Angular Interview Questions & Answers

Basic Questions

1.What is Angular 2?

Angular is a most popular web development framework for developing mobile apps as well as desktop applications.

Angular framework is also utilized in the cross platform mobile development called IONIC and so it is not limited to web apps only.

Angular is an open source framework written and maintained by angular team at Google and the Father of Angular is Misko Hevery.

By Angular Developer Guide book - "Angular is a platform and framework for building client applications in HTML and TypeScript. Angular is itself written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your apps."

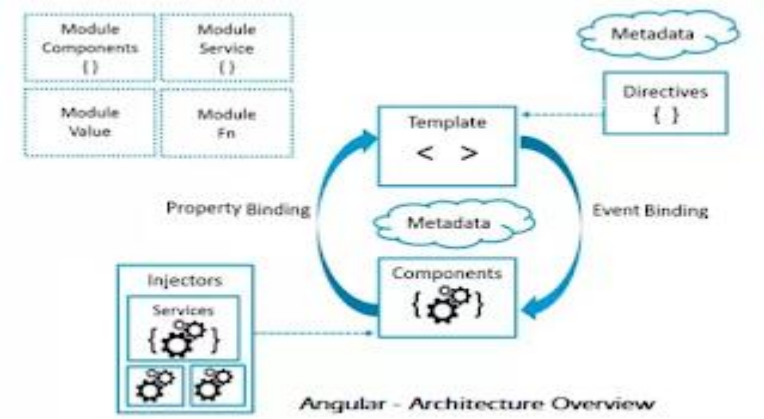
The "Angular 2" is focusing on data-binding, extensible HTML and on application test-ability but it is still in design and prototyping stage.

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 (ES6 is called ECMAScript version 6). It also support to ECMAScript version 5(ES5).

You don't worry about the versions of ECMAScript. The ES6 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, 11 and so on...

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.

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

The “Angular 2” is using “Traceur” compiler. Traceur is a compiler that takes “ES6” and compiles it down (ES5) to regular JavaScript that runs in your browsers. It is run everywhere you want to do.

2.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.

4.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
2. Component
3. Template
4. Outputs
5. Data Binding
6. Directive
7. Service
8. Dependency Injection

5.What Are The New Features Of Angular 2? Why You Used Angular 2?

Angular 2 Features –

- 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.....

Component Based- It is entirely component based. It is not used to scope and controllers and Angular 2 are fully replaced by components and directives.

Directives- The directive can be declared as @Directive annotation.
A component is a directive with a template and the @Component decorator is actually a @Directive decorator extended with template oriented features.

Dependency Injection- Dependency Injection is a powerful pattern for managing code dependencies. There are more opportunities for component and object based to improve the dependency injection.

Use of TypeScript- Type represents the different types of values which are using in the programming languages and it checks the validity of the supplied values before they are manipulated by your programs.

Generics- TypeScript has generics which can be used in the front-end development.

Lambdas and Arrow functions – In the TypeScript, lambdas/ arrow functions are available. The arrow function is additional feature in typescript and it is also known as a lambda function.

Forms and Validations- Angular 2 forms and validations are an important aspect of front-end development.

6.Why You Used Angular 2?

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

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

7.Why should you use Angular 2 ? What are the Advantages of Angular 2 ?

The core differences and many more advantages on Angular 2 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.

For Examples as,

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,

```
agular.element(document).ready(function() {
   angular.bootstrap(document, ['userApp']);
});n
```

Bootstrapping in Angular 2,

```
import { bootstrap } from 'angular2/platform/browser';
import { UsersComponent } from './product.component';
bootstrap(UserComponent)
```

For example as,

```
//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>
```

TypeScript Interview Questions and Answers - JavaScript!

8.What are TypeScript Types?

The Type represents the different types of values which are using in the programming languages and it checks the validity of the supplied values before they are manipulated by your programs.

The TypeScript provides data types as a part of its optional type and its provide us some primitive types as well as a dynamic type “any” and this “any” work like “dynamic”.

In TypeScript, we define a variable with a “type” and appending the “variable name” with “colon” followed by the type name i.e.

```
let isActive: boolean = false; OR var isActive: boolean = false;
let decimal: number = 6; OR var decimal: number = 6;
let hex: number = 0xf00d; OR var hex: number = 0xf00d;
let name: string = "Anil Singh"; OR var name: string = "Anil Singh";
let binary: number = 0b1010; OR var binary: number = 0b1010;
let octal: number = 0o744; OR var octal: number = 0o744;
let numlist: number[] = [1, 2, 3]; OR var numlist: number[] = [1, 2, 3];
let arrlist: Array<number> = [1, 2, 3]; OR var arrlist: Array<number> = [1, 2, 3];

//Any Keyword
let list: any[] = [1, true, "free"];
list[1] = 100;

//Any Keyword
let notSureType: any = 10;
notSureType = "maybe a string instead";

notSureType = false; // definitely a Boolean
```

Number: the “number” is a primitive number type in TypeScript. There is no different type for float or double in TypeScript.
Boolean: The “boolean” type represents true or false condition.
String: The “string” represents sequence of characters similar to C#.
Null: The “null” is a special type which assigns null value to a variable.
Undefined: The “undefined” is also a special type and can be assigned to any variable.
Any : this data type is the super type of all types in TypeScript. It is also known as dynamic type and using “any” type is equivalent to opting out of type checking for a variable.

A note about “let” keyword –

You may have noticed that, I am using the “let” keyword instead of “var” keyword. The “let” keyword is actually a newer JavaScript construct that TypeScript makes available. Actually, many common problems in JavaScript are reducing by using “let” keyword. So we should use “let” keyword instead of “var” keyword.

Setup and Install Typescript NPM and Angular 2

Two main ways to Installing the TypeScript,

1. Installing using npm
2. Installing TypeScript's Visual Studio plugins

In the Visual Studio 2017, TypeScript include by default. If you don't have TypeScript with Visual Studio, try for NPM users,

```
> npm install -g typescript
```

Example

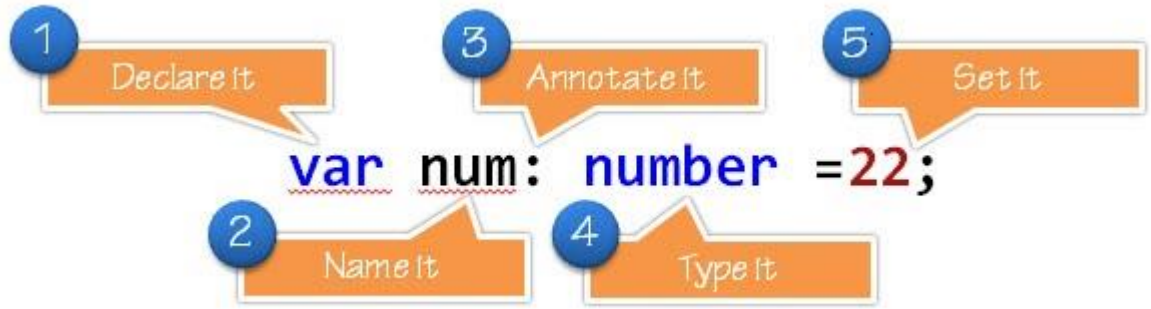
```
class Users {
   private firstName: string;
   private lastName: string;

   //Constructor
   constructor(firstName: string, lastName: string) {
      this.firstName = firstName;
      this.lastName = lastName;
   }

   //Function
   studentFullName(): void {
      alert(this.firstName + ' ' + this.lastName);
   }
}
```

```
private firstName: string;
   private lastName: string;

   //Constructor
   constructor(firstName: string, lastName: string) {
      this.firstName = firstName;
      this.lastName = lastName;
   }

   //Function
   studentFullName(): void {
      alert(this.firstName + ' ' + this.lastName);
   }
}
```

9.What is Variable in TypeScript? How to Declare Variable in TypeScript?

The variable is simply a name of storage location and all variables must be identified with unique names and these unique names are called identifiers.
A variable contains values such as "Hi" or 22. When you use the variable, you refer to the data it represents. For example –

let name: string = "Hi"; //var name: string = "Anil Singh";
let num: number = 22; //var num: number = 22;



There are some rules while declaring variables i.e.

1. The variable names must begin with a letter
2. The variable names can contain letters, digits, underscores, and dollar signs.
3. The variable names can also begin with "$" and "_"
4. The variable names are case sensitive that means "a" and "A" are different variables.
5. The variable reserved words can't be used as names.

Declaring Variables

```
let isActive: boolean = false;
let decimal: number = 6;
let hex: number = 0xf00d;
let name: string = "Anil Singh";
let binary: number = 0b1010;
let octal: number = 0o744;
let numlist: number[] = [1, 2, 3];
let arrlist: Array<number> = [1, 2, 3];

//Any Keyword
let list: any[] = [1, true, "free"];
list[1] = 100;

//Any Keyword
```

```
let notSureType: any = 10;
notSureType = "maybe a string instead";
notSureType = false; // definitely a Boolean
```

TypeScript – [declare vs. var]

The “var” creates a new variable. The “declare” is used to tell TypeScript that the variable has been created elsewhere. If you use declare, nothing is added to the JavaScript that is generated. It is simply a hint to the compiler.

10.What is scope variable?

The scope is set of objects, variables and function and the JavaScript can have global scope variable and local scope variable.

We can say, we can declare a variable in the two type’s i.e.

1. Local Scope Variable
2. Global Scope Variable

Global Scope Variable - The global scope is a window object and it’s used out of function and within the functions.

Local Scope Variable - The local scope is a function object and it’s used within the functions.

Example for global and local variable (var/let) declarations –

```
//DECLARED A VARIABLE NAMED NUM WITH THE VALUE 15.
let num = 15;
```

```
//DECLARE A VARIABLE INSIDE OF A FUNCTION
let getUser = function () {
   let name = 'Anil Singh';

   return name;
}

//RESULTS
getUser(); // returns 'Anil Singh'
```

```
//ACCESS THOSE SAME VARIABLES WITHIN OTHER FUNCTIONS
let sum = function() {
   let num = 15;

   return function subNum () {
      return num + 5;
   }
}

//RESULTS
sum(); // returns 20
```

```
scopingRulesSum(true);  // returns 16
scopingRulesSum(false); // returns 'undefined'
```

```
//IN THE JAVASCRIPT, WE WILL DECLARE MULTIPLE VARIABLES WITH THE SAME NAME.
//IN JAVASCRIPT, NO MATTER HOW MANY TIMES YOU DECLARED YOUR VARIABLES.
//WE CAN SAY THIS IS A BUG.
var fun = function (i, isActive = true) {
   var i;
   var i;

   if (isActive) {
      var i;
   }
   //The all declarations of 'i' actually refer to the same 'i'.
}
```

```
//IN THE TYPESCRIPT, NOT POBISBLE TO DECLARE MULTIPLE VARIABLES WITH THE SAME
NAME
//IT WILL GIVE US ERROR (ERROR: CAN'T RE-DECLARE 'I' IN THE SAME SCOPE).
let i = 15;
let i = 20; // error: can't re-declare variable 'i' in the same scope.
```

11.What is Optional Properties in TypeScript?

