*Q1*:

**What is *Routing Guard* in Angular?**

**Added** **Entry** C:\Users\758530\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\9D83F8F0.tmp

Answer

Angular’s **route guards** are interfaces which can tell the router **whether or not it should allow navigation**to a requested route. They make this decision by looking for a true or false return value from a class which *implements* the given guard interface.

There are five different types of guards and each of them is called in a particular sequence. The router’s behavior is modified differently depending on which guard is used. The guards are:

* CanActivate
* CanActivateChild
* CanDeactivate
* CanLoad
* Resolve

Consider:

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

import { Router, CanActivate } from '@angular/router';

import { AuthService } from './auth.service';

@Injectable()

export class AuthGuardService implements CanActivate {

constructor(public auth: AuthService, public router: Router) {}

canActivate(): boolean {

if (!this.auth.isAuthenticated()) {

this.router.navigate(['login']);

return false;

}

return true;

}

}

*Q2*:

## What is a Module, and what does it contain?

**Add to PDF** **Entry** C:\Users\758530\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\6E2EFC7E.tmp

Answer

An Angular **module** is set of Angular basic building blocks like **component**, **directives**, **services** etc. An app can have more than one module.

A module can be created using @NgModule decorator.

@NgModule({

imports: [ BrowserModule ],

declarations: [ AppComponent ],

bootstrap: [ AppComponent ]

})

export class AppModule { }

*Q3*:

**What's the difference between an Angular Component and Module?**

**Add to PDF** **Junior** C:\Users\758530\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\2D6B5D3C.tmp

Answer

* **Components** control views (html). They also communicate with other components and services to bring functionality to your app.
* **Modules** consist of one or more components. They do not control any html. Your modules declare which components can be used by components belonging to other modules, which classes will be injected by the dependency injector and which component gets bootstrapped. Modules allow you to manage your components to bring modularity to your app.

*Q4*: **What is a Component? Why would you use it?**

**Add to PDF** **Junior** C:\Users\758530\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\27F13596.tmp

Answer

**Components** are the most basic building block of an UI in an Angular application. An Angular application is a tree of Angular components. Angular components are a subset of directives. Unlike directives, components always have a template and only one component can be instantiated per an element in a template.

A component must belong to an NgModule in order for it to be usable by another component or application. To specify that a component is a member of an NgModule, you should list it in the *declarations* field of that NgModule.

@Component({selector: 'greet', template: 'Hello {{name}}!'})

class Greet {

name: string = 'World';

}

*Q6*: **What is a *Service*, and when will you use it?**

**Add to PDF** **Junior** C:\Users\758530\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\4067AE14.tmp

Answer

Angular **services** are singleton objects which get instantiated only once during the lifetime of an application. They contain methods that maintain data throughout the life of an application, i.e. data does not get refreshed and is available all the time. The **main objective** of a service is to **organize and share business logic**, models, or data and functions with different components of an Angular application.

The *separation of concerns* is the main reason why Angular services came into existence. An Angular service is a stateless object and provides some very useful functions.

*Q7*: What is the difference between @Component and @Directive in Angular?

**Add to PDF** **Junior** C:\Users\758530\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\A77A8142.tmp

Answer

* **Directives** add behaviour to an existing DOM element or an existing component instance.
* **A component**, rather than adding/modifying behaviour, actually creates its own view (hierarchy of DOM elements) with attached behaviour.

Write a component when you want to create a reusable set of DOM elements of UI with custom behaviour. Write a directive when you want to write reusable behaviour to supplement existing DOM elements.

*Q12*: **How would you run unit test?**

**Add to PDF** **Junior** C:\Users\758530\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\CADC72A0.tmp

Answer

The Angular CLI downloads and install everything you need to test an Angular application with the Jasmine test framework.

The project you create with the CLI is immediately ready to test. Just run this one CLI command:

ng test

*Q14*: **What are Observables?**

**Add to PDF** **Junior** C:\Users\758530\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\97D291AE.tmp

Answer

**Observables** are declarative which provide support for passing messages between publishers and subscribers in your application.

They are mainly used for event handling, asynchronous programming, and handling multiple values. In this case, you define a function for publishing values, but it is not executed until a consumer subscribes to it. The subscribed consumer then receives notifications until the function completes, or until they *unsubscribe*.

*Q15*: **What is *Interpolation*?**

Answer

**Interpolation** is a special syntax that Angular converts into *property binding*. It’s a convenient alternative to property binding. It is represented by double curly braces({{}}). The text between the braces is often the name of a component property. Angular replaces that name with the string value of the corresponding component property.

Let's take an example,

<h3>

{{title}}

<img src="{{url}}" style="height:30px">

</h3>

In the example above, Angular evaluates the title and url properties and fills in the blanks, first displaying a bold application title and then a URL.

