

Frequently asked: Angular Interview Questions and Answers



Vigo Webs

Follow

Jul 27, 2018 · 10 min read



Q1. What is Angular 4 and how it differs from Angular 1.x?

Angular 4 is a Javascript framework built around the concept of components, and more precisely, with the Web Components standard in mind. It was rewritten from scratch by the Angular team using Typescript (although we can use it with *ES5*, *ES6*, or *Dart* as well).

Angular 4 is a big change for us compared to 1.x. Because it is a completely different framework than 1.x, and is not backward-compatible. Angular 4 is written entirely in Typescript and meets the ECMAScript 6 specification. The main differences are:

- Angular 4 is entirely component based. `Controllers` and `$scope` are no longer used. They have been replaced by components and directives.
- Angular 4 uses TypeScript. TypeScript will not be used in the browser directly. So the program code is compiled to JavaScript. This can be achieved with “Traceur”.
- The digest cycle from Angular 1.x has been replaced by another internal mechanism known as “**Change Detection**”. This feature, along with other improvements and tweaks, yields a considerable increase in performance.

- Unlike Angular 1.x where we can get most of the functionalities in angular.js file, Angular 4 follows module pattern. We need to import the functions ourself and export them when we need anywhere else.
- There are no more `factory`, `service`, `provider` in Angular 4. We need to use `class` for declaring a service.

Q2. What is component decorators in Angular 4?

The main objectives of decorators is to add some metadata to the class that will tell Angular 4 how to process a class. Or in another words, Decorators are functions that modify JavaScript classes. Angular has many decorators that attach metadata to classes so that it knows what those classes mean and how they should work.

If we consider Component in Angular 4, we will have following options to configure.

- **selector:**—define the name of the HTML element in which our component will live.

- **template** or **templateUrl**:—It can be inline string or link an external html file. It allows us to tie logic from our component directly to a view.
- **styles**:—the styles array for our specific component. We can also link external CSS by **styleUrls**.
- **directives**:—another component directives we want to use inside our components.
- **providers**:—This is the place we are passing the services that we need inside our components.

Immediately after this decorator or right to it, we need to export a class where our variables and functions reside that our component uses.

```
export class SampleComponent {  
  name: 'Angular 4';  
}
```

Q3. What is compilation in Angular 4? And what are the types of compilation in Angular 4?

An Angular application consists largely of components and their HTML templates. Before the browser can render the application, the components and templates must be converted to executable JavaScript by the *Angular compiler*.

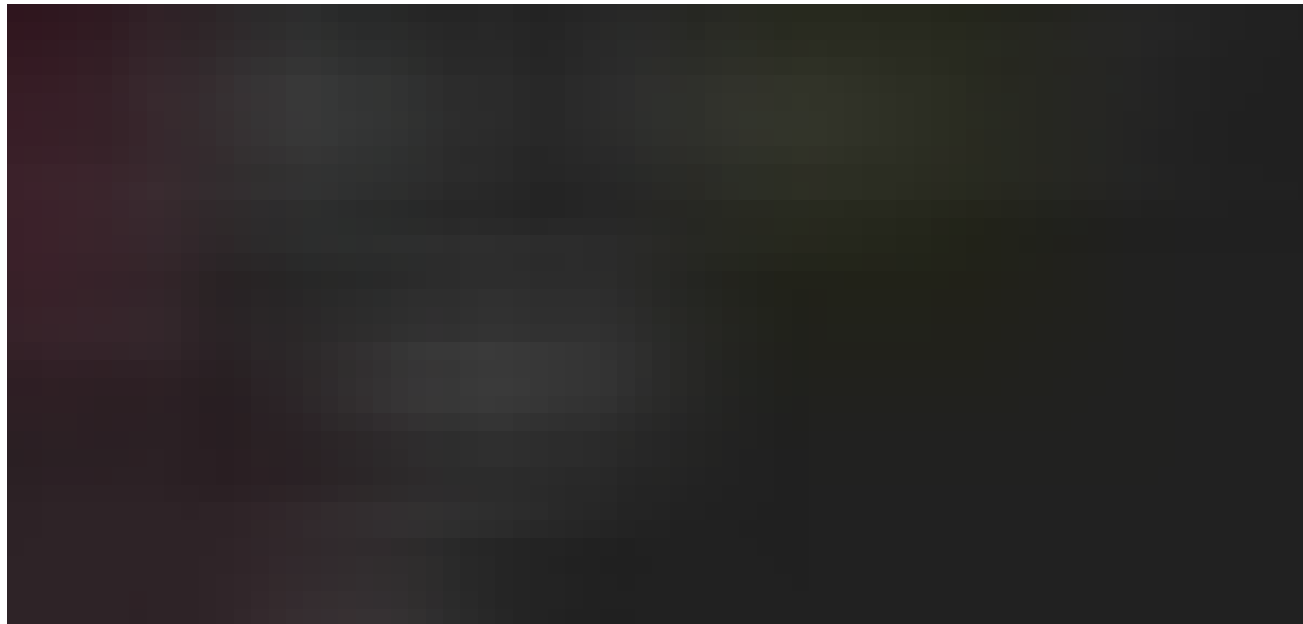
There is actually only one Angular compiler. The difference between AOT and JIT is a matter of timing and tooling. There are two types of compilation Angular 4 provides.