We can specify optional properties on interfaces and the property may be present or missing in the object.

In the below example, the address property is optional on the following “User” interface.

Example as,

```
//FUNCTION USING USER INTERFACE
let userInfo = function(user: User) {
  let info = "Hello, " + user.name + " Your Age is - " + user.age + " and Address is -" +
user.address;

  return info;
}

//USER INFO JSON OBJECT
let info = {
   name: "Anil",
   age: 30
};

//RESULT
console.log(userInfo(info));
```

TypeScript - Default Parameters Function

Default Parameters -

Function parameters can also be assigned values by default.
A parameter can't be declared as optional and default both at the same time.

For Example as,

```
let discount = function (price: number, rate: number = 0.40) {
   return price * rate;
}

//CALCULATE DISCOUNT
discount(500); // Result - 200
```

```
//CALCULATE DISCOUNT
discount(500, 0.45); // Result - 225
```

TypeScript - Named Function

Named Function -

The named function is very similar to the JavaScript function and only one difference - we must declare the type on the passed parameters.

Example – JavaScript

```
function addTwoNumer(num1, num2) {
   return num1 + num2;
}
```

Example – TypeScript

```
function addTwoNumer(num1: number, num2: number): number {
   return num1 + num2;
}
```

TypeScript - Lambda / Arrow Functions

Lambda Function/Arrow Function -

The arrow function is additional feature in typescript and it is also known as a lambda function.

A lambda function is a function without a name.

```
var addNum = (n1: number, n2: number) => n1 + n2;
```

In the above, the "=>" is a lambda operator and (n1 + n2) is the body of the function and (n1: number, n2: number) are inline parameters.

For example –

```
let addNum = (n1: number, n2: number): number => { return n1 + n2; }
let multiNum = (n1: number, n2: number): number => { return n1 * n2; }
```

```
let dividNum = (n1: number, n2: number): number => { return n1 / n2; }

addNum(10, 2);// Result - 12
multiNum(10, 2);// Result - 20
multiNum(10, 2);// Result - 5
```

TypeScript - Rest Parameters Functions

Rest Parameters –

The Rest parameters do not restrict the number of values that we can pass to a function and the passed values must be the same type otherwise throw the error.

For Example as,

```
addNumbers(1, 2);
addNumbers(1, 2, 3);
addNumbers(1, 12, 10, 18, 17);
```

TypeScript Public, Private, Protected and Readonly Modifiers!

TypeScript Modifiers

The TypeScript supports to multiple modifiers and it is by default public.

1. Public,
2. Private,
3. Protected and
4. Read-only

Public Modifier- Public by default! It is freely access anywhere.

In the below example, the class Employee and its members are by default public and we are freely access it.

Example –

```
class Employee {
   empName: string;
   constructor(name: string) {
      this.empName = name;
```

```
    }

    salary(salary: number = 10000) {
        console.log('Hello, ' + this.empName + ' Your Salary -' + salary);
    }
}

let empSal = new Employee("Anil");
console.log(empSal.salary());
console.log(empSal.salary(40000));
```

Private Modifier- When using private modifier, we can't be accessed from outside of its containing class.

Example as,

```
class Employee {
    private empName: string;
    constructor(name: string) {
        this.empName = name;
    }

    salary(salary: number = 10000) {
        console.log('Hello, ' + this.empName + ' Your Salary -' + salary);
    }
}

let emp = new Employee("Anil").empName;
//error: property 'empName' is private and only accesible in the class 'Employee'.
```

Protected Modifier - The protected modifier is very similar to private but only one difference that can be accessed by instances of deriving classes.

Example as,

```
class Employee {
    protected  empName: string;
    constructor(name: string) {
        this.empName = name;
    }
```

```
    salary(salary: number = 10000) {
        console.log('Hello, ' + this.empName + ' Your Salary -' + salary);
    }
}

class Employer extends Employee {
    private department: string;

    constructor(empName: string, department: string) {
        super(empName);
        this.department = department;
    }

    salary(salary = 20000) {
        super.salary(salary);
    }
}

let empSal = new Employer("Anil", "IT");
console.log(empSal.salary());
console.log(empSal.empName); //error- the property 'empName' is protected and only accesible within the class 'Employee' and its child class.
```

Readonly Modifier - Read-only properties must be initialized at their declaration or in the constructor.

For example as,

```
class Employee {
    readonly empName: string;
    constructor(name: string) {
        this.empName = name;
    }

    salary(salary: number = 10000) {
        console.log('Hello, ' + this.empName + ' Your Salary -' + salary);
    }
}

let emp = new Employee('Anil');
emp.empName = 'Anil Singh';//error - cannot assign to 'empName' because it is constant or readonly.
```

TypeScript - Inheritance and Examples

Inheritance - TypeScript is supports the concept of Inheritance.

Inheritance has ability of a program to extend existing classes to create new ones.

The extended class is called parent class or super class and the newly created classes are called child class or sub class.

Inheritance can be classified as -

1. **Single** - every class can at the most extend from one parent class
2. **Multiple** - doesn't support multiple inheritances in TypeScript.
3. **Multi-level**

A class captains the Public, Private, Protected and Read-only modifiers and Public by default. You can see the below example, the class User and each members are public by default.

Example as,

```
class Employee {
   empName: string;
   constructor(name: string) {
      this.empName = name;
   }

   salary(salary: number = 10000) {
      console.log('Hello, ' + this.empName + ' Your Salary -' + salary);
   }
}

class Employer extends Employee {
   constructor(empName: string) {
      super(empName);
   }

   salary(salary = 20000) {
      super.salary(salary);
   }
}
```

```
let empSal = new Employee("Anil");
console.log(empSal.salary());
console.log(empSal.salary(40000));
```

How to create fields, constructor and function in TypeScript Class

16.What is class in TypeScript?

A class is a template definition of the methods and variables in a particular kind of object. It is extensible program, code and template for creating objects.

A TypeScript is object oriented JavaScript and it also supports object oriented programming features like classes, interfaces, etc.

A class captains the Public, Private, Protected and Read-only modifiers and Public by default. You can see the below example, the class User and each members are public by default.

A class definition can contains the following –

1. Fields
2. Constructors
3. Functions

Example – Use of class field, constructor and function i.e.

//Example 1- A simple class based example.

```
class User { // Calss
   name: string; //field
   constructor(nameTxt: string) { //constructor
      this.name = nameTxt;
   }

   getName() {//function
      return "Hello, " + this.name;
   }
}

let user = new User("Anil");//Creating Instance objects
```

17.How Static class in typescript?

We can define a class with static properties i.e.

```
export class Constants {
   static baseUrl = 'http://localhost:8080/';
   static date = new Date();
}
```

18.Ways to declare a nest class structure in typescript?

//Example 1-

```
declare module a{

   class b {
   }

   module b {
      class c {
      }
   }
}

var clB = new a.b();
var clC = new a.b.c();
```

//Example 2-

```
export module a {
   export class b {
   }

   export module b {
      export enum c {
         C1 = 1,
         C2 = 2,
         C3 = 3,
      }
   }
}
```

//Example 3-

```
class A {
   static B = class { }
}

var a = new A();
var b = new A.B();
```

19.What is TypeScript? Why should I use Typescript ?

TypeScript is a strongly typed, object oriented and compiled language and this language developed and maintained by Microsoft. It was designed by "Anders Hejlsberg" at Microsoft.

It is a superset of JavaScript. The TypeScript is JavaScript and also has some additional features like static typing and class-based object-oriented programming, automatic assignment of constructor parameters and assigned null values and so on.

20.Why should I use Typescript? What are the Benefits of Using TypeScript?

1. Supports Object Oriented Programming.
2. Typescript adds static typing to JavaScript. Having static typing makes easier to develop and maintain complex apps.
3. Angular2 uses TypeScript a lot to simplify relations between various components and how the framework is built in general.
4. Provide an optional type system for JavaScript.
5. Provide planned features from future JavaScript editions to current JavaScript engines.
6. Supports type definitions.

21.TypeScript Advantages - Pros and Cons!

The Advantages of TypeScript -

1. It is purely object-oriented programming.
2. It is support static type-checking.
3. It can be used for client-side and server-side development.
4. Build-in Support for JavaScript Packaging
5. It offers a "compiler" that can convert to JavaScript-equivalent code.
6. It has an API for DOM manipulation.
7. It has a namespace concept by defining a "Module".
8. Superset of JavaScript
9. ES6 features support

22.What is an Interface in TypeScript?

An interface in TypeScript is similar to other object oriented programming languages interfaces.
An interface is a way to define a contract on a function with respect to the arguments.

In the below example, I am using an interface that describes objects that have a “name”, “age” and “address” fields and the following code defines an interface and a function that takes a parameter that adheres to that interface.

```
//USER INTERFACE
interface User {
   name: string;
   age: number;
   address: string
}
```

```
//FUNCTION USING USER INTERFACE
let userInfo = function(user: User) {
  let info = "Hello, " + user.name + " Your Age is - " + user.age + " and Address is -" +
user.address;

  return info;
}
```

```
//USER INFO JSON OBJECT
let info = {
   name: "Anil",
   age: 30,
   address: "Noida, India."
};
```

```
//RESULT
console.log(userInfo(info));
```

23.What is Functions in TypeScript? How many types you defined in TypeScript?

A function is a set of statements that perform a specific task and used to create readable, maintainable and re-usable code.

A function declaration tells to compiler about a function name, return type, and parameters.

TypeScript functions are almost similar to JavaScript functions but there are different ways of writing functions in TypeScript.

Different types of functions are available in the TypeScript i.e.

1. Normal function
2. Anonymous Function
3. Named Function
4. Lambda Function/Arrow Function
5. Class Function
6. Optional Parameters
7. Rest Parameters
8. Default Parameters

Anonymous Functions–

An anonymous function is a function that was declared without any named identifier to refer to it.

