1. **What is Angular Framework?**

Angular is a Typescript-based open-source front-end platform that makes it easy to build applications with in web/mobile/desktop. The major features of this framework such as declarative templates, dependency injection, end to end tooling, and many more other features are used to ease the development.

1. **What is TypeScript ?**

TypeScript is a strongly typed, object oriented, compiled language. TypeScript is both a language and a set of tools. TypeScript is a typed superset of JavaScript compiled to JavaScript. It is JavaScript with static typed feature. Because in bigger project where many persons are working, there may be cases where we are using multiple objects and variables and without declaration of types we may land up in error as we will not get any warning while compiling .

TypeScript is a superset of ECMAScript 6 (ES6) and is backward compatible with ECMAScript 5.

1. **What is array type and object type in Typescript ?**

Let arr:String[]

Let obj: { name:String,Age:number}[]---this array of objects

1. **What is type Inference ?**

When we immediately initialize a variable with some value(a=2) then we don’t need to give type to that variable .Typescript will automatically understand and declare its type with the type of its value.

Let name= “Piyush” or let name :String =”Piyush”

Both are equal and 2nd is not needed.now name has became string type variable .

Name = 1245 will give error.

1. **What is visibilityState?**

setInterval( () => {

console. log(document.visibilityState);

}, 1000);

**document.visibilityState : this tells whether user is on the tab of our application or not.**

**Gives output : visible or hidden**

1. **What is union type and type aliases ?**

Let name : string | number;

So using union we are assigning name variable to both string and number type.

Type is a special keyword to declare custom type or alias .

type person ={

Name:string,

Age: number

}

Let obj: person; -------so obj will become person type variable

1. **What is void return type in typescript?**

When we don’t return anything through a function then void is default return type.

Function adding(a:number,b:number):number{

Return a+b;

}

1. **What is Generic type function in TypeScript ?**

we need a way of capturing the type of the argument in such a way that we can also use it to denote what is being returned. Here, we will use a type variable, a special kind of variable that works on types rather than values.

function loggingIdentity<Type>(arg: Type[]): Type[] {

console.log(arg.length);

return arg;

}

1. **What are difference between class in JS vs TS ?**

class BadGreeter {

name: string; ------ **error: Property 'name' has no initializer and is not definitely assigned in the constructor.**

}

The field needs to be initialized in the constructor itself. TypeScript does not analyze methods you invoke from the constructor to detect initializations, because a derived class might override those methods and fail to initialize the members.

We will have to use constructor to initialize class variables in TypeScript.

class GoodGreeter {

name: string;

constructor() {

this.name = "hello";

}

}

Fields may be prefixed with the readonly modifier. This prevents assignments to the field outside of the constructor.

class Greeter {

readonly name: string = "world";

constructor(otherName?: string) {

if (otherName !== undefined) {

this.name = otherName;

}

}

err () {

this.name = "not ok"; ------------------this gives error -----Cannot assign to 'name' because it is a read-only property.

}

}

1. **What are access modifies in Typescript?**

TypeScript provides three access modifiers**:**

* **Private---** The private modifier limits the visibility to the same-class only. When you add the private modifier to a property or method, you can access that property or method within the same class
* **Protected ----** The protected modifier allows properties and methods of a class to be accessible within same class and within subclasses.

class Person {

protected ssn: string;

}

* **Public -----** The public modifier allows class properties and methods to be accessible from all locations. If you don’t specify any access modifier for properties and methods, they will take the public modifier by default.

1. **What are interfaces in TypeScript ?**

In TypeScript, an interface is an abstract type that tells the compiler which property names a given object can have. TypeScript creates implicit interfaces when you define an object with properties.

interface User {

id: number;

firstName: string;

lastName: string;

getFullName(): string;

}

We can use an interface with a class using the keyword implements. this is a major difference between Type keyword and interfaces.

interface Size {

width : string,

height: string,

getWidth(): string;

}

class Shapes implements Size {

width: string;

height: string;

constructor (width:string, height:string) {

this.width = width;

this.height = height;

}

getWidth() {

return this.width;

}

}

There are certain cases, however, where we would expect our objects to have properties that are optional. We can achieve this by placing a question mark (?) just before the property’s type annotation when declaring the interface

We can create a generic interface by passing type parameters, using angle brackets (<>), to our interface.

