Angular & TypeScript Interview Questions:

**1.Difference between AngularJS and Angular2**

Angular JS 1.X is the most popular JavaScript-based framework to build web applications. It was released by Google on October 20, 2010.

The architecture of AngularJS is based on model-view-controller (MVC) design. The model is the central component that expresses the application's behavior and manages its data, logic, and rules. The view generates an output based on the information in the model. The controller accepts input, converts it into commands and sends the commands to the model and the view.

ngular 2 is not an upgraded version of Angular JS 1. It is completely rewritten with a lot of improvements also by Google and released in September 2016. It has been designed for implementing big and complicated applications in a feasible way.  
  
  
Angular 2 is currently being developed in Typescript but will be compatible with both ES5 & ES6 JavaScript standards.

Angular 2 is based on a **Components** structure, like what we see in React.js.

In Angular 2, controllers and $scope were replaced by components and directives. Components are directives with a template.

**2.What are the components in angular**

Components are like the basic building block of UI in an Angular application. Components are defined using the @component decorator. A component has a selector, template, style and other properties, using which it specifies the metadata required to process the component.

Using components we can break application into smaller parts.

**6.Can we create multiple instances of services**

Don't add the service to the providers of the component.

@Component ({

selector : 'Sub-Component',

template : `<hr>

This is the Sub-Component !

<BR>

StateService Log :

<pre>{{ \_stateService.log }}</pre>

<button (click)="WriteToLog ()">Write to log</button>

`,

// providers : [StateService] <== remove

})

Instead add it to app.module.ts

@NgModule({ providers: [...], ...

If you add it on a component you get a new service instance for each component instance.

**7.Can we call components as a type of directive**

**Yes**

**But there is difference when coming to meta-data**

A component, rather than adding/modifying behaviour, actually creates its own view (hierarchy of DOM elements) with attached behaviour.

There are three kinds of directives in Angular:

1. Components—directives with a template.
2. Structural directives—change the DOM layout by adding and removing DOM elements. e.g \*ngIf
3. Attribute directives—change the appearance or behavior of an element, component, or another directive. e.g [ngClass].

**Components**

1. To register a component we use @Component meta-data annotation.
2. Component is a directive which uses shadow DOM to create encapsulated visual behavior called components. Components are typically used to create UI widgets.
3. Component is used to break up the application into smaller components.
4. Only one component can be present per DOM element.
5. @View decorator or templateurl template are mandatory in the component.

**Directive**

1. To register directives we use @Directive meta-data annotation.
2. Directive is used to add behavior to an existing DOM element.
3. Directive is use to design re-usable components.
4. Many directives can be used per DOM element.
5. Directive doesn't use View.

**9.What do we prefer angular over javascript.**

**1.Databingin(two-way)**

**2.services(DI)**

**3.Directive**

**4.Templating(**Angular uses plain old HTML as the templating language.**)**

**3.Routing(Making SPA)**

**10.Wha is the single page application  
Single-Page Application**

A single-page application is an app that works inside a browser and does not require page reloading during use. You are using this type of applications every day. These are, for instance: Gmail, Google Maps, Facebook or GitHub.  
 SPAs are all about serving an outstanding UX by trying to imitate a “natural” environment in the browser — no page reloads, no extra wait time. It is just one web page that you visit which then loads all other content using JavaScript — which they heavily depend on.

SPA requests the markup and data independently and renders pages straight in the browser. We can do this thanks to the advanced JavaScript frameworks like AngularJS, Ember.js, Meteor.js, Knockout.js .  
 Single-page sites help keep the user in one, comfortable web space where content is presented to the user in a simple, easy and workable fashion.

**Pros of the Single-Page Application:**

* SPA is fast, as most resources (HTML+CSS+Scripts) are only loaded once throughout the lifespan of application. Only data is transmitted back and forth.
* The development is simplified and streamlined. There is no need to write code to render pages on the server. It is much easier to get started because you can usually kick off development from a file file://URI, without using any server at all.
* SPAs are easy to debug with Chrome, as you can monitor network operations, investigate page elements and data associated with it.
* It’s easier to make a mobile application because the developer can reuse the same backend code for web application and native mobile application.
* SPA can cache any local storage effectively. An application sends only one request, store all data, then it can use this data and works even offline.

13. How to decrease load time of page in angular

+100

A single page application generally takes more time while loading as it loads all necessary things at once.

I had also faced same problem and my team has optimized our project from loading in 8 seconds to 2 seconds by using following methods.