Example - Normal function

```
function printHello() {
   console.log('Hello Anil!');
}

printHello();
```

Examples - Anonymous function

JavaScript -

```
var hello = function () {
   console.log('Hello Anil!, I am Anonymous.');
};

hello();//Return - Hello Anil!, I am Anonymous.

OR

setTimeout(function () {
   console.log('Hello Anil!, I am Anonymous.');
}, 2000); //Return - Hello Anil!, I am Anonymous.
```

TypeScript–

```
var anonymousFunc = function (num1: number, num2: number): number {
   return num1 + num2;
}

//RESULT
console.log(anonymousFunc(10, 20)); //Return is 30

//RESULT
console.log(anonymousFunc(10, "xyz"));
// error: Argument of type 'number' is not assignable to parameter of type 'string'.
//because return type is number for anonymous function).
```

Named Function -

The named function is very similar to the JavaScript function and only one difference - we must declare the type on the passed parameters.

Example – JavaScript

```
function addTwoNumer(num1, num2) {
   return num1 + num2;
}

Example - TypeScript
function addTwoNumer(num1: number, num2: number): number {
   return num1 + num2;
}
```

Lambda Function/Arrow Function -

The arrow function is additional feature in typescript and it is also known as a lambda function.
A lambda function is a function without a name.

```
var addNum = (n1: number, n2: number) => n1 + n2;
```

In the above, the “=>” is a lambda operator and (n1 + n2) is the body of the function and (n1: number, n2: number) are inline parameters.

For example –

```
let addNum = (n1: number, n2: number): number => { return n1 + n2; }
let multiNum = (n1: number, n2: number): number => { return n1 * n2; }
```

```
let dividNum = (n1: number, n2: number): number => { return n1 / n2; }

addNum(10, 2);// Result - 12
multiNum(10, 2);// Result - 20
multiNum(10, 2);// Result - 5
```

Optional Parameters Function -

We can specify optional properties on interfaces and the property may be present or missing in the object.

In the below example, the address property is optional on the following “User” interface.

For Example as,

```
//USER INTERFACE
interface User {
   name: string;
   age: number;
   address?: string //Optional
}

//FUNCTION USING USER INTERFACE
let userInfo = function(user: User) {
  let info = "Hello, " + user.name + " Your Age is - " + user.age + " and Address is -" +
user.address;

  return info;
}

//USER INFO JSON OBJECT
let info = {
   name: "Anil",
   age: 30
};

//RESULT
console.log(userInfo(info));

//USER INTERFACE
interface User {
   name: string;
   age: number;
   address?: string //Optional
}
```

```
//FUNCTION USING USER INTERFACE
let userInfo = function(user: User) {
  let info = "Hello, " + user.name + " Your Age is - " + user.age + " and Address is -" +
user.address;

  return info;
}

//USER INFO JSON OBJECT
let info = {
   name: "Anil",
   age: 30
};

//RESULT
console.log(userInfo(info));
-
```

Rest parameters-

The Rest parameters do not restrict the number of values that we can pass to a function and the passed values must be the same type otherwise throw the error.

For Example as,

```
//Rest Parameters
let addNumbers = function(...nums: number[]) {
   let p;
   let sum: number = 0;

   for (p = 0; p < nums.length; p++) {
      sum = sum + nums[p];
   }

   return sum;
}

//The Result
addNumbers(1, 2);
addNumbers(1, 2, 3);
addNumbers(1, 12, 10, 18, 17);
```

Default Parameters -

Function parameters can also be assigned values by default.
A parameter can't be declared as optional and default both at the same time.

For Example as,

```
let discount = function (price: number, rate: number = 0.40) {
   return price * rate;
}

//CALCULATE DISCOUNT
discount(500); // Result - 200

//CALCULATE DISCOUNT
discount(500, 0.45); // Result - 225
```

TypeScript - Anonymous Functions

Anonymous Functions–

An anonymous function is a function that was declared without any named identifier to refer to it.

Example - Normal function

```
function printHello() {
   console.log('Hello Anil!');
}

printHello();
```

Examples - Anonymous function

JavaScript -

```
var hello = function () {
   console.log('Hello Anil!, I am Anonymous.');
};

hello();//Return - Hello Anil!, I am Anonymous.

OR
```

```
setTimeout(function () {
   console.log('Hello Anil!, I am Anonymous.');
}, 2000); //Return - Hello Anil!, I am Anonymous.
```

TypeScript –

```
var anonymousFunc = function (num1: number, num2: number): number {
   return num1 + num2;
}

//RESULT
console.log(anonymousFunc(10, 20)); //Return is 30

//RESULT
console.log(anonymousFunc(10, "xyz"));
// error: Argument of type 'number' is not assignable to parameter of type 'string'.
//because return type is number for anonymous function).
```

TypeScript - Optional Parameters Function

Optional Parameters Function -

We can specify optional properties on interfaces and the property may be present or missing in the object.

In the below example, the address property is optional on the following “User” interface.

```
//USER INTERFACE
interface User {
   name: string;
   age: number;
   address?: string //Optional
}

//FUNCTION USING USER INTERFACE
let userInfo = function(user: User) {
  let info = "Hello, " + user.name + " Your Age is - " + user.age + " and Address is -" +
user.address;

  return info;
}

//USER INFO JSON OBJECT
let info = {
```

```
    name: "Anil",
    age: 30
};

//RESULT
console.log(userInfo(info));
```

How to create fields, constructor and function in TypeScript Class?

24.What is class in TypeScript?

A class is a template definition of the methods and variables in a particular kind of object. It is extensible program, code and template for creating objects.

A TypeScript is object oriented JavaScript and it also supports object oriented programming features like classes, interfaces, etc.

A class captains the Public, Private, Protected and Read-only modifiers and Public by default. You can see the below example, the class User and each members are public by default.

A class definition can contains the following –

1. Fields
2. Constructors
3. Functions

Example – Use of class field, constructor and function i.e.

```
//Example 1- A simple class based example.
class User { // Calss
    name: string; //field
    constructor(nameTxt: string) { //constructor
        this.name = nameTxt;
    }

    getName() {//function
        return "Hello, " + this.name;
    }
}

let user = new User("Anil");//Creating Instance objects
```

25.How Static class in typescript?

We can define a class with static properties i.e.

```
export class Constants {
   static baseUrl = 'http://localhost:8080/';
   static date = new Date();
}
```

26.Ways to declare a nest class structure in typescript?

```
//Example 1-
declare module a{

   class b {
   }

   module b {
      class c {
      }
   }
}

var clB = new a.b();
var clC = new a.b.c();
```

```
//Example 2-
export module a {
   export class b {
   }

   export module b {
      export enum c {
         C1 = 1,
         C2 = 2,
         C3 = 3,
      }
   }
}
```

```
//Example 3-
class A {
   static B = class { }
}

var a = new A();
var b = new A.B();
```

TypeScript Method Overriding

Method Overriding -

In Method Overriding, redefined the base class methods in the derive class or child class.

Example as,

```
class NewPrinter extends Printer {
   doPrint(): any {
      super.doPrint();
      console.log("Called Child class.");
   }

   doInkJetPrint(): any {
      console.log("Called doInkJetPrint().");
   }
}

let printer: new () => NewPrinter;
printer.doPrint();
printer.doInkJetPrint();
```

Automatic Assignment of Constructor Parameters in TypeScript

27.What is Parameter Property TypeScript?

The TypeScript has so many useful features which not have in other programming languages. The TypeScript has an automatic assignment of constructor parameters that is called “Parameter Property”.

It is automatic assignment of constructor parameters to the relevant property. It is great but it is different to other languages.

Examples as,

Declaring a class with constructor arguments in C# and other programming language as,

```
class Customer {
   _name: string;
   _age: number;
   _adrress: string;

   constructor(name: string, age: number, adrress: string) {
      this.name = _name;
      this.age = _age;
      this.adrress = _adrress;
   }
}
```

Declaring a class with constructor arguments in TypeScript –that is called automatic parameter assignment as,

```
export class Customer {
   constructor(private name: string, age: number, private adrress: string) { }
}
```

You can take a look at this on the JavaScript.

```
var Customer = (function () {
   function Customer(name, age, adrress) {
      this.name = name;
      this.age = age;
      this.adrress = adrress;
   }

   return Customer;
})();
```

Public, Private, and **Protected** modifiers as

1. Public - accessible outside of the class
2. Private - only accessible in the class only

3. Protected - accessible in the class and the derived classes

Public modifier by default - When you are not put a modifier (public, private or protected) on your member definition then TypeScript will choose the public by default.

28.Which access modifiers are implied when not specified?

Everything in a class is public if not specified. Everything in a module is private unless export keyword is used.

29.What is the purpose of the public access modifier for classes in Typescript?

Your sample code means exactly the same in TypeScript. When you don't put a modifier public, private or protected on your member definition then TypeScript will choose the default one which is public.

Pros and Cons of Ahead-of-Time!

34.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.

35.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.
5. And so on.

36.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.

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

As like your other web applications, you should flow in angular 2 applications also.

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.

There are more advantages over performance, template system, application debugging, testing, components and nested level components.

For Examples as,

```
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.

For example as,
//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>
```

38.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.

39.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.

40.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
2. Component
3. Template
4. Outpouts
5. Data Binding
6. Directive
7. Service
8. Dependency Injection

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

41.Modern browsers are supported in Angular 2?

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.

42.Why should you use Angular 2 ? What are the Advantages of Angular 2 ?

The core differences and many more advantages on Angular 2 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.

Angular 1 and Angular 2 Integration

1. Angular frameworks provide the support of mixing code of Angular 1 and Angular 2 in the same application.
2. We can write the mixing components of Angular 1 and Angular 2 in the same view.
3. We can inject services across frameworks of Angular 1 and Angular 2 in the same application.
4. Both Angular 1's and Angular 2's data binding works across frameworks in the same view.

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

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(){

  }
}
```

43.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.”

For example as,

```
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>
```

Angular 2 Component Lifecycle Hooks

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().

Example as,

```
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 :-

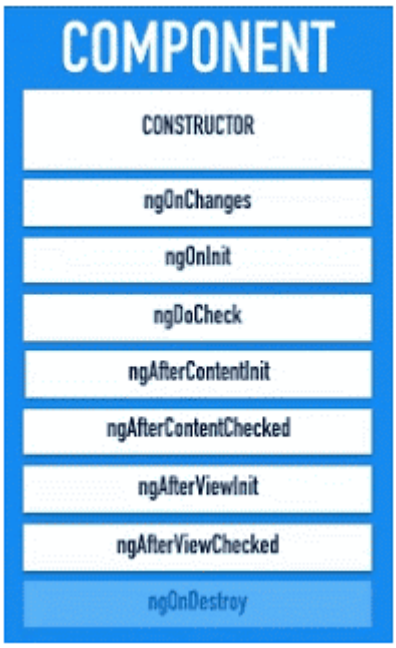
The ngDestroy directive is called in a component lifecycle just before the instance of the component is finally destroyed.