1. **What is Blazel Tool?**

Bazel is a [free software](https://en.wikipedia.org/wiki/Free_software) tool that allows for the automation of building and testing of software.

Bazel is an advanced build and test tool. It supports lot of features suitable for large projects.

Some of the features of Bazel are as follows:

* Support multiple languages.
* Support multiple platforms.

Angular supports building the application using bazel. The angular framework is itself built in Bazel.

1. **What is Angular CLI ?**

Angular CLI is a command-line interface (CLI) to automate your development workflow. It allows you to: create a new Angular application. run a development server with LiveReload support to preview your application during development

1. **Why were client-side frameworks like Angular introduced?**

Back in the day, web developers used VanillaJS and jQuery to develop dynamic websites but, as the logic of one's website grew, the code became more and more tedious to maintain. For applications that use complex logic, developers had to put in extra effort to maintain separation of concerns for the app. Also, jQuery did not provide facilities for data handling across views.  
For tackling the above problems, client-side frameworks like Angular came into the picture, which made life easier for the developers by handling separation of concerns and dividing code into smaller bits of information (In the case of Angular, called Components).  
Client-side frameworks allow one to develop advanced web applications like Single-Page-Application. Not that we cannot develop SPAs using VanillaJS, but by doing so, the development process becomes slower.

1. **How can be change default port in angular ?**

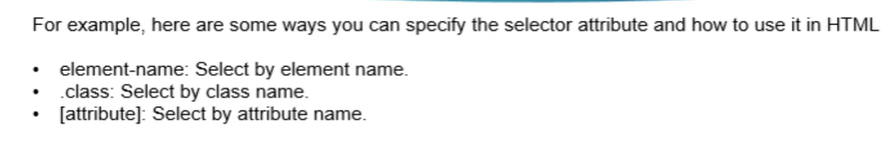
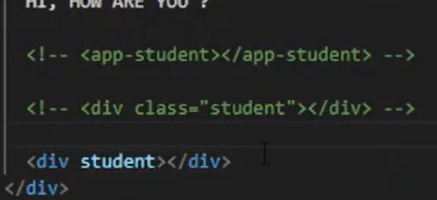
Default port is 4200.

We can change it using : ng serve –port=Your port number

1. **What are the three parts of a component ?**

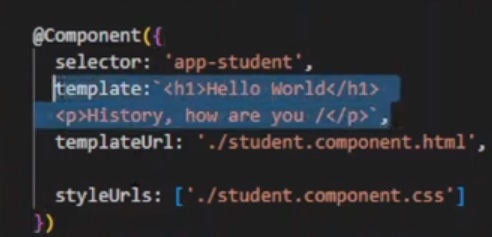
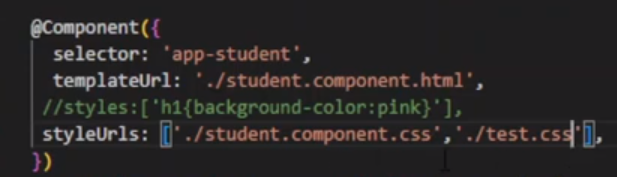
Each component is declared with three properties:  
1. Selector - used for accessing the component  
2. Template/TemplateURL - contains HTML of the component  
3. StylesURL - contains component-specific stylesheets.

*Selectors can be used in HTML in 3 format: 1)element 2)class 3)attribute*



**Templte/templteURL :**

* TemplateURL has more priority than templte. In Example below TemplateURL will be loaded.

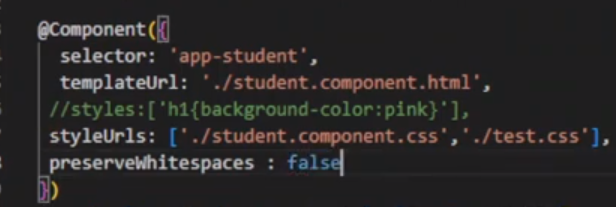


**Style/styleURLS :**

* Styles has more priority then styleURLS. Because they are inline CSS.
* We can have multiple .css file in styleURLs arr.
* IF we write a css in global css file which style.css and also in our component css file then our component css will be applied.

