Angular interview questions

1. Angular advantages

* Custom reusable components : We can create our own components as per requirements and which can be reusable across the application
* Productivity and code consistency : Components and services
* Easy testing – Unit testing and we can add debugger in browser with sourcemap as true
* High compatibility : Cross platform compatible e.g chrome or firefox. Can develop small, medium, large applications. JS is in constant state of flux so angular is a framework for which company can rely on as it provides constant support and upgradation

1. Router

* [@angular/router](https://v17.angular.io/api/router) API
* Define routes object
* // AppRouting.module.ts  
  const routes = [

{

path : ‘home’,

component: HomeComponent,

title: ‘Home’,

canActivate: [CanActivateAuthGuard],

loadChildren: () =>

import('./menu/menu.module').then((x) => x.MenuModule), // it will load menumodule where we will specify menu routes

pathMatch: full | prefix,  
},

{  
path: ‘learn’,

component: LearnComponent,

children:[

{

path : ‘learnAngular’,

component: LearnAngularComponent

},

path : ‘learnReact’,

component: LearnReactComponent

]},

{

path: 'team/:id',

component: TeamComponent

},

// default route

{

path: '',

redirectTo: '/home’,

pathMatch: 'full'

},

// 404 page not found if any route not matches

{

path: '\*\*',

component: WildcardComponent

},

],

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

// MenuRoutes.modules.ts

export const MenuRoutes: Routes = [

{

path: 'menu',

component: MenuListComponent

},

{

path: 'menu/:id',

component: MenuSingleComponent

}

];

@NgModule({

imports: [RouterModule.forChild(MenuRoutes)]

})

export class MenuModule {}

1. Components

* Main building blocks.
* Components provide us the way to build ui using template, logic file ts , styles files.
* Components are reusable
* @Component({

selector : ‘app-component-overview’,

templateUrl:’./component-overview.component.html’,

styleUrls : [‘./component-overview.component.scss’]  
})

1. Compilation types

* AOT
  + Best and recommended Compiler
  + Default compilation from Angular 9
  + Code compiles before loading into the browser
  + Source code translate to optimized JS code during the development phase
  + Reduces initial load time as it pre-compiles
  + Detects and reports errors during compilation phase so developer can resolve it and load application
  + Leading to reliable application with fewer runtime issues
  + ng build –aot
  + ng serve –aot
  + To create a production build with AOT optimizations, run the following command: ng build –prod

* JIT
  + Compiles code during run time, right into your browser
  + Detects and reports error during runtime
  + When opens web app in browser JIT starts compilation
  + JIT compiler reads and understands the code
  + Then translate to javascript code understandable by browser
  + Browser uses this translated JS code to build and display the applicatopn
  + More Chances of getting error during running the application in browser
  + You can use this during development but recommended AOT for prod

1. What are the advantages of TypeScript over JavaScript?
2. Directives

* Directives are used to add some behavior to DOM elements
* We can build custom directives too
* Attribute directive
  + Change look and feel of element
  + ngStyle , ngClass
  + Custom attribute directive

import { Directive, ElementRef } from '@angular/core';

@Directive({

selector: '[appHighlight]'

})

export class HighlightDirective {

constructor(private eleRef: ElementRef) {

eleRef.nativeElement.style.background = 'red';

}

}

* Structural directive
  + Add or remove element
  + \*ngIf
  + Custom structural directive

import { Directive, Input, TemplateRef, ViewContainerRef } from '@angular/core';

@Directive({

selector: '[appNot]'

})

export class AppNotDirective {

constructor(

private templateRef: TemplateRef<any>,

private viewContainer: ViewContainerRef) { }

@Input() set appNot(condition: boolean) {

if (!condition) {

this.viewContainer.createEmbeddedView(this.templateRef);

} else {

this.viewContainer.clear(); }

}

}

1. Services

* Common tasks can be written is services
* It is used to share the data within application
* Reusable
* Code organization
* Dependency injection – Services are injected into components using DI

1. Dependency injection

* For service to be used in component first it needs to be injected. DI takes care of it
* Component doesn’t need to worry about creating the service as it will be take care by DI
* Modular so reusable
* We can use service in component, directive, pipes and other service

1. Providers
   1. Object for managing and creating instances of dependencies that can be injected into component and services
   2. Import { Component, Injectable } from ‘@angular/core’;

@Injectable()

export class MyService{

getData() {  
 return ‘some data’;

}

}

@Component({

Selector: ‘app-my-component’,

Providers: [ MyService ],

Template: ‘templateurl’

})

export class MyComponent {

constructor(private myService: MyService) {}

data = this.myService.getData();

}

1. Injectors
   1. Injectors responsible for managing and creating dependencies
   2. import { Component, Injectable, Injector } from '@angular/core';

@Injectable()

export class MyService {

getData() {

return "Data from MyService";

}

}

@Component({

selector: 'my-component',

template: '{{ data }}'

})

export class MyComponent {

constructor(private injector: Injector){}

data = this.injector.get(MyService).getData();

}

1. Normal way
   1. import { Component, Injectable, Injector } from '@angular/core';

@Injectable(

{

providedIn: 'root'

}

)

export class MyService {

getData() {

return "Data from MyService";

}

}

@Component({

selector: 'my-component',

template: '{{ data }}'

})

export class MyComponent {

constructor( private myService: MyService ){}

data = this.myService.getData();

}