*Q16*: **What is an Observer?**

Answer

**Observer** is an interface for a consumer of push-based notifications delivered by an Observable. It has below structure,

interface Observer<T> {

closed?: boolean;

next: (value: T) => void;

error: (err: any) => void;

complete: () => void;

}

A handler that implements the Observer interface for receiving observable notifications will be passed as a parameter for observable as below,

myObservable.subscribe(myObserver);

**Note:** If you don't supply a handler for a notification type, the observer ignores notifications of that type.

*Q17*: What is the difference between Structural and Attribute directives in Angular?

  Answer

* **Structural directives** are used to alter the DOM layout by removing and adding DOM elements. It is far better in changing the structure of the view. Examples of Structural directives are NgFor and Nglf.
* **Attribute Directives** These are being used as characteristics of elements. For example, a directive such as built-in NgStyle in the template Syntax guide is an attribute directive.

*Q18*: **What is a *bootstrapping module*?**

C:\Users\758530\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\770A00C4.tmp

Answer

Every application has at least one Angular module, the root module that you *bootstrap* to launch the application is called as **bootstrapping module**. It is commonly known as AppModule. The default structure of AppModule generated by AngularCLI would be as follows:

/\* JavaScript imports \*/

import {

BrowserModule

} from '@angular/platform-browser';

import {

NgModule

} from '@angular/core';

import {

FormsModule

} from '@angular/forms';

import {

HttpClientModule

} from '@angular/common/http';

import {

AppComponent

} from './app.component';

/\* the AppModule class with the @NgModule decorator \*/

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

FormsModule,

HttpClientModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule {}

*Q19*: **What is the purpose of base href tag?**

Answer:

The routing application should add element to the index.html as the first child in the tag inorder to indicate how to compose navigation URLs. If app folder is the application root then you can set the href value as below

<base href="/">

*Q20*: **What is an Observable?**

Answer

An **Observable** is a unique Object similar to a Promise that can help manage async code. Observables are not part of the JavaScript language so we need to rely on a popular Observable library called **RxJS**.

The observables are created using new keyword. Let see the simple example of observable,

import { Observable } from 'rxjs';

const observable = new Observable(observer => {

setTimeout(() => {

observer.next('Hello from a Observable!');

}, 2000);

});`

*Q21*: **What are *Pipes*? Give me an example.**

**Add to PDF** **Entry** C:\Users\758530\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\D21032DE.tmp

Answer

A **pipe** takes in data as input and transforms it to a desired output. You can chain pipes together in potentially useful combinations. You can write your own custom pipes. Angular comes with a stock of pipes such as DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, and PercentPipe.

Consider:

<p>The hero's birthday is {{ birthday | date }}</p>

In this page, you'll use pipes to transform a component's birthday property into a human-friendly date.

*Q22*:  **What is *Angular Universal*?**

Angular Universal is the process of **server-side rendering** (SSR) your application to HTML on the **Server** (ie: Node.js).

*Q23*:**Explain Lazy Loading in Angular?**

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.

*Q24*: **Explain the difference between Constructor and ngOnInit?**

* Most of the time we use ngOnInit() for all the initialization/declaration.
* It’s better to avoid writing actual work in the constructor.
* The constructor() should only be used to initialize class members but shouldn't do actual "work".
* So we should use constructor() to setup Dependency Injection, Initialization of class fields, etc.
* ngOnInit() is a better place to write "actual work code" that we need to execute as soon as the class is instantiated.

*Q25*: **What is the difference between @Component and @Directive in Angular?**

Decorator that marks a class as an Angular component and provides configuration metadata that determines how the component should be processed, instantiated, and used at runtime.

Directives can change the appearance or behavior of DOM elements and Angular components.

*\*\*\*Q26*: **Explain the difference between Promise and Observable in Angular?**

**Promises** deal with one asynchronous event at a time, while **observables** handle a sequence of asynchronous events over a period of time.

*\*\*\*Q27*: **Why should ngOnInit be used, if we already have a constructor?**

Mostly **we** use **ngOnInit for** all the initialization/declaration and avoid stuff to work in the **constructor**. The **constructor should** only be **used** to initialize class members but shouldn't do actual "work". So **you should** use **constructor**() to setup Dependency Injection and not much else.

*\*\*\*Q28*: **What is difference between declarations, providers and import in *NgModule*?**

*\*\*\*Q30*: **What is *Router Outlet*?**

# **RouterOutlet**

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

*Q31*: **What are the ways to control AOT compilation?**

*\*\*\*Q32*: **Do I always need a *Routing Module*?**

The Routing Module is a design choice whose value is most obvious when the configuration is complex and includes specialized guard and resolver services. It can seem like overkill when the actual configuration is dead simple.

Some developers skip the Routing Module (for example, AppRoutingModule) when the configuration is simple and merge the routing configuration directly into the companion module (for example, AppModule).

*\*\*\*Q45*: **What is the purpose of *Wildcard route*?\*\***

A **Wildcard route** has a **path** consisting of two asterisks (\*\*). It matches every URL, the router will select this **route** if it can't match a **route** earlier in the configuration. A **Wildcard Route** can navigate to a custom component or can redirect to an existing **route**.

*\*\*\*Q46*: **What is *Router State*?**

Represents the state of the router as a tree of activated routes.