1. **Can Components be nested ?**

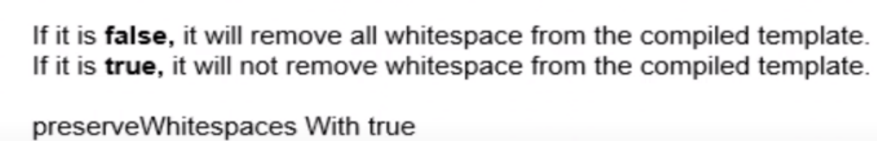
Yes we can call any component inside any other component .

1. **What is preserveWhiteSpace in angular ?**

It is a property used in @component decorator.

By Defualt it is false.We can set it to true.

It will not show as many space as we will give.It will show

only default space allowed.

1. **What is viewProviders ?**

We know that the decorator functions of @Component take object and this object contains many properties.

The viewProviders property allows us to make providers available only for the component's view.

When we want to use a class in our component that is defined outside the @Component () decorator function, then,

first of all, we need to inject this class into our component, and we can achieve this with the help of the "viewProvider" property of a component.

**class abc{**

**constructor(){**

**console.log("abc constructor called .. ");**

**}**

**display(name:string){**

**console.log("Hello : "+ name);**

**}**

**}**

**@Component({**

**selector: 'app-root',**

**templateUrl: './app. component.html',**

**styleUrls: ['./app.component.css'],**

**viewProviders: [Employee, abc]**

**}**

Here abs CLASS ihas been created inside component and Employee CLASS has been created inside outside using command (ng g class employee).

We can use this class in MODULE level by injecting them in module.ts file.

1. **What is ng-template ?**

* <ng-template> is a virtual element and its contents are displayed only when needed (based on conditions).
* <ng-template> should be used along with structural directives like [nglf],[ngFor],[NgSwitch] or custom structural directives.
* <ng-template> never meant to be used like other HTML elements. It's an internal implementation of Angular's structural directives.
* When you use a structural directive in Angular we will add a prefix asterisk(\*) before the directive name. This asterisk is short hand notation for <ng-template>.
* Whenever Angular encounter with the asterisk(\*) symbol, we are informing Angular saying that it is a structural directive and Angular will convert directive attribute to <ng-template> element.
* <ng-template> is not exactly a true web element. When we compile our code, we will not see a <ng-template> tag in HTML DOM.Angular will evaluate the <ng-template> element to convert it into a comment section in HTML DOM.

1. **What is ng-container ?**

ng-container is an element that's available in Angular 2+.<ng-container> is a logical container that can be used to group nodes but is not rendered in the DOM tree as a node. <ng-container> is rendered as an HTML comment.

In order to avoid having to create that extra div, we can instead use ng-container directive.<ng-container> is an Angular grouping element that is similar to <ng- template> in that it doesn't represent a DOM element. The difference is that it will always be rendered, whereas an <ng-template> will only be rendered if it is explicitly requested. <ng-container> are useful anywhere you need an extra container for some template elements, but don't want to (or can't) create a container such as a

div to hold them with due to syntax or style constraints.

1. **Can we use multiple structural directive in one element ?**

No we cannot use.Like we cannot use ngIf and ngFor together.

<ul>

<ng-container \*ngFor="let book of books" >

<li \*ngIf="display">

{{book.name}}

</li>

</ng-container>

</ul>

<ng-container \*ngTemplateOutlet="template ;context:obj">

</ng-container>

<ng-template #template let-ob="data" >

{{ob}}

</ng-template>

1. **What is encapsulation in Angular?**

To emulate the shadow DOM and encapsulate styles, Angular provides three types of view encapsulation. Are the

following:

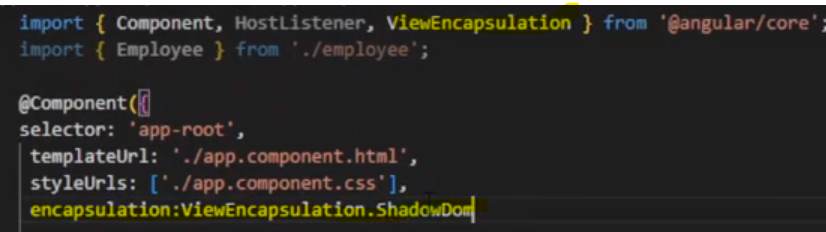
* 1. Emulated (default): The main HTML styles are propagated to the component. The styles defined in the component's @Component decorator are limited to this component only.
  2. Native/shadow dom: The main HTML styles are not propagated to the component. The styles defined in the

component's @Component decorator are limited to this component only.

It will pass css from parent to child i.e. it can go in only in lower hierarchy .

* 1. None: the component styles are propagated to the main HTML and therefore are visible to all components of the page. with apps that have None and Native components in the application .. All components with encapsulation None will have their duplicate styles on all components with native encapsulation.

It will pass css from child to parent i.e. it can go in upper hierarchy .



1. **What are providers in Angular ?**

Providers metadata is available in the @NgModule() decorator and @Component() decorator. provider's metadata is

used to configure the provider.

The injector injects the objects provided by provider into components and services. Only those classes configured by

providers are available for dependency injection (DI).

Provider can be used for : 1. Module Level 2. Component Level

1. **How does an Angular application work?**

Step1 : Every Angular app consists of a file named angular.json. This file will contain all the configurations of the app. While building the app, the builder looks at this file to find the entry point of the application.

Step2 : Inside the build section, the main property of the options object defines the entry point of the application which is mainly main.ts.

Step3 : The main.ts file creates a browser environment for the application to run, and, along with this, it also calls a function called bootstrapModule, which bootstraps the application.

Step 4 : After this, Angular calls the index.html file. This file consequently calls the root component that is app-root. The root component is defined in app.component.ts.

5.**What are some of the advantages of Angular over other frameworks?**

Features that are provided out of the box - Angular provides a number of built-in features like

* Routing
* state management
* rxjs library
* http
* services straight out of the box.

This means that one does not need to look for the above stated features separately. They are all provided with angular.

1. **What are ng prime filter options?**

PrimeNG filtering type can be any one of the following -

* StartsWith -Returns the values which start with the specified value.
* EndsWith -Returns the values which ends with the specified value.
* Contains -Returns the values which contains the specified value.
* Equals -Returns the values which exactly matches the specified value.

1. **List out differences between AngularJS and Angular**

**Architecture**

AngularJS uses MVC or Model-View-Controller architecture, where the Model contains the business logic, Controller processes information and View shows the information present in the Model.  
Angular replaces controllers with Components. Components are nothing but directives with a predefined template.

**Language**

AngularJS uses JavaScript language, which is a dynamically typed language.  
Angular uses TypeScript language, which is a statically typed language and is a superset of JavaScript. By using statically typed language, Angular provides better performance while developing larger applications.

**Mobile Support**

AngularJS does not provide mobile support.  
Angular is supported by all popular mobile browsers.

**Structure**

While developing larger applications, the process of maintaining code becomes tedious in the case of AngularJS.  
In the case of Angular, it is easier to maintain code for larger applications as it provides a better structure.

| **AngularJS** | **Angular** |
| --- | --- |
| It is based on MVC architecture | This is based on Service/Controller |
| It uses JavaScript to build the application | Introduced the TypeScript to write the application |
| Based on controllers concept | This is a component based UI approach |
| Not a mobile friendly framework | Developed considering mobile platform |
| Difficulty in SEO friendly application development | Ease to create SEO friendly applications |

1. **What is AOT compilation? What are the advantages of AOT?**

Angular provides two types of compilation:

\* JIT(Just-in-Time) compilation : [Just in time compiler](https://www.geeksforgeeks.org/just-in-time-compiler/) provides compilation during the execution of the program at a run time before execution. In simple words, code get compiles when it’s needed, not at the build time.

* Just in time compiler compiles each file separately and it’s mostly compiled in the browser. You don’t have to build your project again after changing your code.
* Most compiling is done on the browser side, so it will take less compiling time.
* If you have a big project or a situation where some of your components don’t come in use most of the time then you should use the Just in time compiler.