- **Just-in-time (JIT) compilation:** This is a standard development approach which compiles our Typescript and html files in the browser at runtime, as the application loads. It is great but has disadvantages. Views take longer to render because of the in-browser compilation step. App size increases as it contains angular compiler and other library code that won't actually need.
- **Ahead-of-time (AOT) compilation:** With AOT, the compiler runs at the build time and the browser downloads only the 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. This

compilation is better than JIT because of Fast rendering, smaller application size, security and detect template errors earlier.

Q4. What is `@NgModule` ?

An `NgModule` class describes how the application parts fit together. Every application has at least one `NgModule`, the root module that we bootstrap to launch the application.



Here the `AppComponent` is the root module of our application that Angular creates and inserts it into the `index.html` page.

Q5. What are all the *metadata* properties of `NgModule` ? And what are they used for?

`@NgModule` accepts a metadata object that tells Angular how to compile and launch the application. The properties are:

- **`imports`** – Modules that the application needs or depends on to run like, the `BrowserModule` that every application needs to run in a browser.
- **`declarations`** – the application's components, which belongs to the `NgModule` class. We must declare every component in an `NgModule` class. If we use a component without declaring it, we'll see a clear error message in the browser console.
- **`bootstrap`** – the root component that Angular creates and inserts into the `index.html` host web page. The application will be launched by creating the components listed in this array.

Q6. What is Template reference variables?

A template reference variable (`#var`) is a reference to a DOM element within a template. We use hash symbol (`#`) to declare a reference variable in a template.

```
<input #name placeholder="Your name">
{{ name.value }}
```

In the above code the `#name` declares a variable on the `input` element. Here the `name` refers to the `input` element. Now we can access any property of the `input` DOM, using this reference variable. For example, we can get the value of the `input` element as `name.value` and the value of the placeholder property by `name.placeholder` anywhere in the template.

Finally, a Template reference variable refers to its attached element, component or directive. It can be accessed anywhere in the entire template. We can also use `ref-` instead of `#`. Thus we can also write the above code as `ref-name`.

Q7. What are structural directives?

Structural directives are responsible for HTML layout. They shape or reshape the DOM's structure, typically by adding, removing, or manipulating elements. Structural directives are easy to recognize. An asterisk (*) precedes the directive attribute name as in this example.

```
<ul>
  <li *ngFor="let name of names">{{name}}</li>
</ul>
```

The `ngFor` directive iterates over the component's `names` array and renders an instance of this template for each `name` in that array.

Some of the other structural directives in Angular are `ngIf` and `ngSwitch`.

Q8. What is Directive in Angular 4? How it differs from Components?

Directives allow us to attach behavior to elements in the DOM, for example, doing something on mouse over or click. In Angular, a Directive decorator (`@Directive`) is used to mark a class as an Angular directive and provides

additional metadata that determines how the directive should be processed. Below are the metadata properties of a directive.

