**Angular 2**

**Important question links:**

* [https://www.onlineinterviewquestions.com/angular2-interview-questions/#.W3wHQVUzbZ4](https://www.onlineinterviewquestions.com/angular2-interview-questions/)
* <https://iq.js.org/questions/angular/what-is-the-benefit-of-automatic-inlining-of-fonts>
* <https://www.greycampus.com/blog/programming/top-30-interview-questions-and-answers-on-angular-5>
* <https://www.code-sample.com/2017/04/angular-4-interview-questions-and.html>
* <https://dzone.com/articles/angular-6-release-vs-angular-5-new-features-and-im>
* <https://www.peerbits.com/blog/angular-7-features-and-updates.html>
* <https://dzone.com/articles/top-10-features-of-angular-60>

Angular is a platform and framework that makes it easy to build single page web application.

* A component-based framework for building scalable web applications
* A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more

The basic building blocks of the Angular framework are Angular components that are organized into NgModules. Every Angular app has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application.

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

**Modules**

<https://angular.io/guide/architecture-modules>

Angular NgModules differ from and complement JavaScript (ES2015) modules.

Every Angular application has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. An application typically contains many functional modules.

Why several NgModules needed?

Organizing your code into distinct functional modules helps in managing development of complex applications, and in designing for reusability. In addition, this technique lets you take advantage of lazy-loading—that is, loading modules on demand—to minimize the amount of code that needs to be loaded at startup.

**Components**

Components are the most basic building block of a UI in Angular applications and it controls views (HTML/CSS). Technically components are basically TypeScript classes that interact with the HTML files of the components, which get displayed on the browsers.

Every Angular application has at least one component, the root component that connects a component hierarchy with the page document object model (DOM). Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.

The @Component() decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

Ex.

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

@Component({

selector: 'my-app',

template: `<h1>Hello {{name}}</h1>`

})

export class AppComponent { name = 'Angular'; }

The selector property tells Angular to display the component inside a custom <my-app> tag in the index.html

The template property defines html web pagedirect.

Files outside src/ concern building, deploying, and testing your app. They include configuration files and external dependencies.

Files inside src/ "belong" to your app. Add new Typescript, HTML and CSS files inside the src/ directory.

**Templates, directives, and data binding:**

A template combines HTML with Angular markup that can modify HTML elements before they are displayed. Template directives provide program logic, and binding markup connects your application data and the DOM. There are two types of data binding:

* Event binding lets your app respond to user input in the target environment by updating your application data.
* Property binding lets you interpolate values that are computed from your application data into the HTML.

Angular supports two-way data binding,

<https://angular.io/guide/architecture-components#data-binding>

**Directives:**

When Angular renders view, it transforms the DOM according to the instructions given by directives. A directive is a class with a @Directive() decorator.

**Structural directives**

Structural directives alter layout by adding, removing, and replacing elements in the DOM.

Ex. NgIf, NgFor

**Attribute directives**

Attribute directives alter the appearance or behavior of an existing element. In templates they look like regular HTML attributes, hence the name.

The ngModel directive, which implements two-way data binding, is an example of an attribute directive. ngModel modifies the behavior of an existing element (typically <input>) by setting its display value property and responding to change events.

For example: ngStyle and ngClass

**Services:**

For data or logic that isn't associated with a specific view, and that you want to share across components, you create a service class. A service class definition is immediately preceded by the @Injectable() decorator. The decorator provides the metadata that allows other providers to be injected as dependencies into your class.

**Dependency Injection:**

Dependency injection, or DI, is a design pattern in which a class requests dependencies from external sources rather than creating them. Angular's DI framework provides dependencies to a class upon instantiation.

**Basically, DI is a pattern which we use to resolve dependencies in different components.**

DI is wired into the Angular framework and used everywhere to provide new components with the services or other things they need.

* The injector is the main mechanism. Angular creates an application-wide injector for you during the bootstrap process, and additional injectors as needed. You don't have to create injectors.
* An injector creates dependencies, and maintains a container of dependency instances that it reuses if possible.
* A provider is an object that tells an injector how to obtain or create a dependency.

For any dependency that you need in your app, you must register a provider with the application's injector, so that the injector can use the provider to create new instances. For a service, the provider is typically the service class itself.

In software engineering, dependency injection is **a technique in which an object receives other objects that it depends on**, called dependencies. ... The intent behind dependency injection is to achieve separation of concerns of construction and use of objects. This can increase readability and code reuse.

A dependency doesn't have to be a service—it could be a function, for example, or a value.