1. **Lazy loading a module :** Lazy loading modules helps to decrease the startup time. With lazy loading our application does not need to load everything at once, it only needs to load what the user expects to see when the app first loads. Modules that are lazily loaded will only be loaded when the user navigates to their routes. Angular2 has introduced modules in its final release RC5. **See below for step-by-step guide.**
2. **Aot Compilation :** With AoT, the browser downloads a pre-compiled version of the application. The browser loads executable code so it can render the application immediately, without waiting to compile the app first.

*It reduces the payload size :* There's no need to download the Angular compiler if the app is already compiled. The compiler is roughly half of Angular itself, so omitting it dramatically reduces the application payload. For more info see [this](https://angular.io/docs/ts/latest/cookbook/aot-compiler.html).

1. **Webpack :** Webpack is a popular module bundler, a tool for bundling application source code in convenient chunks and for loading that code from a server into a browser. You can configure your Angular 2 web application with webpack (see [this guide](https://angular.io/docs/ts/latest/guide/webpack.html)).
2. **Remove scripts,stylesheet from index.html :** Remove all scripts and stylesheet which are not needed in index.html. You can load these script dynamically in component itself by calling a service.

Make a file script.service.ts which can load any script on demand for that component

\script.service.ts

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

declare var document: any;

@Injectable()

export class Script {

loadScript(path: string) {

//load script

return new Promise((resolve, reject) => {

let script = document.createElement('script');

script.type = 'text/javascript';

script.src = path;

if (script.readyState) { //IE

script.onreadystatechange = () => {

if (script.readyState === "loaded" || script.readyState === "complete") {

script.onreadystatechange = null;

resolve({ loaded: true, status: 'Loaded' });

}

};

} else { //Others

script.onload = () => {

resolve({ loaded: true, status: 'Loaded' });

};

};

script.onerror = (error: any) => resolve({ loaded: false, status: 'Loaded' });

document.getElementsByTagName('head')[0].appendChild(script);

});

}

}

This is just a sample code to load script dynamically, you can customize and optimize it by yourself according to your need. For stylesheet you should load it in component using styleUrl.

1. **Use Browser Caching :** Your webpage files will get stored in the browser cache when you use browser caching. Your pages will load much faster for repeat visitors and so will other pages that share those same resources. For more info <https://varvy.com/pagespeed/leverage-browser-caching.html>
2. **minimize the code in app.component.ts :** minimize the code present in app.component.ts which always run when the app loads or reloads.
3. **set data on app Initialization :** if you are using same api calls multiple times in your project or in components, or you are dependent upon same data in multiple component, instead of calling api multiple times what you can do is save the data as an object in service on app initialization. That service will act as a singleton throughout the project and you can access that data without calling api.

**Lazy loading of modules step by step**

1. **Modular structure :** We have to divide our App into separate modules. For example an app may have a user side and an admin side and each will have its own different components and routes, so we will separate this two sides into modules admin.module.ts and user.module.ts.
2. **Root Module :** Every Angular app has a root module class. By convention it's a class called AppModule in a file named app.module.ts , this module will import the above two module and also the AppComponent for bootstrap. You can also declare multiple components according to your need. Sample code in app.module.ts:

\app.module.ts

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

import { UserModule } from './user/user.module';

import { AdminModule } from './admin/admin.module';

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

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

@NgModule({

imports: [UserModule, AdminModule],

declarations: [AppComponent, LoginComponent],

bootstrap: [AppComponent]

})

export class AppModule { }

1. **Routes :** Now in your routes you can specify like the following

\app.router.ts

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

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

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

const routes: Routes = [

{ path: 'login', component: 'LoginComponent' }, //eager loaded

{ path: 'admin', loadChildren: './admin/admin.module#AdminModule' }, // Lazy loaded module

{ path: 'user', loadChildren: './user/user.module#UserModule' } //lazy loaded module

];

Now when the application loads, it will only load LoginComponent and AppComponent code. These modules will only be loaded when we visit /admin or /user routes. Hence it will decrease the size of payload for loading into the browser, thus resulting in fast loading.

1. **Nesting Modules :** Just like app.module every module has its own set of components and routes. As your project becomes larger, the nesting of modules inside module is the best way to optimize because we can lazily load those modules whenever we require.