- `selector` - css selector that identifies this component in a template
- `host` - map of class property to host element bindings for events, properties and attributes
- `inputs` - list of class property names to data-bind as component inputs
- `outputs` - list of class property names that expose output events that others can subscribe to
- `providers` - list of providers available to this component and its children
- `queries` - configure queries that can be injected into the component
- `exportAs` - name under which the component instance is exported in a template

A Component is a directive with a template. So we should use a Component whenever we want reusable set of DOM elements with behaviors of UI. And we should use a Directive whenever we want reusable behavior to supplement the DOM.

Q9. What are all the types of Directives?

There are three types of directives in Angular. They are **attribute directives**, **structural directives**, and **components**.

- **Structural directives** change the DOM layout by adding and removing DOM elements. For example, `*ngIf` and `*ngFor`
- **Attribute directives** change the appearance or behavior of an element. . For example, `*ngStyle` and `*ngClass`
- **Components** are basically directives with a template.

Q10. What are all the uses of a service?

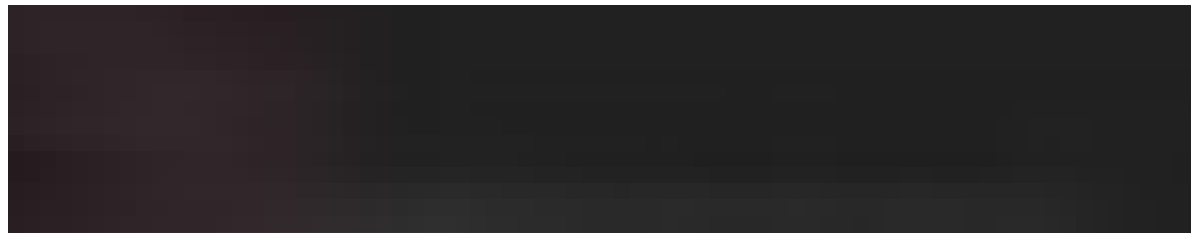
Services encapsulates business logic and separates them from UI concerns or the controller concerns, which governs them both.

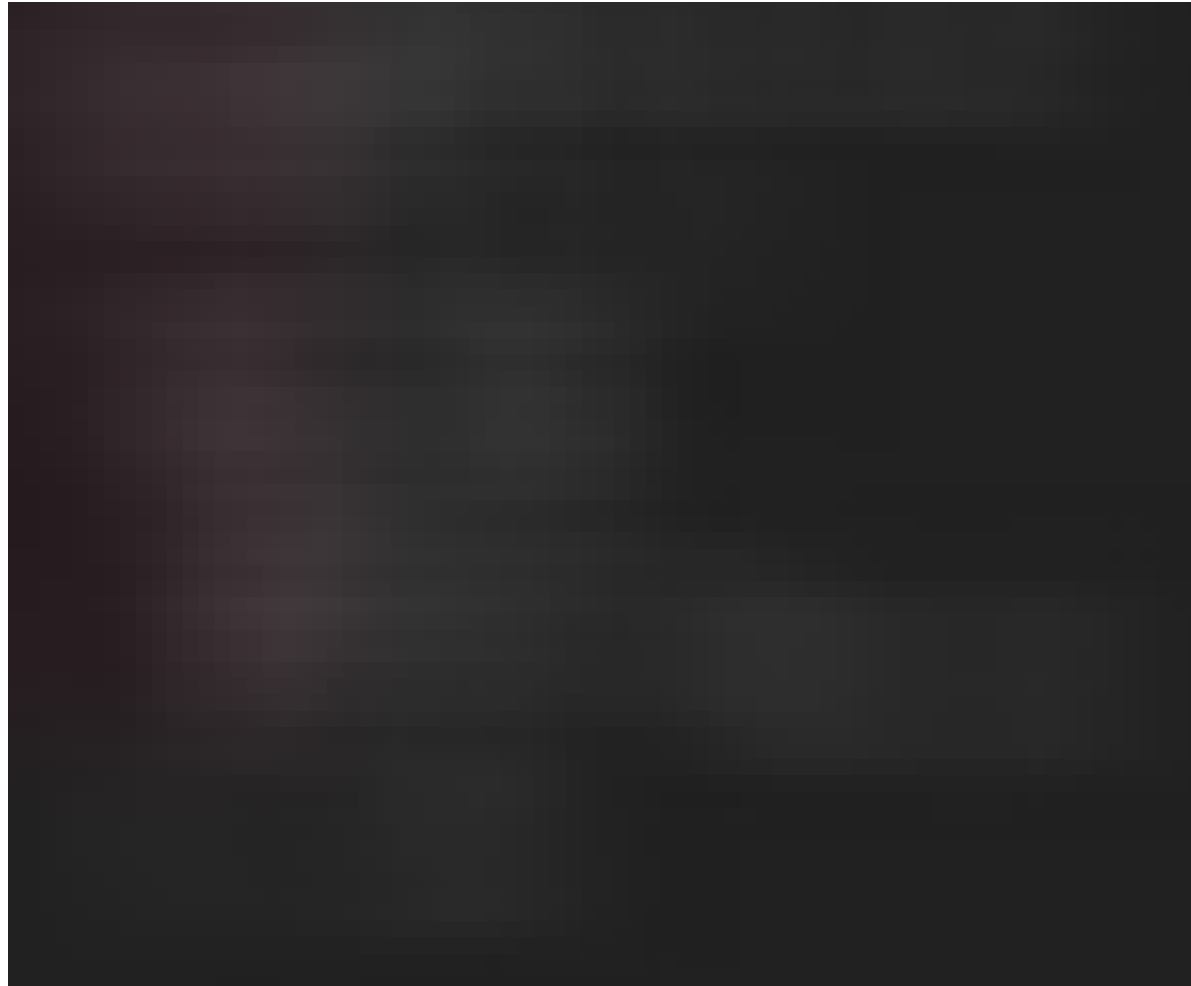
Services focus on functionality thus benefits in maintainability. The separation of UI logic from business logic is intended to reduce the coupling between the UI layer and the Model layer, leading to a cleaner design that is easier to develop, test, and maintain.