Just in Time compiler is best when your application is in local development.

\* AOT(Ahead-of-Time) compilation

The advantages of using AOT compilation are:

1 Since the application compiles before running inside the browser, the browser loads the executable code and renders the application immediately, which leads to faster rendering.

2 Developers can detect and handle errors during the building phase, which helps in minimizing errors.

3 The AOT compiler adds HTML and templates into the JS files before they run inside the browser. Due to this, there are no extra HTML files to be read, which provide better security to the application.

1. **What are the different phases of AOT ?**

**There are three phases of AOT compilation.**

**Phase 1** is code analysis. In this phase, the TypeScript compiler and AOT collector create a representation of the source. The collector does not attempt to interpret the metadata it collects. It represents the metadata as best it can and records errors when it detects a metadata syntax violation.

**Phase 2** is code generation. In this phase, the compiler's StaticReflector interprets the metadata collected in phase 1, performs additional validation of the metadata, and throws an error if it detects a metadata restriction violation. Read More Here [Angular Training](https://onlineitguru.com/angularjs-online-training-placement.html).

**Phase 3** is template type checking. In this optional phase, the Angular template compiler uses the TypeScript compiler to validate the binding expressions in templates. You can enable this phase explicitly by setting the fullTemplateTypeCheck configuration option; see Angular compiler options.

1. **What is ElementRef ?**

Angular ElementRef is a wrapper around a native DOM element (HTML element) object. It contains the property nativeElement, which holds the reference to the underlying DOM object. We can use it to manipulate the DOM. We use the [ViewChild](https://www.tektutorialshub.com/angular/understanding-viewchild-viewchildren-querylist-in-angular/) to get the ElementRef of an HTML element in the component class.

It can be used to manipulate any DOM elements content like we can use it to clear the file name selected after uploading the file. Ex: <div #'inputFile2'>Hello Angular</div>

@ViewChild('inputFile2') myInputVariable2: ElementRef;

And then this.myInputVariable2.nativeElement.value = '';

1. **What are different testing frameworks in angular ?**

Jasmin---for unit testing

Karma – for test runner and execution

Protactor – for end to end test scripts i.e. e2e which is present in out project in e2e folder.

1. **Explain the folder structure of the Angular project ?**

* e2e folder,
* node\_modules,
* src (app folder, assets folder, environment folder, index.html, style.css, polyfills.tc),
* package.json,
* package-lock.json,
* angular.json

1. **What is the difference between package.json and package-lock.json ?**

* package.json -------------- Contains all dependencies of the project
* package-lock.json -------- Conatins all dependencies and configurations of the project.

1. **When would you use the constructor() versus the ngOnInit() method?**

To answer this question, we need to understand a component lifecycle and the role of constructor. Angular creates components based on two phases: constructing components tree and running change detection. The constructor() method is invoked in the first step. Component Lifecycle Hooks are methods on Component or Directive that Angular calls at a specific moment of the change detection process. The ngOnInit() method is the second at this lifecycle sequence. It is called once and means that the object is ready to use because Angular already set all input properties and displayed the data-bound properties.

The code added to the constructor is always initialized before the ngOnInit() method. We need to be sure that logic set in the constructor is not added too early (when the component is out of control).

We typically use the constructor to Inject Dependencies. Also, practical experience says - the less logic in the constructor the better. Moreover, remember that input binding is not available in constructor considering the change detection and the Input communication mechanism. The ngOnInit is a great place to add logic for a component that is outside of Dependency Injection, Input Binding, DOM, Router ecosystem.

1. **What practices could you adopt to share data to another component elsewhere in the component tree?**

*1. Sharing data between Parent <-> Child component*

* Input binding
* Input property setter
* EventEmmiter()
* template reference variable (local variable)
* ViewChild injection

*2. Sharing data between unrelated components*

* Subject() or BehaviourSubject() in a shared service,
* passing data to components via NGRX selectors

1. **What is @input() ?**

Input property is used within one component (child component) to receive a value from another component (parentcomponent). This is a one-way communication from parent to child. A component can receive a value from another component through component input property.