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:

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

44.What is the Best way to Declare and Access a Global Variable in Angular 2?

This post helps us to learn “Declare and Access a Global Variable in Angular 2” using “Typescript” and also share the steps to create and use of this global variables.

Steps –

1. Create Global Variables.
2. Import and Use the Global Variables in the Component.
3. Result

Create Global Variables :- "app.global.ts"

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

@Injectable()
export class AppGlobals {
   readonly baseAppUrl: string = 'http://localhost:57431/';
   readonly baseAPIUrl: string = 'https://api.github.com/';
}
```

Import and Use the Global Variables in the Component:- "user.component.ts"

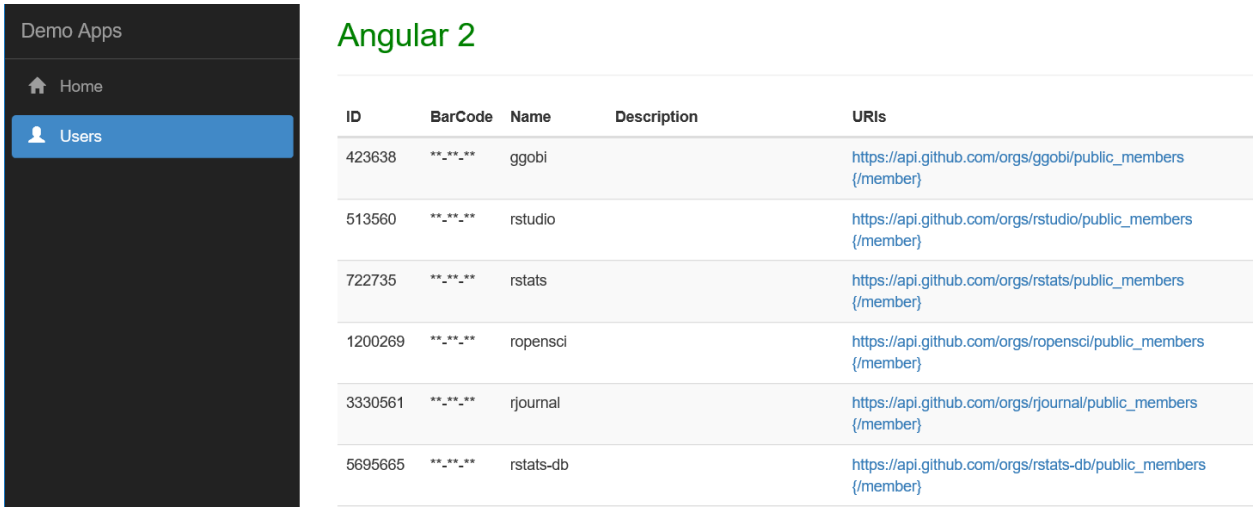
```
import { Component, Injectable} from '@angular/core';
import { CommonModule } from '@angular/common';
import { HttpModule, Http } from '@angular/http';
import { UserService } from '../service/user.service';
import { AppGlobals } from '../shared/app.globals';


@Component({
   selector: 'user',
   templateUrl: './user.component.html',
   styleUrls: ['./user.component.css'],
   providers: [UserService, AppGlobals]
})

export class UserComponent {
   //USERS DECLARATIONS.
   users = [];

   //HOME COMPONENT CONSTRUCTOR
   constructor(private userService: UserService, private _global: AppGlobals) { }

   //GET USERS SERVICE ON PAGE LOAD.
   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));
   }
}
//END BEGIN - USERCOMPONENT
```

"user.server.ts" :-

```
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
```

Result:-



45.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

1. Smaller & Faster Apps - Angular 4 applications is smaller & faster in comparison with Angular 2.

2. View Engine Size Reduce - Some changes under to hood to what AOT generated code compilation that means in Angular 4, improved the compilation time. These changes reduce around 60% size in most cases.

3. Animation Package- Animations now have their own package i.e. @angular/platform-browser/animations

4. Improvement - Some Improvement on *ngIf and *ngFor.

5. Template - The template is now ng-template. You should use the “ng-template” tag instead of “template”. Now Angular has its own template tag that is called “ng-template”.

6. NgIf with Else – Now in Angular 4, possible to use an else syntax as,

<div *ngIf="user.length > 0; else empty"><h2>Users</h2>
<ng-template #empty><h2>No users.</h2></ng-template>

7. AS keyword – A new addition to the template syntax is the “as keyword” is use to simplify to the “let” syntax.

Use of as keyword,

```
<div *ngFor="let user of users | slice:0:2 as total; index as = i">
  {{i+1}}/{{total.length}}: {{user.name}}
</div>

To subscribe only once to a pipe "|" with "async" and If a user is an observable, you can now use to write,

<div *ngIf="users | async as usersModel">
  <h2>{{ usersModel.name }}</h2> <small>{{ usersModel.age }}</small>
</div>
```

8. Pipes - Angular 4 introduced a new “titlecase” pipe “|” and use to changes the first letter of each word into the uppercase.

The example as,

```
<h2>{{ 'anil singh' | titlecase }}</h2>
<!-- OUPPUT - It will display 'Anil Singh' -->
```

9. Http - Adding search parameters to an “HTTP request” has been simplified as,

```
//Angular 4 -
http.get(`${baseUrl}/api/users`, { params: { sort: 'ascending' } });

//Angular 2-
const params = new URLSearchParams();
params.append('sort', 'ascending');
http.get(`${baseUrl}/api/users`, { search: params });
```

10. Test- Angular 4, overriding a template in a test has also been simplified as,

```
//Angular 4 -
TestBed.overrideTemplate(UsersComponent, '<h2>{{users.name}}</h2>');

//Angular 2 -
TestBed.overrideComponent(UsersComponent, {
  set: { template: '<h2>{{users.name}}</h2>' }
});
```

11. Service- A new service has been introduced to easily get or update “Meta Tags” i.e.

```
@Component({
  selector: 'users-app',
  template: `<h1>Users</h1>`
})
export class UsersAppComponent {
  constructor(meta: Meta) {
    meta.addTag({ name: 'Blogger', content: 'Anil Singh' });
  }
}
```

12. Forms Validators - One new validator joins the existing “required”, “minLength”, “maxLength” and “pattern”. An email helps you validate that the input is a valid email.

13. Compare Select Options - A new “compareWith” directive has been added and it used to help you compare options from a select.

```
<select [compareWith]="byUId" [(ngModel)]="selectedUsers">
  <option *ngFor="let user of users" [ngValue]="user.UId">{{user.name}}</option>
</select>
```

14. Router - A new interface “paramMap” and “queryParamMap” has been added and it introduced to represent the parameters of a URL.

```
const uid = this.route.snapshot.paramMap.get('UId');
this.userService.get(uid).subscribe(user => this.name = name);
```

15. CanDeactivate - This “CanDeactivate” interface now has an extra (optional) parameter and it is containing the next state.

16. I18n - The internationalization is tiny improvement.

```
//Angular 4-
<div [ngPlural]="value">
  <ng-template ngPluralCase="0">there is nothing</ng-template>
  <ng-template ngPluralCase="1">there is one</ng-template>
</div>

//Angular 2-
<div [ngPlural]="value">
  <ng-template ngPluralCase="=0">there is nothing</ng-template>
  <ng-template ngPluralCase="=1">there is one</ng-template>
</div>
```

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

Angular 5 is going to be a much better Angular and you will be able to take advantage of it much easier and Version 5 will be fully released on September/October 2017.

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 5 Performance Improvements - Angular 5

1. Use of addEventListener for the faster rendering and it is the core functionality.
2. Update to new version of build-optimizer.
3. Added some Improvements on the abstract class methods and interfaces
4. Remove decorator DSL which depends on Reflect for Improve the Performance of Apps and This is the core functionality.
5. Added an option to remove blank text nodes from compiled templates
6. Switch Angular to use Static-Injector instead of Reflective-Injector.
7. Improve the applications testing.
8. Improve the performance of hybrid applications
9. Improvements on Lazy loading for Angular
10. And so on...

Some **Improvement on HttpClient** – This is used for Applications communicate with backend services over the HTTP protocol!

1. Improvement on Type-checking the response
2. Improvement on Reading the full response
3. Improvement on Error handling and fetching error details
4. Improvement on Intercepting all requests or responses
5. Improvement on Logging
6. Improvement on Caching
7. Improvement on XSRF Protection

Added Features - Angular 5

1. Added Representation of Placeholders to xliff and xmb in the compiler
2. Added an Options Arg to Abstract Controls in the forms controls
3. Added add default updateOn values for groups and arrays to form controls
4. Added updateOn blur option to form controls
5. Added updateOn submit option to form controls
6. Added an Events Tracking Activation of Individual Routes
7. Added NgTemplateOutlet API as stable in the common controls
8. Create StaticInjector which does not depend on Reflect polyfill
9. Added [@.disabled] attribute to disable animation children in the animations
10. And so on..

Router Life Cycle Events – Angular 5

Added new router life cycle events for Guards and Resolvers -

1. GuardsCheckStart,
2. GuardsCheckEnd,
3. ResolveStart and
4. ResolveEnd

Angular 5 Bug Fixes - Angular 5

1. Fixed compilation error by using the correct type for providers
2. Skip PWA test when redeploying non-public commit
3. Don't strip CSS source maps. This is the compiler related fix
4. Remove tsickle (language-service) dependency
5. Support persisting dynamic styles within animation states
6. Ignore @import in multi-line css
7. Fix platform-browser-dynamic
8. Forbid destroyed views to be inserted or moved in VC
9. Support persisting dynamic styles within animation states
10. And so on...

Componenet Questions

Angular 2 Components Example

In Angular 2, the components are the main way to build or specify HTML elements and business logic on the page.

In AngularJs 1, we are handling using scope, directives and controllers but all those concepts are using in a single combined that is called components.

The component is the core functionality of Angular 2 app but we need to know to pass the data in to the components to configure them.

To build an Angular 2 application you define a set of components, for every HTML elements, views, and route.

Angular 2 applications must have a root component that contains all other components. That means all Angular 2 applications have a component tree.

Application è Component è Component1 and Component2

Example of Components

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';

@Component({
   selector: 'home',
   templateUrl: './home.component.html',
   styleUrls: ['./home.component.css'],
})
```

```
export class HomeComponent {
   userlist: Users[];

   constructor() {
      this.userlist = [
         { Id: '1001', name: 'Anil Singh', site: 'http://www.code-sample.com' },
         { Id: '1002', name: 'Alok', site: 'http://www.code-view.com' },
         { Id: '1003', name: 'Reena', site: 'http://www.code-sample.xyz' },
         { Id: '1004', name: 'Dilip', site: 'http://www.codefari.com' },
      ];
   }

   values = '';
   onKeyUp(event: any) {
      this.values = event.target.value;
      console.log(this.values);
   };

   onKeyDown(event: any) {
      this.values = event.target.value;
      console.log(this.values);
   };
}

export class Users {
   Id: String;
   name: String;
   site: String;
}

/* For HTML Components
 <input type="text" [(value)]="values" (keyup)="onKeyUp($event)"
(keydown)="onKeydown($event)" />
*/
```

Angular 2 Component Summary

- Angular 2 Component meta-data annotation is used to register the components.
- Angular 2 components are used to create UI widgets.
- Angular 2 components are used to split to application into smaller parts.
- Only one component is used per DOM element.
- In the Angular 2 components, @View, template and templateUrl are mandatory in the components.