Q11. What is Pure and Impure Pipes?

`Pure pipes` are stateless that flow input data without remembering anything or causing detectable side-effects. Pipes are pure by default, hence most pipes are pure. We can make a pipe impure by setting its pure flag to `false`. Angular executes a pure pipe only when it detects a *pure* change to the input value. A pure change is either a change to a primitive input value or a changed object reference.

`Impure pipes` are those which can manage the state of the data they transform. A pipe that creates an HTTP request, stores the response and displays the output, is an impure or stateful pipe. Stateful Pipes should be used cautiously. Angular provides `AsyncPipe`, which is stateful. In the following code, the pipe only calls the server when the request URL changes and it caches the server response. The code uses the Angular http client to retrieve data:





Q12. What is Redux and `@ngrx` ?

Redux is an application state manager for JavaScript applications, and keeps with the core principles of the Flux-architecture by having a unidirectional

data flow in our application. Redux applications have only one global, read-only application state. This state is calculated by “reducing” over a collection or stream of actions that update it in controlled ways.

`@ngrx` is a set of modules that implement the same way of managing state as well as some of the middleware and tools in the Redux ecosystem. In other way, `ngrx` is a collection of reactive libraries for angular, containing a redux implementation and many other useful libraries.

Using this technique, we keep our application state in Store and everything saved in the store is read only. The only way to change the state is to emit an action, an object describing what happened.

Q13. How to prevent security threads in Angular App? What are all the ways we could secure our App?

Some of them are:

- Avoid using/injecting dynamic HTML content to your component.

- If using external HTML which is coming from database or somewhere outside the application, sanitize it before using.
- Try not to put external urls in the application unless it is trusted. Avoid url re-direction unless it is trusted.
- Consider using AOT compilation or offline compilation.
- Try to prevent XSRF attack by restricting the api and use of the app for known or secure environment/browsers.

Q14. How to optimize Angular app?

- Consider lazy loading instead of fully bundled app if the app size is more.
- Make sure that any 3rd party library, which is not used, is removed from the application.
- Have all dependencies and dev-dependencies are clearly separated.
- Make sure the application doesn't have un-necessary import statements.
- Make sure the application is bundled, uglified, and tree shaking is done.
- Consider AOT compilation.

Q15. What is `NgZone` service? How Angular is notified about the changes?