Inside Parent :

<div style="text-align: center">

<h1>Welcome to Sahosoft Solutions !!< /h1>

<h1>Parent Component</h1>

Parent Component : <input type="text" #Pcomponent (keyup)="0"/>

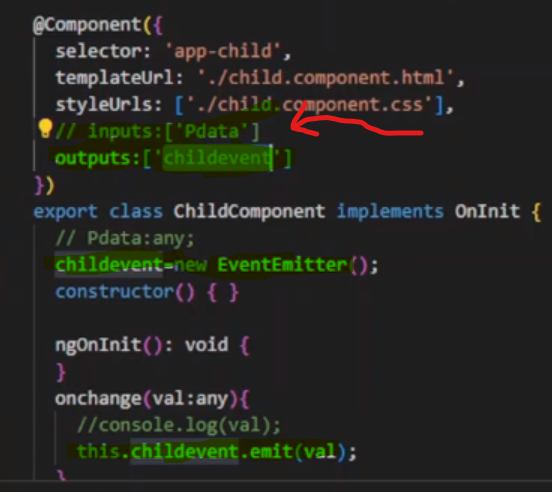
<app-child [Pdata]="Pcomponent.value"></app-child>

</div>

Inside Child :

@Input() Pdata:string;

1. **What is @output() ?**

Output decorates the property as the output property. We initialize it as an EventEmitter. The child component raises the event and passes the data as the argument to the event. The parent component listens to events using [event binding](https://www.tektutorialshub.com/angular/event-binding-in-angular/) and reads the data.

@Output() customerChange:EventEmitter<Customer> =new EventEmitter<Custr>();

//Raise the event to send the data back to parentome

update() {

  this.customerChange.emit(this.customer);

}

<app-customer-detail [customer]="selectedCustomer" (customerChange)="update($event)"></app-customer-detail>

Using @input() data from parent <app-customer> is passed to <app-customer-detail> child.

After making changes in child, it passes information to parent using @output().

Both @input() and @output() are defined inside Child component.

And to we will create a function in parent to react on incoming @output event.

1. **What is event binding** ?

When a user interacts with an application in the form of a keyboard movement, a mouse click, or a mouseover, it generates an event. These events need to be handled to perform some kind of action.This is where event binding comes into picture.

Syntax:

<button (click)="onClick()">Click me</button>

In this case, the onClick() method of the component class is called when the click event occurs.

1. **What is Eager Loading and Pre-loading ?**

Eager loading is default loading strategy in Angular. Here modules are loaded before application starts. As soon as we type ng serve it loads all modules and then in frontend Page is loaded.

**Pre-loading** will load modules in background automatically after application starts. We need to import PreLoadingStartegy from angular/routes to use it.

Modules are loaded only once.

import {PreloadAllModules} from '@angular/router';

Inside app.routing.ts file :

@NgModule({

imports: [

RouterModule.forRoot(routes,

{

preloadingStrategy: PreloadAllModules

})

]

1. **Why adopt a modular architecture that can be lazy-loaded?**

Lazy-loading is a design pattern that doesn’t load all ngModules that build Angular app but only indeed modules for a specific route. It’s a great way to decrease bundle size and reduce the loading time of your app. The more complex app, the more relevant usage of this pattern.

We can define lazy-loading modules as a part of **Route Configurations.**

{ path: '/admin', loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule) }

**Lazy-loaded** modules are a good practice to improve app performance but consider declaring the default page route as non-lazy. Don’t apply it for default route because you can get the reverse effect. It could slow down the initial page rendering by triggering extra HTTP requests and performing unnecessary computations during the initial page load.

If we add service in provider[] in eager-loaded module and also in Lazy-loaded module both then we will get two instances of the services , which may be we don’t want .So we must keep this in mind.