When Angular discovers that a component depends on a service, it first checks if the injector has any existing instances of that service. If a requested service instance doesn't yet exist, the injector makes one using the registered provider, and adds it to the injector before returning the service to Angular.

<https://angular.io/guide/architecture-services#providing-services>

**Routing:**

The Angular Router NgModule provides a service that lets you define a navigation path among the different application states and view hierarchies in your app.

The router maps URL-like paths to views instead of pages. When a user performs an action, such as clicking a link, the router intercepts the browser's behavior, and shows or hides view hierarchies.

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

This guide works with a CLI-generated Angular application. If you are working manually, make sure that you have <base href="/"> in the <head> of your index.html file. This assumes that the app folder is the application root, and uses "/".

You must add a <base href> element to the application's index.html for pushState routing to work. The browser uses the <base href> value to prefix relative URLs when referencing CSS files, scripts, and images.

The order of routes is important because the Router uses a first-match wins strategy when matching routes, so more specific routes should be placed above less specific routes.

<https://angular.io/guide/router#getting-route-information>

The two asterisks, \*\*, indicate to Angular that this routes definition is a wildcard route

const routes: Routes = [

{ path: 'first-component', component: FirstComponent },

{ path: 'second-component', component: SecondComponent },

{ path: '', redirectTo: '/first-component', pathMatch: 'full' }, // redirect to `first-component`

{ path: '\*\*', component: PageNotFoundComponent }, // Wildcard route for a 404 page

];

<https://angular.io/guide/router#nesting-routes>

<https://angular.io/api/router/Route#componentless-routes>

<https://codeburst.io/understanding-angular-guards-347b452e1892>

**Lazy-loading:**

Lazy loading speeds up application load time by splitting the application into multiple bundles and loading them on demand. To use lazy loading, provide the loadChildren property in the [Route](https://angular.io/api/router/Route) object, instead of the children property.

Given the following example route, the router will lazy load the associated module on demand using the browser native import system.

content\_copy[{

path: 'lazy',

loadChildren: () => import('./lazy-route/lazy.module').then(mod => mod.LazyModule),

}];

By default, NgModules are eagerly loaded, which means that as soon as the application loads, so do all the NgModules, whether or not they are immediately necessary. For large applications with lots of routes, consider lazy loading—a design pattern that loads NgModules as needed. Lazy loading helps keep initial bundle sizes smaller, which in turn helps decrease load times.

<https://angular.io/guide/lazy-loading-ngmodules>

**Note:** You can specify the path-match strategy 'full' to make sure that the path covers the whole unconsumed URL. It is important to do this when redirecting empty-path routes. Otherwise, because an empty path is a prefix of any URL, the router would apply the redirect even when navigating to the redirect destination, creating an endless loop.

We should use path match as full when using empty path or param

**Pipes**

Angular pipes let you declare display-value transformations in your template HTML. A class with the @Pipe decorator defines a function that transforms input values to output values for display in a view.

See real time example from: <https://dotnettutorials.net/lesson/angular-pipes/>

**Directives:**

Angular templates are dynamic. When Angular renders them, it transforms the DOM according to the instructions given by directives. A directive is a class with a @Directive() decorator.

A component is technically a directive. Angular defines the @Component() decorator, which extends the @Directive() decorator with template-oriented features.

In addition to components, there are two other kinds of directives: structural and attribute. You can define your own using the @Directive() decorator.

**Content projection:**

Purpose: to create flexible, reusable components.

Content projection is a pattern in which you insert, or project, the content you want to use inside another component.

In the template for your component, add an ng-content element where you want the projected content to appear. The ng-content element is a placeholder that does not create a real DOM element. Custom attributes applied to ng-content are ignored.

<https://angular.io/guide/content-projection>

* [https://vibhas1892.medium.com/difference-betw een-ng-template-ng-container-and-ng-content-a1d264619655](https://vibhas1892.medium.com/difference-betw%20een-ng-template-ng-container-and-ng-content-a1d264619655)

<https://codeburst.io/angular-interview-question-what-are-ng-container-ng-content-and-ng-template-9fafbbc255d5>

Angular will not initialize the content of an ng-template element until that element is explicitly rendered.

**Difference between library and framework.**

**Library** (<https://www.quora.com/What-is-a-framework-in-programming>)**:** Libraries are a bunch of codes that has already implementing functionalities. You can call the appropriate library for the appropriate job and the library does the job for you. You don’t need to know how the functions inside the libraries do their job. You just need to know how to call them.