Angular 2 components vs directive

@Components
@Directive

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.

Example for using Component.

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

  @Component({
    selector: 'hello-world'
  })

  @View({
    template: "<h1>Hello  {{angular}}</h1>"
  })

  class hello {
     constructor(public angular: string) {}
  }

  <hello-world></hello-world>
```

Example for using Directive.

```
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>
```

Angular 2 Component Lifecycle Hooks [Examples Also]

The common questions ask bye most of Angular 2 lovers,

"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.

48.What Are Components in Angular 5,4 and 2?

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().

Example as,

```
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:-

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

Angular 2 @Inputs

49.How to Passing data into Angular 2 components with @Input?

@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]

@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>
```

50.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 @Outputdecorator.

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);
```

Angular 2 Hidden Attribute with Multiple Examples

Angular 2 [hidden] is a special case binding to hidden property.

It is closest cousin of ng-show and ng-hide.

It is more powerful to bind any property of elements. Both the ng-show and ng-hide are used to manage the visibility of elements using ng-hide css class. It is also set the display property “display:none”.

Stayed Informed - Angular 2 @Inputs
All the above features are supported in Angular 2 but added some extra feature like animations etc.

Syntax:-

```
<div [hidden]="!active">
   Hello, this is active area!
</div>
```

Note:
- Don't use hidden attribute with Angular 2 to show/hide elements.

51.Don't use hidden attribute with Angular 2. Here is why?

The hidden attribute is used to hide elements. Browsersot supere is whyposed to display elements that have the hidden attribute specified. Browsers attach "display: none" styles to elements with hidden attribute.

Example,

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

@Component({
  selector: 'demo',
  templateUrl: 'app/component.html'
})
export class MainComponent {
  Ishide: true;
}

<div [hidden]="Ishide">
    Hey, I'm using hidden attribute.
</div>
```

Works great but some time its override hidden attribute with some css and that time behave wrong!..

For example,

Be sure to don't have a display css rule on your <p> tags who override hidden behaviour like i.e.

```
p {
   display: inline-block !important;
}
```

The above hidden html attributes acts like display: none;

52.How do Components Communicate with Each Other in Angular 2?

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();
  }
}
```

53.How do we display errors in a component view with Angular 2? How To Display Validation and Error Messaging on form submit in Angular 2?

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>
```

What is Angular 2 Service? [Steps For Create and Use of Angular 2 Services]

55.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.

56.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.

57.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 :-

```
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',
```

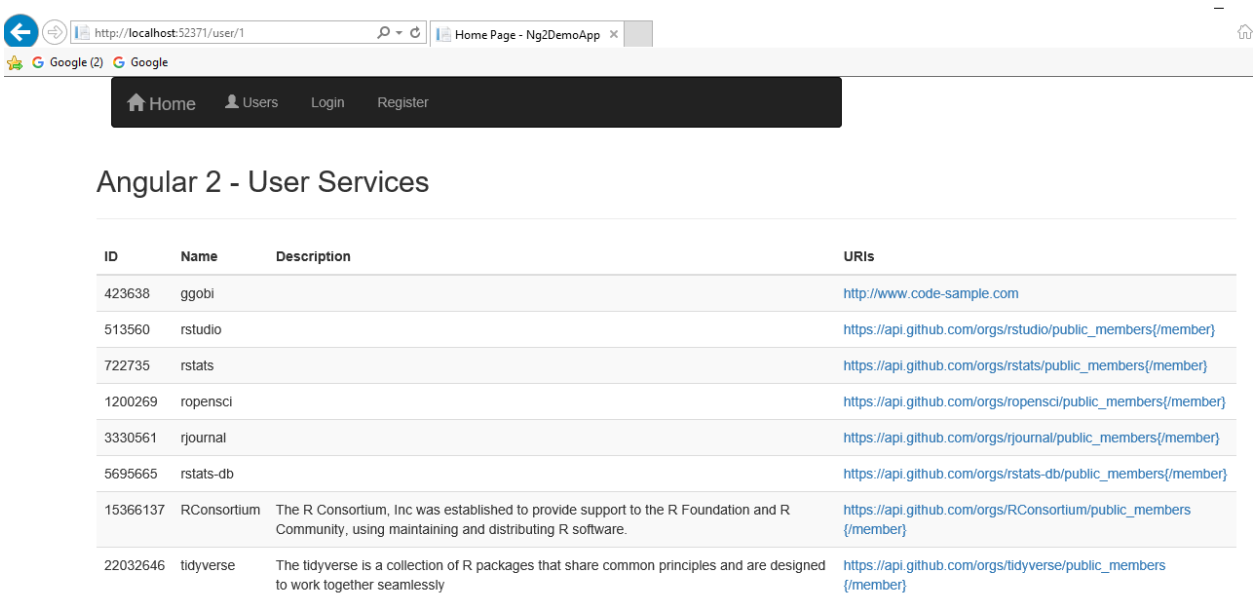
```
  templateUrl: './user.component.html',
  styleUrls: ['./user.component.css'],
  providers: [UserService]
})

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>
```

Result :-



Pipes Questions

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

58.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.

Filter/Pipe Name	Angular 1.x	Angular 2
currency	✓	✓
date	✓	✓
uppercase	✓	✓
json	✓	✓
limitTo	✓	✓
lowercase	✓	✓
number	✓	
orderBy	✓	
filter	✓	
async		✓
decimal		✓
percent		✓

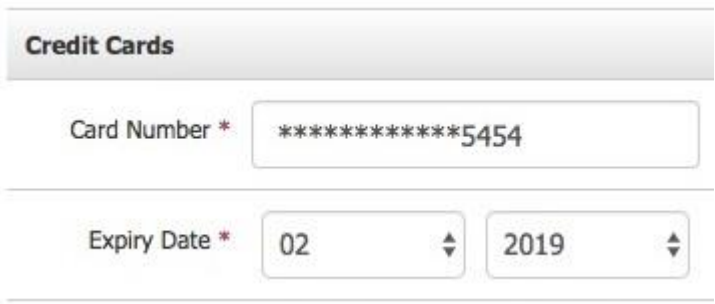
59.Why use Pipes?

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.



60.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): '')
  }
}
```

61.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);
  }
}
```

62.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>
```

Angular 2 - User Services

ID	Name	Description	URIs
....-	ggobi		https://api.github.com/orgs/ggobi/public_members{/member}
....-	rstudio		https://api.github.com/orgs/rstudio/public_members{/member}
....-	rstats		https://api.github.com/orgs/rstats/public_members{/member}
....-	ropensci		https://api.github.com/orgs/ropensci/public_members{/member}
....-	rjournal		https://api.github.com/orgs/rjournal/public_members{/member}
....-	rstats-db		https://api.github.com/orgs/rstats-db/public_members{/member}
....-	RConsortium	The R Consortium, Inc was established to provide support to the R Foundation and R Community, using maintaining and distributing R software.	https://api.github.com/orgs/RConsortium/public_members{/member}
....-	tidyverse	The tidyverse is a collection of R packages that share common principles and are designed to work together seamlessly	https://api.github.com/orgs/tidyverse/public_members{/member}

Directive Questions

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.

63.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.

64.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.

65.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>
```

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 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,

```
@RouteConfig([
      {path: '/employ/:id', component: employe, name: 'emp'},
])
```

Angular 2 Structural Directives [How To Write Structural Directives?]

66.What are directives?

Angular lets you extend HTML with new attributes called directives.

There are two other kinds of Angular directives,

1. Components
2. Attribute directives

67.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>
```

68.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 { }
}
```

69.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;
 }
}
```

Template Questions

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

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.

71.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;
   }
}
```

72.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);
   }
}
```

Angular 2 templateUrl and styleUrls

The styleUrls is a component which uses to write inline styles, style Urls and template inline style.

The templateUrl is a function which returns HTML template.

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.

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;
   }
}
```

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);
   }
}
```

73.How to use styleUrls and styles in Angular 2?

The Angular 2 “styles” or “styleUrls” should only be used for css rules and it is affect the style of the template elements.

This is the best approaches to add styles directly to the components and the view encapsulation is set per component. It is use for some situations.

An example to add external styles to the components result text color is red,

Syntax –

```
@Component({
   selector: 'home',
   templateUrl: './home.component.html',
   styleUrls: ['./home.component.css'],
   styles: ['.tbl-row-border { border: 1px solid red;}','.txt-color{color:red;}']
})
```

home.component.html :-

```
<div class="row">
<div class="col-lg-12">
<div class="ibox float-e-margins">
<div class="ibox-title">
   <h5>Angular 2 for loop typescript example - *ngFor</h5>
</div>
<div class="ibox-title">
   <input type="text" [(value)]="values" (keyup)="onKeyUp($event)" /> <strong>Resut-
{{values}}</strong>
</div>
<div class="ibox-content">
   <div class="table-responsive">
      <table class="table table-striped">
         <thead>
            <tr>
```

```
            <th>ID</th>
            <th>Name </th>
            <th>SiteUrl </th>
            <th>Actions </th>
          </tr>
        </thead>
        <tbody>
          <tr *ngFor="let user of userlist; let i = index" class="tbl-row-border">
            <td>{{user.Id}}</td>
            <td>{{user.name}}</td>
            <td><a href="{{user.site}}" target="_blank">{{user.site}}</a></td>
            <td><a (click)="addUser(user)">A</a>|<a
(click)="updateUser(user)">E</a>|<a (click)="deleteUser(user)">D</a></td>
          </tr>
        </tbody>
      </table>
    </div>
</div>
</div>
</div>
</div>
```

home.component.ts :-

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'home',
  templateUrl: './home.component.html',
  //styleUrls: ['./home.component.css'],
  styles: ['.tbl-row-border { border: 1px solid red;}']
})

export class HomeComponent {
  userlist: Users[];