`Zone.js` is one of the Angular dependencies which provides a mechanism, called zones, for encapsulating and intercepting asynchronous activities in the browser (e.g. `setTimeout`, `setInterval`, `promises`). These zones are *execution contexts* that allow Angular to track the start and completion of asynchronous activities and perform tasks as required (e.g. change detection). `Zone.js` provides a global zone that can be forked and extended to further encapsulate/isolate asynchronous behaviour, which Angular does so in its `NgZone` service, by creating a fork and extending it with its own behaviours.

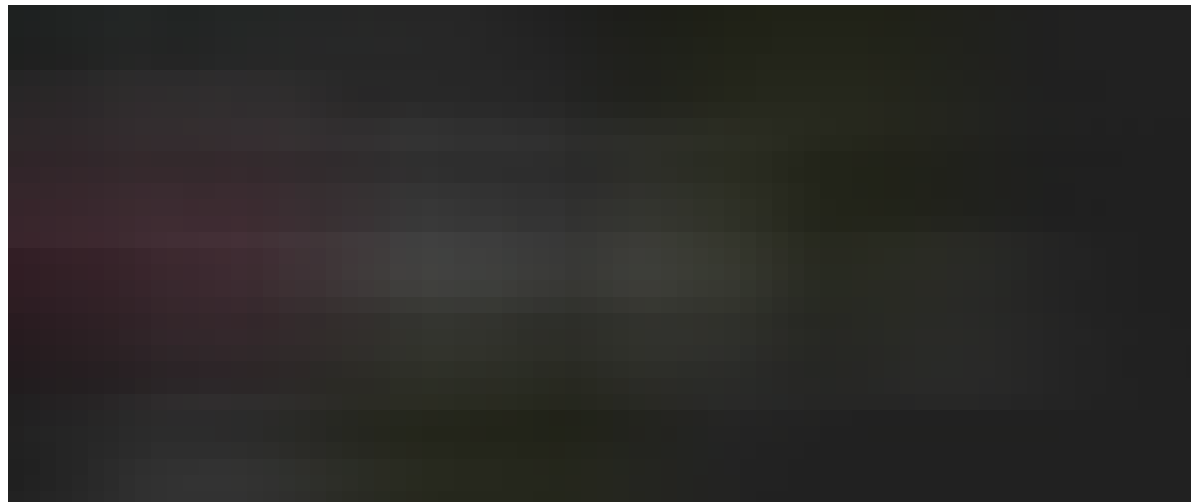
The `NgZone` service provides us with a number of Observables and methods for determining the state of Angular's zone and to execute code in different ways inside and outside Angular's zone.

`NgZone` exposes a set of Observables that allow us to determine the current status, or stability, of Angular's zone.

- `onUnstable` – Notifies when code has entered and is executing within the Angular zone.

- `onMicrotaskEmpty` - Notifies when no more microtasks are queued for execution. Angular subscribes to this internally to signal that it should run change detection.
- `onStable` – Notifies when the last `onMicroTaskEmpty` has run, implying that all tasks have completed and change detection has occurred.
- `onError` – Notifies when an error has occurred. Angular subscribes to this internally to send uncaught errors to its own error handler, i.e. the errors you see in your console prefixed with 'EXCEPTION:'.

We can inject the `NgZone` service in our component/services/etc. and can subscribe to these observables.





Subscribing to these can help you determine if your code is unexpectedly triggering change detection as a result of operations that do not affect application state.

Q16. What is Traceur compiler?

Traceur compiler is a Google project. It compiles ECMAScript Edition 6 (ES6) (including classes, generators and so on) code on the fly to regular Javascript (ECMAScript Edition 5 [ES5]) to make it compatible for the browser.

Traceur itself is written in ES6, compiled to ES5.

For more Angular Interview Questions and Answers visit my blog:

<http://blog.vigoweb.com/post/2018/angular5-interview-questions/>

[JavaScript](#)[Angular 4](#)[Angular 5](#)[Rxjs](#)[Angular2](#)

229 claps



2



Vigo Webs

Follow



Related reads

Version 6 of Angular Now Available



Stephen Fluin

May 3, 2018 · 7 min re



31K



Related reads

The Three Pillars of Angular Routing. Angular Router Series...



Nate Lapinski

Sep 4, 2018 · 8 min re



2.2K



Related reads

What's new in Angular 7.0 and how you can upgrade



Ankit Sharma


Nov 2, 2018 · 6 min re



1.1K



Responses

 Write a response...

Show all responses