**FrameWork:** It is same as library but on top of the implemented functionalities we can add our own code.

**Just-in-time (JIT) compilation**

A bootstrapping method of compiling components and modules in the browser and launching the application dynamically. Just-in-time mode is a good choice during development.

**app/app.component.ts**

Defines the same AppComponent as the one in the QuickStart playground. It is the root component of what will become a tree of nested components as the application evolves.

**app/app.module.ts**

Defines AppModule, the root module that tells Angular how to assemble the application. Right now it declares only the AppComponent. Soon there will be more components to declare.

**main.ts**

<https://dev.to/casperns/how-angular-trigger-indexhtml-and-start-working-1l46>

<https://medium.com/siam-vit/how-an-angular-app-work-behind-the-scenes-angular-flow-dcc4d1df27bd>

**Observables**

An Observable is basically a function that can return a stream of values to an observer over time, this can either be synchronously or asynchronously. The data values returned can go from zero to an infinite range of values.

For Observables to work there needs to be observers and subscriptions. The observer executes some instructions when there is a new value or a change in data values.

The observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of state changes. This pattern is similar (but not identical) to the publish/subscribe design pattern.

Each Http service method returns an Observable of HTTP Response objects.

Background

An Observable is a stream of events that you can process with array-like operators.

Angular core has basic support for observables. Developers augment that support with operators and extensions from the RxJS library. You'll see how shortly.

Recall that the HeroService chained the toPromise operator to the Observable result of http.get(). That operator converted the Observable into a Promise and you passed that promise back to the caller.

Converting to a Promise is often a good choice. You typically ask http.get() to fetch a single chunk of data. When you receive the data, you're done. The calling component can easily consume a single result in the form of a Promise.

But requests aren't always done only once. You may start one request, cancel it, and make a different request before the server has responded to the first request.

A request-cancel-new-request sequence is difficult to implement with Promises, but easy with Observables.

<https://hassantariqblog.wordpress.com/2016/12/03/angular2-http-delete-using-observable-in-angular-2-application/>

When importing from the @angular/http module, SystemJS knows how to load services from the Angular HTTP library because the systemjs.config.js file maps to that module name. The HttpModule is necessary for making HTTP calls.

Observable

Think of an Observable as a stream of events published by some source. To listen for events in this stream, subscribe to the Observable. These subscriptions specify the actions to take when the web request produces a success event or a fail event (with the error in the payload).

The observable’s map callback moves to the success parameter and its catch callback to the fail parameter in this pattern.

The errorHandler forwards an error message as a failed promise instead of a failed observable.

* **Angular Change Detection :**

<https://medium.com/technofunnel/angular-change-detection-strategy-onpush-and-default-strategy-edd8d41ba9ef> -> preffered

“<https://blog.thoughtram.io/angular/2016/02/22/angular-2-change-detection-explained.html>”

Some time later when the DOM has already been rendered. How do we figure out what has changed in our model, and where do we need to update the DOM? Accessing the DOM tree is always expensive, so not only do we need to find out where updates are needed, but we also want to keep that access as tiny as possible.

This can be tackled in many different ways. One way, for instance, is simply making a http request and re-rendering the whole page. Another approach is the concept of diffing the DOM of the new state with the previous state and only render the difference, which is what ReactJS is doing with Virtual DOM.

Angular Change Detection is responsible for making the component dynamic. During Change Detection Cycle, Angular looks for all the bindings, re-executes all the expression, compares it will the previous values and if the change is detected, it propagates the change to the DOM Elements.

Basically application state change can be caused by three things:

* There are updates to Angular State variable
* Events are invoked inside Angular Component
* @Input Values are updated for the Components

**Who notifies Angular?**

Zones take care of this. In fact, Angular comes with its own zone called NgZone.

The short version is, that somewhere in Angular’s source code, there’s this thing called ApplicationRef, which listens to NgZones onTurnDone event. Whenever this event is fired, it executes a tick() function which essentially performs change detection.

With NgZone, we can subscribe to, as they are observable streams:

* onTurnStart() - Notifies subscribers just before Angular’s event turn starts. Emits an event once per browser task that is handled by Angular.
* onTurnDone() - Notifies subscribers immediately after Angular’s zone is done processing the current turn and any micro tasks scheduled from that turn.
* onEventDone() - Notifies subscribers immediately after the final onTurnDone() callback before ending VM event. Useful for testing to validate application state.

In Angular, each component has its own change detector.