  constructor() {
    this.userlist = [{ Id: '1001', name: 'Anil SIngh', site: 'http://www.code-sample.com' },
    { Id: '1002', name: 'Alok', site: 'http://www.code-view.com' },
    { Id: '1003', name: 'Reena', site: 'http://www.code-sample.xyz' },
    { Id: '1004', name: 'Dilip', site: 'http://www.codefari.com' },
    ];
  }

  values = '';
```

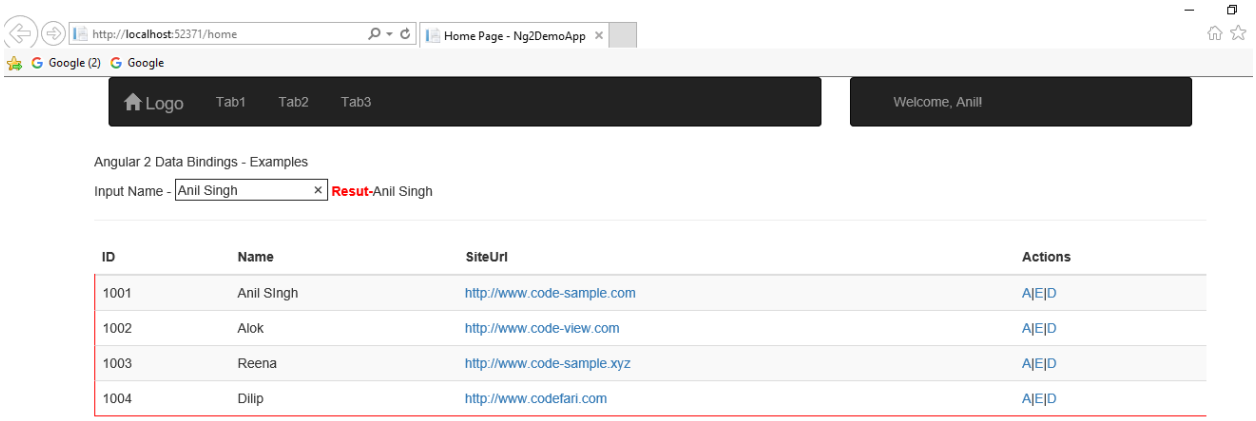
```
  onKeyUp(event: any) {
    this.values = event.target.value;
  };

  addUser(user) {
    alert(JSON.stringify(user));
  };

  updateUser(user) {
    alert(JSON.stringify(user));
  };

  deleteUser(user) {
    alert(JSON.stringify(user));
  };
}

export class Users {
  Id: String;
  name: String;
  site: String;
}
```

app.module.ts :-

```
import { NgModule } from '@angular/core';
import { RouterModule } from '@angular/router';
import { UniversalModule } from 'angular2-universal';
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';

@NgModule({
  bootstrap: [ AppComponent ],
  declarations: [
    AppComponent,
    HomeComponent,
    HeaderComponent,
    MenuComponent
  ],
  imports: [
    UniversalModule, // Must be first import. This automatically imports BrowserModule,
HttpModule, and JsonpModule too.
    RouterModule.forRoot([
      { path: '', redirectTo: 'home', pathMatch: 'full' },
```

```
        { path: 'home', component: HomeComponent },
        { path: '**', redirectTo: 'home' }
      ])
  ]
})
export class AppModule {
}
```

Result



74.How To Import CSS using System.import?

You do need the “System.import” to bootstrap and run your application.

It can't run without it, and if it does, you might have a cashed version in your browser.

The error is -
This error indication that some of the script files didn't get loaded correctly!

syntax error: unexpected token <

Syntax is -

```
System.import('./app/bootstrap/css/boots-trap.css!').then(() => {
  System.import('./app/main-app.css!');
});
```

Load external css style into Angular 2 components

The “styles” or “styleUrls” should only be used for css rules and it is affect the style of the template elements.

This is the best approaches to add styles directly to the components and the view encapsulation is set per component. It is use for some situations.

An example to add external styles to components as,

```
@Component({
   selector: 'app',
   templateUrl: 'app/login.html',
   styleUrls: [
      'app/app.css',
      'app/main.css'
   ],
   encapsulation: ViewEncapsulation.None,
})

export class Component {}
```

Routing Questions

Angular 2 Routing Concepts and Examples

This post helps us to learn application “Routings” in Angular 2. In the below routing example I am using Angular 2 for the client side and ASP.NET Core with single page application (SPA) for the server application.

“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>
```

75.What is 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 as,

```
@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
    ]

})
```

76.What is 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 as,

```
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
  ]
})
```

77.What is router-outlet directive in Angular 2?

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>
```

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

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 { }
```

79.What is Router-link directive in Angular 2?

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

Syntax:-

```
<router-link> </router-link>
```

Example as,

```
<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>
```

Dependency Questions

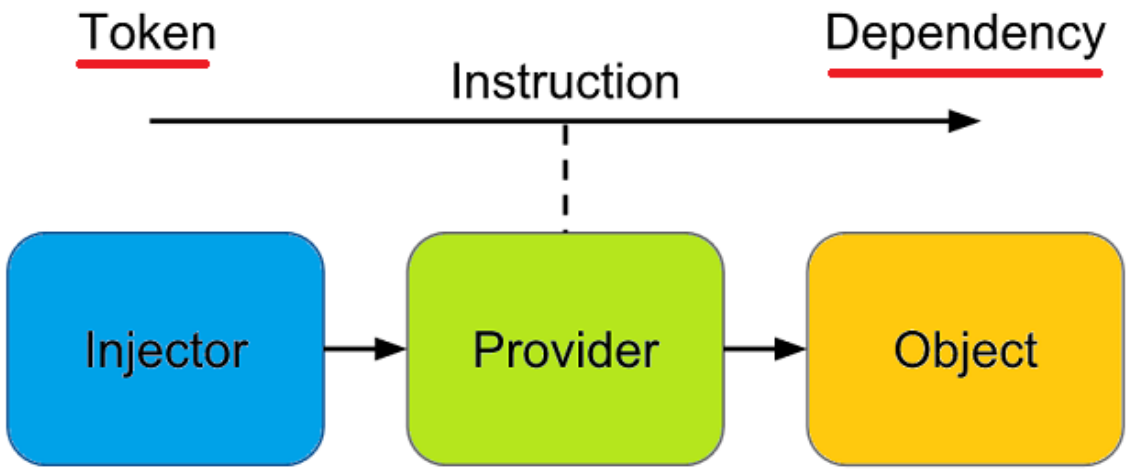
Dependency Injection (DI) in Angular 2 [Why @Injectable()?]

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.



Angular 2 Dependency Injection

@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.

80.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;
   };
}
```

Angular 2 Injectable - Why @Injectable ()?

81.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());
  }
}
```

Angular 2 Inject

82. 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) {
  }
}
```

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

83.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) {
   }
}
```

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

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

Angular 2 @NgModel [Purpose Of Root Module & Export Module]

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 { }
```

85.Why Angular 2 modules needed?

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

86.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

87.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.

88.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.

89.How to declaration Module?

```
import {NgModule, ApplicationRef} from '@angular/core';
import {CommonModule} from '@angular/common';
import {FormsModule} from '@angular/forms';
import {MaterialModule} from '@angular2-material/module';
import {AppComponent} from './app.component';
```

```
@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>;
}
```

90.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

Angular 2 Modules vs. JavaScript Modules vs. Angular 1 Modules

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.

Ng-Zones Questions

91.What are Zones? What is Change Detection? What is NgZone run outside ? 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.

92.What is Change Detection?

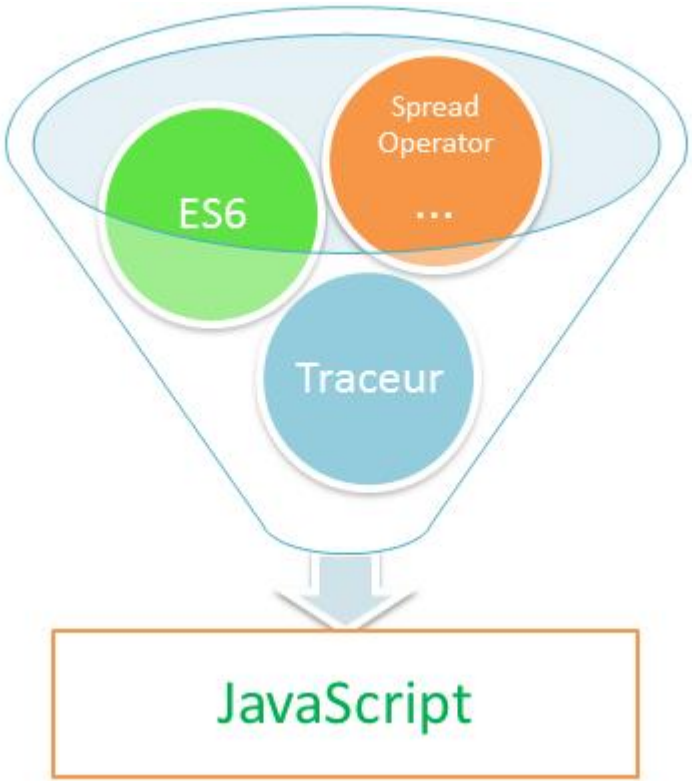
Angular 2 runs inside of its own special zone called NgZone and this special zone extends the basic functionality of a zone to facilitate change detection.

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.

Traceur compiler Questions

95.What is Traceur compiler in Angular 2 ?

The Traceur is a JavaScript compiler. The Traceur compiler used to allow us to use the features from the future. The Traceur compiler is fully supported to ECMAScript(ES6) and ES.vNext also.



The main goal of Traceur compiler is to inform the designs of new JavaScript features and also allow us to write the code in better manners and it also prefer, tell us to use design patterns.

Now the days Traceur compiler are broadly used in Angularv2.0 because Angular v2.0 are fully used to ES5 and ES6.

Angular Testing Questions

96.What Are Isolated Unit Tests? 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.

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 ...
});
```