**Ahead-of-Time Compilation**

In AngularJS 2, you can compile the applications in two ways: **Just-in-Time (JIT)**Compilation and **Ahead-of-Time (AOT)** Compilation. Till now we have been working in AngularJS 2, but what you may have not realized is that we have used the JIT compilation which compiles the application at run-time, so you can see the changes on the browser the moment you modify something in your code. However, the problem with JIT is that it may take longer to render the views (especially the files that contain large amounts of code!), and so in order to tackle the problem, Ahead-of-Time Compilation (AOT) was introduced.

AOT allows you to run the pre-compiled version of your application, so that the browser loads the executable code faster. Before our application renders on the browser, the compiler needs to transform the TypeScript code to JavaScript which takes longer, but this must be done by the Angular CLI because we are using TypeScript.

With AOT, you can build the application offline (in a kind of step-by-step build), and once the compilation is done, we can upload the compiled version. This may lead to performance improvements (such as faster loading times) and also reduce the size of the program.

Oh, another cool feature of the AOT is **Tree Shaking**, which does nothing but reads the dependency graph from top to bottom and “shakes” the unused code. It greatly reduces the downloaded size of the application, so for example, if you are not using code from @angular/forms then it will not download Forms-related code. Also, AOT and Tree Shaking are different processes, but AOT helps the code, making it more shakable by converting the HTML code to JavaScript so the Tree shaker can process more of our application.

In order to do the AOT compilation you can just use angular-cli it already doing it for us.

**Angular cli**

npm uninstall -g angular-cli  
npm cache clean  
npm install -g angular-cli@latest

Now, you can “ng serve” to see the results.

**Bundling for Production**

You can use the following commands to build the final application for the production level:

**ng build --prod --aot**

This will generate the files in new “dist/” folder. In those files, you can use the “index.html” for opening the website in your browser.

Also, make sure to enable production mode while doing bootstrapping:

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

AOT build should be enough but we can take care of code how we are managing modules, lazy loading and only required imports should be in our code. Explore build with explorer modules & see what unnecessary stuff we have.

11.Can we customize bootstrap classes

Yes.

**Simple CSS Overrides**

One way to customize is simply using CSS to override Bootstrap CSS. For maintainability, CSS customizations are put in a separate custom.css file, so that the bootstrap.css remains unmodified. The reference to the custom.css follows after the bootstrap.css for the overrides to work...

<link rel="stylesheet" type="text/css" href="css/bootstrap.min.css">

<link rel="stylesheet" type="text/css" href="css/custom.css">

Just add whatever changes are needed in the custom CSS. For example...

/\* remove rounding from cards, buttons and inputs \*/

.card, .btn, .form-control {

border-radius: 0;

}

17.What are the advantages of typescript over javascript? Why do we use it in angular.

**Advantages of using TypeScript over JavaScript**

* TypeScript always point out the compilation errors at the time of development only. Because of this at the run-time the chance of getting errors are very less whereas JavaScript is an interpreted language.
* TypeScript has a feature which is strongly-typed or supports static typing. That means Static typing allows for checking type correctness at compile time. This is not available in JavaScript.
* TypeScript is nothing but JavaScript and some additional features i.e. ES6 features. It may not be supported in your target browser but TypeScript compiler can compile the **.ts** files into ES3,ES4 and ES5 also.

**Difference between TypeScript and JavaScript:**

* TypesScript is known as Object oriented programming language whereas JavaScript is a scripting language.
* TypeScript has a feature known as Static typing but JavaScript does not have this feature.
* TypeScript gives support for modules whereas JavaScript does not support modules.
* TypeScript has Interface but JavaScript does not have Interface.
* TypeScript support optional parameter function but JavaScript does not support optional parameter function.

What is interface

An interface is a TypeScript artifact, it is not part of ECMAScript. An interface is a way to define a contracton a function with respect to the arguments and their type. Along with functions, an interface can also be used with a Class as well to define custom types.

An interface is an abstract type, it does not contain any code as a class does. It only defines the 'signature' or shape of an API. During transpilation, an interface will not generate any code, it is only used by Typescript for type checking during development.

One of TypeScript’s core principles is that type checking focuses on the shape that values have. This is sometimes called “duck typing” or “structural subtyping”. In TypeScript, interfaces fill the role of naming these types, and are a powerful way of defining contracts within your code as well as contracts with code outside of your project.

# Optional Properties

Not all properties of an interface may be required. Some exist under certain conditions or may not be there at all. These optional properties are popular when creating patterns like “option bags” where you pass an object to a function that only has a couple of properties filled in.

Here’s an example of this pattern:

**interface** SquareConfig {

color?: string;

width?: number;

}