Reducing the number of checks:

Angular can skip entire change detection subtrees when input properties don’t change. If we use immutable objects in our Angular app, all we need to do is tell Angular that a component can skip change detection, if its input hasn’t changed.

We can tell Angular to skip change detection for this component’s subtree if none of its inputs changed by setting the change detection strategy to OnPush like this

@Component({

template: `

<h2>{{vData.name}}</h2>

<span>{{vData.email}}</span>

`,

changeDetection: ChangeDetectionStrategy.OnPush

})

<https://www.mokkapps.de/blog/the-last-guide-for-angular-change-detection-you-will-ever-need#trigger-change-detection-manually>

<https://stackoverflow.com/questions/41364386/whats-the-difference-between-markforcheck-and-detectchanges>

* Angular ngIf vs [hidden]: ngIf will not create the template or it won't render the template (i.e it will completly remove the template) but ngShow will create the template and hide it.

ngIf is good pratice when we have to hide some template in the beginning base d on some logic but this is happening frequently then ngIf will be costly thats why in that case we will use ngShow.

* Angular css : any css defined in a particular component is limited to that component only,it won't affect the child nor parent component.

and if we want some css to be applied to the child component the we should use "deep".

* ngSwitch will also hide the template by not creating them just like ngIf or commenting it.
* constructor(private eventService:EventService){}

here we are creating an instance of EventService. above code is same as

let eventService = new EventService();

* @Injectable : when we have to inject some other service or dependencies inside service then we need @Injectable.
* To use a global variable we can declare like:

ex. declare let toastr: any;

* Angular Routing
* canLoad: it will let us decide whether a not a user should load children(lazy loaded modules).

**Injection Token:** A lookup token associated with a dependency provider, for use with the dependency injection system. Using a custom provider allows you to provide a concrete implementation for implicit dependencies, such as built-in browser APIs.

<https://angular.io/guide/dependency-injection-in-action#supply-a-custom-provider-with-inject>  
The Inject decorator is a constructor parameter used to specify a custom provider of a dependency.

* + Use an InjectionToken whenever the type you are injecting is not reified (does not have a runtime representation) such as when injecting an interface, callable type, array or parametrized type.
  + Generally, use Injection Token for third-party global.
  + <https://codecraft.tv/courses/angular/dependency-injection-and-providers/tokens/>

Ex. import { ReflectiveInjector } from '@angular/core';

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

class MandrillService {};

class SendGridService {};

let EmailService = new InjectionToken<string>("EmailService");

let injector = ReflectiveInjector.resolveAndCreate([

{ provide: EmailService, useClass: SendGridService }

]);

let emailService = injector.get(EmailService);

console.log(emailService);

* **Make a dependency @Optional and limit search with @Host**

Dependencies can be registered at any level in the component hierarchy. When a component requests a dependency, Angular starts with that component's injector and walks up the injector tree until it finds the first suitable provider. Angular throws an error if it can't find the dependency during that walk.

In some cases, you need to limit the search or accommodate a missing dependency. You can modify Angular's search behavior with the @Host and @Optional qualifying decorators on a service-valued parameter of the component's constructor.

The @Optional property decorator tells Angular to return null when it can't find the dependency.

The @Host property decorator stops the upward search at the host component. The host component is typically the component requesting the dependency. However, when this component is projected into a parent component, that parent component becomes the host. The following example covers this second case.

* <https://angular.io/guide/dependency-injection-in-action#modify-the-provider-search-with-self-and-skipself>
* **ViewChild:**

<https://angular.io/api/core/ViewChild>

**(**<https://codecraft.tv/courses/angular/components/viewchildren-and-contentchildren/>)

It adds a reference of child from view DOM to the component.

A @ViewChild decorator means, search inside this components template to find the exact DOM node, it’s view, for this child component.

The parameter we pass as the first argument to @ViewChild is the type of the component we want to search for, if it finds more than one it will just give us the first one it finds.

* **ViewChildren:**

<https://angular.io/api/core/ContentChild>

But when there are multiple child components or DOM nodes (using ngFor), then we use Viewchildren. It returns a querylist.

Ex. @ViewChildren(JokeComponent) jokeViewChildren: QueryList<JokeComponent>;

* **ContentChildren:**

The concept of a content child is similar to that of a view child but the content children of the given component are the child elements that are projected into the component from the host component.

* **Differnce between observable and promise**

**Promise:** The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

* + Restricted to single response
  + It is asynchronous

**Observable**

* + An Observable is like a **Stream** of values produced over the time.
  + It is both asynchronous and synchronous
  + It is able to handle multiple values time to time.
  + We can make changes to the response using “map”.