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

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

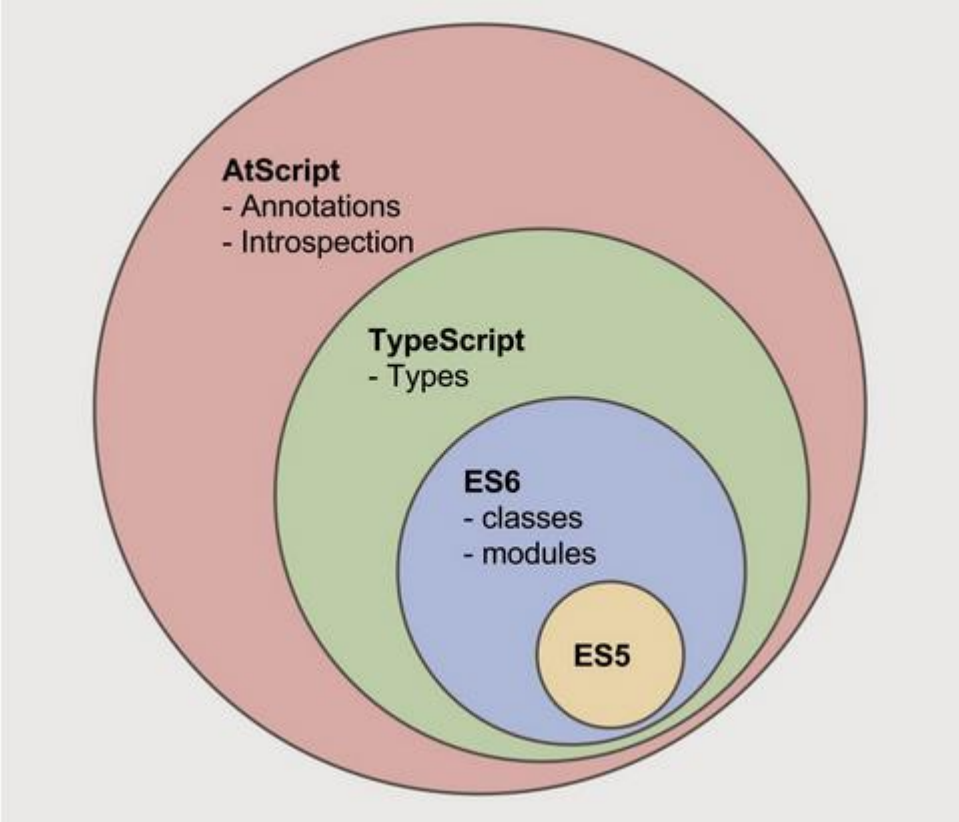
Example As –

```
beforeEach(() => {
  fixture = TestBed.configureTestingModule({
    declarations: [YourComponent ]
  })
  .createComponent(YourComponent);
});
```

ECMAScript and TypeScript Questions

98.What is ECMAScript (ES6) in Angular 2?

The ECMAScript is known as now ES6. The ES6 is version 6. The ES6 is a scripting language and it developed by Ecma International org.



The JavaScript is an implementation of ES6. The ES6 features are fully supported to latest browsers(chrome, Firefox etc.)

A basic simple example with live demo of Add two numbers using ES6 as given below.

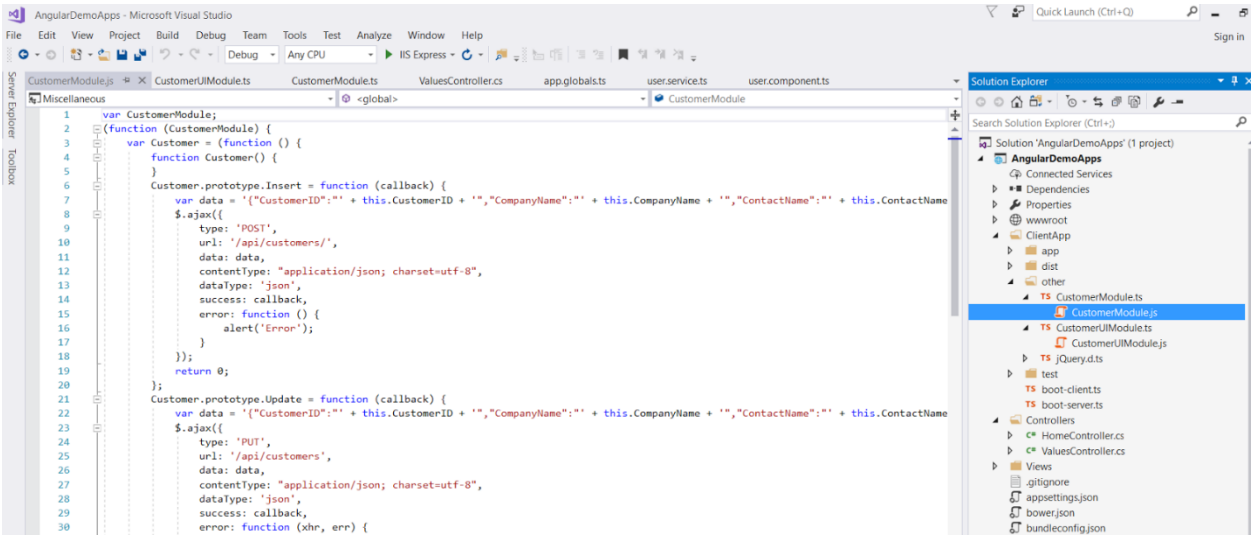
```
let AddTwoNum = (num1,num2) => num1+num2;
 console.log(AddTwoNum(4,3));
```

99.What is TypeScript in Angular? What is TypeScript?

TypeScript is a strongly typed, object oriented and compiled language and this language developed and maintained by Microsoft. It was designed by “Anders Hejlsberg” at Microsoft.

It is a superset of JavaScript. The TypeScript is JavaScript and also has some additional features like static typing and class-based object-oriented programming, automatic assignment of constructor parameters and assigned null values and so on.

The entire JavaScript program is valid for TypeScript because the entire TypeScript (.ts) file converted to JavaScript (.js) file after compiled source compiled and this process is automatic.
See in the below compiled project pic.



TypeScript adds support for features such as classes, modules and arrow function syntax as proposed in the ECMAScript 2015 standard.

100.How to use arrow function?

```
//Interface
interface IStudent {
   yearOfBirth: number;
   age : () => number;
}

//Base Class
class College {
   constructor(public name: string, public city: string) {
   }
}

//Child Class implements IStudent and inherits from College
class Student extends College implements IStudent {
   firstName: string;
   lastName: string;
   yourAge: number;
```

```
  //Constructor
  constructor(firstName: string, lastName: string, name: string, city: string, yourAge:
number) {
    super(name, city);

    this.firstName = firstName;
    this.lastName = lastName;
    this.yourAge = yourAge;
  }

  age () {
    return this.yourAge;
  }
}
```

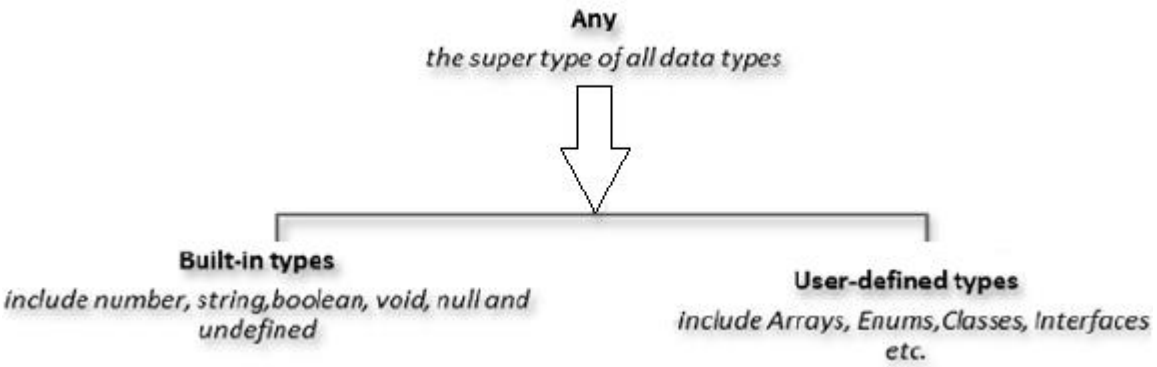
101.Why should I use Typescript? What are the Benefits of Using TypeScript?

1. Supports Object Oriented Programming.
2. Typescript adds static typing to JavaScript. Having static typing makes easier to develop and maintain complex apps.
3. Angular2 uses TypeScript a lot to simplify relations between various components and how the framework is built in general.
4. Provide an optional type system for JavaScript.
5. Provide planned features from future JavaScript editions to current JavaScript engines.
6. Supports type definitions.

102.What are Types in TypeScript? What are TypeScript Types?

The Type represents the different types of values which are using in the programming languages and it checks the validity of the supplied values before they are manipulated by your programs.

The TypeScript provides data types as a part of its optional type and its provide us some primitive types as well as a dynamic type “any” and this “any” work like “dynamic”.



In TypeScript, we define a variable with a “type” and appending the “variable name” with “colon” followed by the type name i.e.

```
let isActive: boolean = false; OR var isActive: boolean = false;
let decimal: number = 6; OR var decimal: number = 6;
let hex: number = 0xf00d; OR var hex: number = 0xf00d;
let name: string = "Anil Singh"; OR var name: string = "Anil Singh";
let binary: number = 0b1010; OR var binary: number = 0b1010;
let octal: number = 0o744; OR var octal: number = 0o744;
let numlist: number[] = [1, 2, 3]; OR var numlist: number[] = [1, 2, 3];
let arrlist: Array<number> = [1, 2, 3]; OR var arrlist: Array<number> = [1, 2, 3];

//Any Keyword
let list: any[] = [1, true, "free"];
list[1] = 100;

//Any Keyword
let notSureType: any = 10;
notSureType = "maybe a string instead";

notSureType = false; // definitely a Boolean
```

Number: the “number” is a primitive number type in TypeScript. There is no different type for float or double in TypeScript.
Boolean: The “boolean” type represents true or false condition.
String: The “string” represents sequence of characters similar to C#.
Null: The “null” is a special type which assigns null value to a variable.
Undefined: The “undefined” is also a special type and can be assigned to any variable.
Any : this data type is the super type of all types in TypeScript. It is also known as dynamic type and using “any” type is equivalent to opting out of type checking for a variable.

A note about “let” keyword –

You may have noticed that, I am using the “let” keyword instead of “var” keyword. The “let” keyword is actually a newer JavaScript construct that TypeScript makes available. Actually, many common problems in JavaScript are reducing by using “let” keyword. So we should use “let” keyword instead of “var” keyword.

TypeScript Advantages - Pros and Cons!

The Advantages of TypeScript -

1. It is purely object-oriented programming.
2. It is support static type-checking.
3. It can be used for client-side and server-side development.

