,

1.     **Events** – Looks like as click, change, input, submit, etc.

2.     **XMLHttpRequests** – It’s occurs when we fetch data from a remote service.

3.     **Timers** – when you use timer methods such as setTimeout (), setInterval (), etc.

**Q. What is Change Detection?**

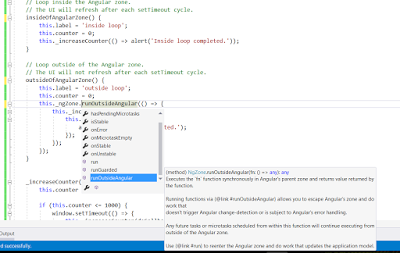
Angular 2 runs inside of its own special zone called NgZone and this special zone extends the basic functionality of a zone to facilitate change detection.

A zone is not a concept that is specific to Angular 2 and these Zones features or functionality can be added to any JavaScript application with the inclusion of the “Zone.js” library.

**Q. What is NgZone run outside Angular 2?**

Execute the “fn” functions asynchronously in Angular’s parent zone and returns value returned by the function.

**Syntax**- NgZone.runOutsideAngular(fn:() => any): any

[](https://4.bp.blogspot.com/-MvvVnCvwp0s/WU4PDA312_I/AAAAAAAAQD4/LLFWz61ZhrMniCA9SBwFZ_8j6-Wb4f77QCLcBGAs/s1600/ngzone%2Bangular2%2Bexample%2B%2B-%2Brun%2Boutside%2Bangular.png)

**Example using NgZone  as,**

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

**import** { CommonModule } from '@angular/common';

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

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

**import** { AppGlobals } from '../../shared/app.globals';

@Component({

selector: 'user',

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

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

providers: [UserService, AppGlobals]

})

**export** **class** UserComponent { //USERS DECLARATIONS.

users = [];

label: **string**;

counter: **number** = **0**; //USER COMPONENT CONSTRUCTOR.

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

**private** \_global: **AppGlobals**,

**private** \_ngZone: **NgZone**)

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

ngOnInit() {

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

**this**.\_userService.getAppUsers(**this**.\_global.baseAppUrl + 'api/User/GetUsers').subscribe(data => console.log(data));

} // Loop inside the Angular zone.

// The UI will refresh after each setTimeout cycle.

insideOfAngularZone() {

**this**.label = 'inside loop';

**this**.counter = **0**;

**this**.\_increaseCounter(() => alert('Inside loop completed.'));

} // Loop outside of the Angular zone.

// The UI will not refresh after each setTimeout cycle.

outsideOfAngularZone() {

**this**.label = 'outside loop';

**this**.counter = **0**;

**this**.\_ngZone.runOutsideAngular(() => {

**this**.\_increaseCounter(() => {

**this**.\_ngZone.run(() => {

alert('Outside loop completed.');

});

});

}); } \_increaseCounter(doCallback: () => **void**) {

**this**.counter += **2**;

**if** (**this**.counter <= **1000**) {

window.setTimeout(() => {

**this**.\_increaseCounter(doCallback);

}, **100**);

} **else** {

doCallback();

} }

}//END BEGIN - USER-COMPONENT

=================================================================

# Q. What Are Isolated Unit Tests? [Angular 4 and Angular 2]

**Ans: What Are Isolated Unit Tests?**

The Isolated unit tests check-up an instance of a class itself without using any Angular dependence or any injected values.

Mostly application tester creates a test instance of the class with new keyword and supplying test doubles for the constructor parameters and then investigation the test instance.

The isolated unit tests don't realize how components interact with Angular and also don't realize how a component class interacts with its own template or components.

For testing Angular Pipes and Services - we should write isolated unit tests!

The isolated unit tests don't realize how components interact with Angular and also don't realize how a component class interacts with its own template or components.  
Stayed Informed - [Angular 2 Docs with Examples](https://www.code-sample.com/2015/07/angularjs-2-documentation-with-example.html)  
Stayed Informed - [Angular 4 Docs with Examples](https://www.code-sample.com/2017/04/angular-4-interview-questions-and.html)

**The most familiar Unit Test for the Tester and Developers as following -**

1.     Create an Instances directly with new keyword

2.     Angular Agnostic Testing Techniques

3.     Exhibit Standard

4.     Substitute Test

**The Most of the Tester and Developers are tried to avoid Unit Testing following methodology-**

1.     Import from the Angular Test Libraries - @angular/core/testing

2.     Configure Angular module

3.     Prepare Dependency Injection Providers

4.     Call Inject or (async/fakeAsync)

**Example as –**This example is used to display Credit Card Number with a custom formatted in the user templates.

**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**): '')

}

}

**Unit Testing to the Pipe -** **BarCodePipe**

describe('BarCodePipe', () => {

**let** pipe = **new** BarCodePipe();

//Todo tests ...

});

-------------------------------------------------------------------------------------------------------------------------------

**References** -

<https://angular.io/guide/testing>

<https://angular.io/guide/testing#isolated-unit-tests>

**==============================================================**

# Q. What Are Angular Testing Utilities? [Angular 4 and Angular 2]

**Ans:** The Angular Testing utilities include the **TestBed** class and helper functions from the test libraries - @angular/core/testing.

The **TestBed** class is one of the principal Angular testing utilities!

The **TestBed** class is responsible for configuring and initializing the environment that we are going to write our tests in by calling **TestBed.configureTestingModule.**  
The **TestBed.configureTestingModule** is used to define the environment that we want our component under test to live in.

**The Angular Testing utility APIs are –**

1.     getTestBed

2.     async

3.     fakeAsync

4.     tick

5.     inject

6.     discardPeriodicTasks

7.     flushMicrotasks

8.     ComponentFixtureAutoDetect

**The most important static methods are –**

1.     configureTestingModule

2.     compileComponents

3.     createComponent

4.     overrideModule

5.     overrideComponent

6.     overrideDirective

7.     overridePipe

8.     get

9.     initTestEnvironment

10.  resetTestEnvironment

-------------------------------------------------------------------------

**Example As –**

beforeEach(() => {

fixture = TestBed.configureTestingModule({

declarations: [YourComponent ]

})

.createComponent(YourComponent);

})

---------------------------------------------------------------