While a Promise only emits the result once, Observables can emit multiple values over time.

Observable also has the advantage over Promise to be cancelable(by unsubscribing it). If the result of an HTTP request to a server isn't needed anymore, the Subscription of an Observable allows to cancel the subscription, while a Promise will eventually call the success or failed callback even when you don't need the notification or the result it provides anymore.

Observables are lazy, which means nothing happens until a subscription is made. Whereas Promises are eager; which means as soon as a promise is created, the execution takes place.

const promise = new Promise((resolve) => {

resolve(10);

});

const obs = new Observable((observer) => {

observer.next(10);

}) ;

<https://www.freecodecamp.org/news/what-are-observables-how-they-are-different-from-promises/>

**SOME ADVANTAGES OF ANGULAR 2 OVER ANGULAR1:**

|  |  |
| --- | --- |
| **Angular 2** | **Angular1** |
| Angular 2 is a mobile-oriented framework | Whereas Angular1 was not developed with mobile base in mind. |
| Angular 2 is a versatile framework, i.e. we have more choices for languages. We can use ES5, ES6, Typescript or Dart to write an Angular 2 code | Whereas an Angular1 code can written by using only ES5, ES6 and Dart. We don’t have many choices of language in Angular1. |
| Nowadays, the controllers are replaced by components and Angular 2 is completely component based. | Whereas Angular1 was based on controllers whose scope is now over. |

**For Extra Knowledge:** <http://info.menlo-technologies.com/blog/angular-1-vs-angular-2-how-do-they-compare>

**What is Traceur Compiler?**

Traceur is a compiler which takes ECMAScript and compiles it down to regular Javascript that runs in your browser.

**What is Webpack?**Webpack is a module bundler. Its main purpose is to bundle JavaScript files for usage in a browser.

**Angular 5 and 6 updates: (**<https://dzone.com/articles/angular-6-release-vs-angular-5-new-features-and-im>**)**

**Angular 11 features:**

<https://www.angularminds.com/blog/article/top-features-of-angular-11.html>

**What is npm?**

Node package manager, It's the world's largest software registry, Open-source developers from every continent use npm to share and borrow packages.

**What are Decorators?**

Decorators are functions that adds metadata to class members and functions. It was proposed in ES2016 and implemented in Typescript.

**Explain $event in Angular5?**

In Angular5 $event is a reserved keyword that represents the data emitted by an event (event data).It is commonly used as a parameter for event based methods.

**double curly** brackets are used form data interpolation in Angular5.

**What is transpiling?**

Transpiling is a process of converting code from one language to another.

**Traceur compiler** takes classes, generators, and other features from ECMAScript edition 6 (ES6) and compiles it into JavaScript ES5 that runs on the browser.

In Angular, Traceur compiler is used for converting TypeScript to JavaScript so that browsers can understand.

**Explain component life cycle in Angular?**

In Angular component life cycle in Angular goes through following stages.

* + Create
  + Render
  + Create and render children
  + Check for bound data changes and re-render
  + Destroy

**Ng-content:** when using a reusable component in Angular, you probably had to project content inside of it. You discovered <ng-content>,

**Polyfill:** A polyfill is a browser fallback, made in JavaScript, that allows functionality you expect to work in modern browsers to work in older browsers, e.g., to support canvas (an HTML5 feature) in older browsers.

**inline.bundle.js:** This is a webpack loader. A tiny file with Webpack utilities that are needed when loading other files. Eventually this will be written inside the index.html file itself and not being generated as a separate file at all.

**vendor.bundle.js:** This is generated by default in dev mode, and ignored by default in prod mode (ng build -prod or ng serve -prod).

It includes the Angular libraries with little or no modification. This is to speed up the build process. Also some people think it's a good idea to keep these in a separate file when it doesn't change much and then it can be cached for longer.

The typical Angular approach though is to merge them into the main bundle, and when doing so, run Webpack tree-shaking, which removes any EcmaScript / TypeScript modules that were never imported from any other modules in your app and its imports. This means the final bundle is much smaller. For example, when running Ahead of Time compiler (AoT), the Angular Compiler gets tree-shaked away.

You can explicitly control generating a separate vendor bundle or not by setting the argument --vendor-chunk.

**main.bundle.js:** Your own code, and anything else you imported etc, as explained in previous point.

**ViewEncapsulation**: ViewEncapsulation decides whether the styles defined in a component can affect the entire application or not.