4. Build-in Support for JavaScript Packaging
5. It offers a “compiler” that can convert to JavaScript-equivalent code.
6. It has an API for DOM manipulation.
7. It has a namespace concept by defining a “Module”.
8. Superset of JavaScript
9. ES6 features support

Setup and Install Typescript NPM and Angular 2

Two main ways to Installing the TypeScript,

1. Installing using npm
2. Installing TypeScript’s Visual Studio plugins

In the Visual Studio 2017, TypeScript include by default. If you don’t have TypeScript with Visual Studio, try for NPM users,

```
> npm install -g typescript
```

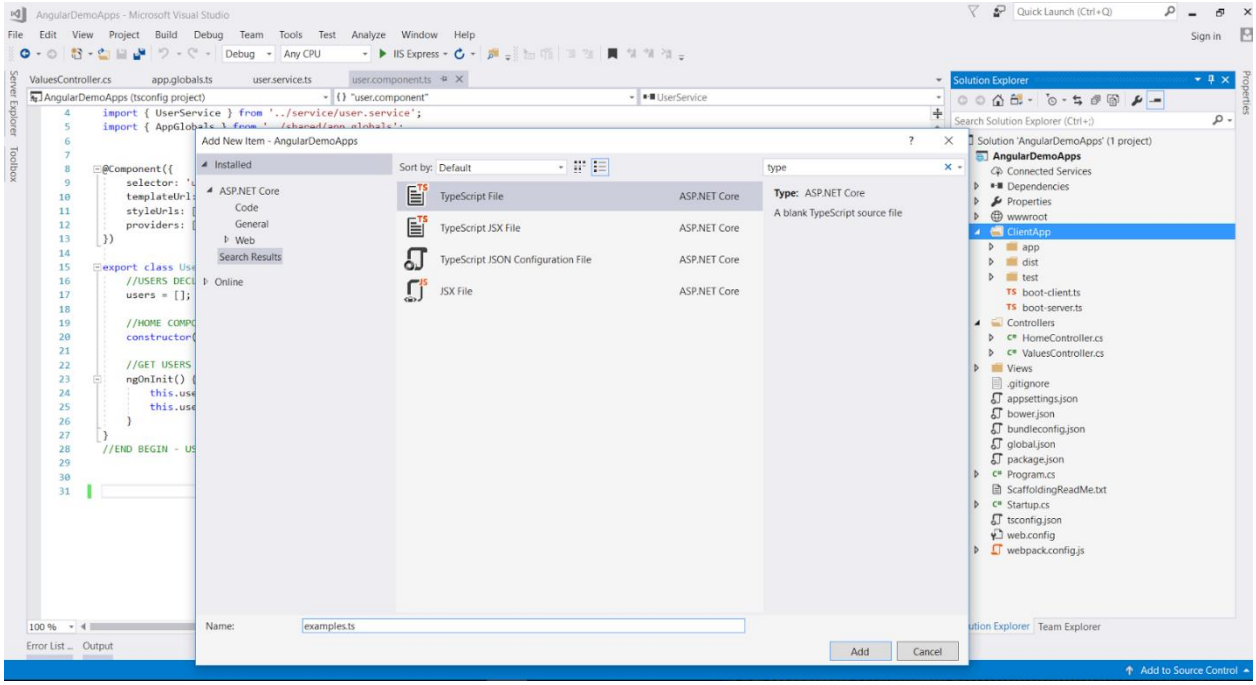
Example-1

```
class Users {
   userName: string;
   constructor (name: string) {
      this.userName = name;
   }
   getUserName() {
      return "Hello, " + this.userName;
   }
}
```

Example -2

```
class Users {
   private firstName: string;
   private lastName: string;

   //Constructor
   constructor(firstName: string, lastName: string) {
      this.firstName = firstName;
      this.lastName = lastName;
   }
```

```
    //Function
    studentFullName(): void {
        alert(this.firstName + ' ' + this.lastName);
    }
}
```



103.What is an Interface in TypeScript?

An interface in TypeScript is similar to other object oriented programming languages interfaces.
An interface is a way to define a contract on a function with respect to the arguments.

In the below example, I am using an interface that describes objects that have a “name”, “age” and “address” fields and the following code defines an interface and a function that takes a parameter that adheres to that interface.

```
//USER INTERFACE
interface User {
    name: string;
    age: number;
```

```
    address: string
}
```

```
//FUNCTION USING USER INTERFACE
let userInfo = function(user: User) {
  let info = "Hello, " + user.name + " Your Age is - " + user.age + " and Address is -" +
user.address;

  return info;
}
```

```
//USER INFO JSON OBJECT
let info = {
    name: "Anil",
    age: 30,
    address: "Noida, India."
};

//RESULT
console.log(userInfo(info));
```

104.What is Functions in TypeScript? How many types you defined in TypeScript?

A function is a set of statements that perform a specific task and used to create readable, maintainable and re-usable code.

A function declaration tells to compiler about a function name, return type, and parameters.

TypeScript functions are almost similar to JavaScript functions but there are different ways of writing functions in TypeScript.

Different types of functions are available in the TypeScript i.e.

1. Normal function
2. Anonymous Function
3. Named Function
4. Lambda Function/Arrow Function
5. Class Function
6. Optional Parameters
7. Rest Parameters
8. Default Parameters

Anonymous Functions–

An anonymous function is a function that was declared without any named identifier to refer to it.

Example - Normal function

```
function printHello() {
   console.log('Hello Anil!');
}

printHello();
```

Examples - Anonymous function

JavaScript -

```
var hello = function () {
   console.log('Hello Anil!, I am Anonymous.');
};

hello();//Return - Hello Anil!, I am Anonymous.

OR

setTimeout(function () {
   console.log('Hello Anil!, I am Anonymous.');
}, 2000); //Return - Hello Anil!, I am Anonymous.
```

TypeScript–

```
var anonymousFunc = function (num1: number, num2: number): number {
   return num1 + num2;
}

//RESULT
console.log(anonymousFunc(10, 20)); //Return is 30

//RESULT
console.log(anonymousFunc(10, "xyz"));
// error: Argument of type 'number' is not assignable to parameter of type 'string'.
//because return type is number for anonymous function).
```

Named Function -

The named function is very similar to the JavaScript function and only one difference - we must declare the type on the passed parameters.

Example – JavaScript

```
function addTwoNumer(num1, num2) {
   return num1 + num2;
}
```

Example – TypeScript

```
function addTwoNumer(num1: number, num2: number): number {
   return num1 + num2;
}
```

Lambda Function/Arrow Function -

The arrow function is additional feature in typescript and it is also known as a lambda function.
A lambda function is a function without a name.

```
var addNum = (n1: number, n2: number) => n1 + n2;
```

In the above, the "=>" is a lambda operator and (n1 + n2) is the body of the function and (n1: number, n2: number) are inline parameters.

For example –

```
let addNum = (n1: number, n2: number): number => { return n1 + n2; }
let multiNum = (n1: number, n2: number): number => { return n1 * n2; }
let dividNum = (n1: number, n2: number): number => { return n1 / n2; }

addNum(10, 2);// Result - 12
multiNum(10, 2);// Result - 20
multiNum(10, 2);// Result - 5
```

Optional Parameters Function -

We can specify optional properties on interfaces and the property may be present or missing in the object.

In the below example, the address property is optional on the following “User” interface.

For Example as,

```
//USER INTERFACE
interface User {
   name: string;
   age: number;
   address?: string //Optional
}

//FUNCTION USING USER INTERFACE
let userInfo = function(user: User) {
  let info = "Hello, " + user.name + " Your Age is - " + user.age + " and Address is -" +
user.address;

  return info;
}

//USER INFO JSON OBJECT
let info = {
   name: "Anil",
   age: 30
};

//RESULT
console.log(userInfo(info));
```

Rest Parameters –

The Rest parameters do not restrict the number of values that we can pass to a function and the passed values must be the same type otherwise throw the error.

For Example as,

```
//Rest Parameters
let addNumbers = function(...nums: number[]) {
   let p;
   let sum: number = 0;

   for (p = 0; p < nums.length; p++) {
      sum = sum + nums[p];
   }

   return sum;
```

```
}

//The Result
addNumbers(1, 2);
addNumbers(1, 2, 3);
addNumbers(1, 12, 10, 18, 17);
```

Default Parameters -

Function parameters can also be assigned values by default.
A parameter can't be declared as optional and default both at the same time.

For Example as,

```
let discount = function (price: number, rate: number = 0.40) {
   return price * rate;
}

//CALCULATE DISCOUNT
discount(500); // Result - 200

//CALCULATE DISCOUNT
discount(500, 0.45); // Result - 225
```

105.Why type definition with TypeScript in Angular 2?

A TypeScript definition file contains the type information for written in JavaScript code and the “JavaScript” does not contain type information itself, so “TypeScript” cannot retrieve that information. To solve this problem, we will use the type definition with TypeScript.

106.Can I use Angular 2 with Typings in Angular 2 ?

Yes!, “TSD” and “Typings” can play together but it is not recommended and both output in the same directory but it will not conflict.

No!, you do not need “Typings” or “TSD” to use TypeScript. It is only needed for the typings of libraries not in TypeScript and Angular 2 already has typings in the “node_modules”.

107.Are Typescript type definitions Required?

No!, If the module does it properly using the typings property it will work nicely.

108.Do we only need type definition files not "node_modules"?

A Type definition files provide better experience when using "JS libraries" with auto completion and type checking but are ignored otherwise.

If we install and import TypeScript libraries and they will be compiled to "JS" and it's included in the "JS" build output.

109.What is Typing?

Typings is the simple way to manage and install TypeScript definitions. It uses "typings.json", which can resolve to the Typings Registry, NPM, HTTP and local files.

Custom Type Definitions:-

In Angular 2, when we including 3rd party modules. We also need to include the type definition for this module and if they do not provide one within the modules.

If you want to try to install it with typings,

typings install node –save

If we cannot find the type definition in the registry, we can make an extensive definition in this file for now.

For example as,

```
declare module "myModule" {
  export function toDo(value: string): string;
}
```

Angular 2 Dynamic vs. Static Typing :-

Angular 2 applications can be written both in Typescript and in plain old JavaScript. This means we have an important choice to make: whether to go with static typing or choose the dynamically typed path.

111.What's the difference between statically and dynamically typed languages?

Static Typing:-

Static typing requires that all variables and function return values be typed and TypeScript is an examples of statically typed languages.

For example as,

```
public sumOfTwoNumbers(num1: number, num2: number): number{
   return num1 + num2;
}
```

This "sumOfTwoNumbers()" function takes two numeric (or int) values as arguments and returns a number.

Dynamically Typing :-

Dynamically typing does not require explicit type declaration. Python and JavaScript are best examples of dynamically typed languages.

The above example looks like as,

```
public sumOfTwoNumbers(num1, num2){
   return num1 + num2;
}
```

After calling this method "sumOfTwoNumbers(12, "34");" The output is 1234. There is "No error", "No warning" and nothing else.