<https://angular.io/guide/view-encapsulation>

The valid values for this config property are: (<https://codecraft.tv/courses/angular/components/templates-styles-view-encapsulation/>)

* + ViewEncapsulation.Native
  + ViewEncapsulation.Emulated
  + ViewEncapsulation.None.

Native is deprecated instead use ShadowDom

**What is Dependency Injection in Angular 4?**

When a component is dependent on another component the dependency is injected/provided during runtime.

**RouterOutlet:**

Acts as a placeholder that Angular dynamically fills based on the current router state.

**What is the purpose of using package.json in the angular project?**

With the existence of package.json, it will be easy to manage the dependencies of the project. If we are using typescript in the angular project then we can mention the typescript package and version of typescript in package.json

**How is SPA (Single Page Application) technology different from the traditional web technology?**

In traditional web technology, the client requests for a web page (HTML/JSP/asp) and the server sends the resource (or HTML page), and the client again requests for another page and the server responds with another resource. The problem here is a lot of time is consumed in the requesting/responding or due to a lot of reloading. Whereas, in the SPA technology, we maintain only one page (index.HTML) even though the URL keeps on changing.

A **single-page application (SPA)** is a web application or web site that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server.

**A component controls view.**

**What are ngModel and how do we represent it?**

ngModel is a directive which can be applied on a text field. This is two-way data binding. ngModel is represented by [()]

How ngModel works?

<https://stackoverflow.com/questions/44611815/how-does-ngmodel-work-internally-set-value-initially>

**ngModelChange**: <https://angular.io/api/forms/NgModel>

**What does a Subscribe method do in Angular 4?**

It is a method which is subscribed to an observable. Whenever the subscribe method is called, an independent execution of the observable happens.

**Ahead-of-Time (AOT) Compiler**

The Angular Ahead-of-Time (AOT) compiler converts your Angular HTML and TypeScript code into efficient JavaScript code during the build phase before the browser downloads and runs that code.

**What Is a Template Reference Variable?**

A template reference variable is a way of capturing a reference to a specific element, component, directive, and pipe so that it can be used someplace in the same template HTML.

You should declare a reference variable using the hash symbol (#).

**Angular architecture:**



**Entry Components:**

Previously, the entryComponents array in the NgModule definition was used to tell the compiler which components would be created and inserted dynamically. With Ivy, this isn't a requirement anymore and the entryComponents array can be removed from existing module declarations. The same applies to the ANALYZE\_FOR\_ENTRY\_COMPONENTS injection token.

An entry component is any component that Angular loads imperatively, (which means you’re not referencing it in the template), by type. You specify an entry component by bootstrapping it in an NgModule, or including it in a routing definition.

**Ivy:**

Ivy is the code name for Angular's next-generation compilation and rendering pipeline. With the version 9 release of Angular, the new compiler and runtime instructions are used by default instead of the older compiler and runtime, known as View Engine.

**Difference between dependency and dev dependency**

The difference between these two, is that devDependencies are modules which are only required during development, while dependencies are modules which are also required at runtime.

**HostBinding and HostListener:**

<https://www.digitalocean.com/community/tutorials/angular-hostbinding-hostlistener>

**@HostBinding** lets you set properties on the element or component that hosts the directive, and **@HostListener** lets you listen for events on the host element or component.

**Tree-Shaking**

<https://stackoverflow.com/questions/45884414/what-is-tree-shaking-and-why-would-i-need-it>

* <https://www.code-sample.com/2018/06/service-workers-in-angular-5-6-7.html>

FOR XSS attacks

* **Content security policy**

Content Security Policy (CSP) is a defense-in-depth technique to prevent XSS. To enable CSP, configure your web server to return an appropriate Content-Security-Policy HTTP header. Read more about content security policy at the Web Fundamentals guide on the Google Developers website.

* **Enforcing Trusted Types**

We recommend the use of Trusted Types as a way to help secure your applications from cross-site scripting attacks. Trusted Types is a web platform feature that can help you prevent cross-site scripting attacks by enforcing safer coding practices. Trusted Types can also help simplify the auditing of application code.

* **Data share between components:**

<https://levelup.gitconnected.com/5-ways-to-share-data-between-angular-components-d656a7eb7f96>

* **Subject:**

Subject is a special type of Observable in RxJs Library in which we can send our data to other components or services.

* **What is Rest API:**

<https://www.ibm.com/cloud/learn/rest-apis>

<https://www.redhat.com/en/topics/api/what-is-a-rest-api>