If we are using Angular 7 or lower versions, then use the following syntax

The loadChildren accepts the value as string. The string is split into two sections separated by #.The first part is the full path to the module file (without the ts extension). In the example above ./admin/admin.module points to admin.module.ts file. The second part is the export class name of the Module. i.e AdminModule

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

1. **What is CUSTOM PRELOADING ?**

With PreloadAllModules all the modules are preloaded, which may actually create a bottleneck if the application has large no of modules to be loaded.The better way strategy would be :

* + 1. Eagerly Load the modules required at startup. For Example authentication module, core module, shared module etc
    2. Preload all frequently used modules, may be after some delay
    3. Lazy load remaining modules

Inside app.routing.ts :

const routes: Routes = [

  {path: "admin", loadChildren:'./admin/admin.module#AdminModule',data: { preload: **true**, delay:5000 }},

  {path: "test", loadChildren:'./test/test.module#TestModule',data: { preload: **true**, delay:10000 }},

];

import { PreloadingStrategy, Route } from '@angular/router';

@Injectable()

export class CustomPreloadingWithDelayStrategy implements PreloadingStrategy {

preload(route: Route, load: () => Observable<any>): Observable<any> {

if (route.data && route.data['preload']) {

console.log('Preload Path: ' + route.path + ', delay:' + route.data['delay']);

if (route.data['delay']) {

delay:number=route.data['delay']

return timer(delay).mergeMap(() => load());

}

return load();

else {

return Observable.of(null);

}

1. **Why do Observables and reactive programming fit well with Angular?**

Observables and reactive programming are extremely effective to deal with asynchronous programming. Some elements of Angular architecture return streams of values by default. Let’s take a look for sources of asynchronous streams in Angular that return Observable:

* http methods (get(), put(), …)
* async pipe
* Router.events
* EventEmmiter
* AbstractControl.valueChanges
* ActivatedRoute.params

Observables are pretty productive to handle values emitted asynchronously. A bunch of operators of reactive programming allows you to transform, filter data and make your code clear and easy to understand**.**

1. **Explain Components, Modules and Services in Angular ?**

**Components :**  
In Angular, components are the basic building blocks, which control a part of the UI for any application.  
A component is defined using the @Component decorator. Every component consists of three parts:

A) The template which loads the view for the component,

B) a stylesheet which defines the look and feel for the component

C) a class that contains the business logic for the component.

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

@Component({

selector: 'app-test',

templateUrl: './test.component.html',

styleUrls: ['./test.component.css']

})

**Services :**

Services are objects which get instantiated only once during the lifetime of an application. The main objective of a service is to share data, functions with different components of an Angular application.  
A service is defined using a @Injectable decorator. A function defined inside a service can be invoked from any component or directive.

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

@Injectable({

providedIn: 'root'

})

**Modules:**  
A module is a place where we can group components, directives, services, and pipes. Module decides whether the components, directives, etc can be used by other modules, by exporting or hiding these elements. Every module is defined with a @NgModule decorator.  
By default, modules are of two types:

 Root Module

 Feature ModuleEvery application can have only one root module whereas, it can have one or more feature modules.  
A root module imports BrowserModule, whereas a feature module imports CommonModule.

import { BrowserModule } from '@angular/platform-browser';

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

import { CommonModule } from '@angular/common';

Declaration[] --- it contains the list of components that we have created and wants to use in our project.

Imports[] ---- it contains the list of modules we want to use

Providers [] ---- all services

Bootstrap[] -----the root component that is to be loaded .

entryComponent[] ------ contains the list of component that need to be created dynamically .

We cannot use components of one module into other modules.

1. **What is Tree Shaking in Angular Services ?**

Tree Shaking is a Process wherein the unused code is removed during the build phase. Tree Shaking Ensures that our application includes the code which is needed. Angular version 6 introduced a new feature, Tree Shakeable Providers which helps us to enhance the performance of our application.

Using providedIn: 'root' implements tree shaking mechanism.

1. **What is the use Gaurds ?**

There are 5 types of Route Guards we use in Angular.

* **CanActivate :** This helps to decide whether the route can be activated or not. 1
* **CanActivateChild :** This helps to decide whether child routes can be activated or not.
* **CanDeactivate :** This helps to decide whether the route can be deactivated or not. Generally, this is used to warn a user if they are navigating from the component.
* **Resolve :** This performs route data retrieval before any activation of the route.
* **CanLoad :** Checks to see if the user can navigate to the module which is lazily loaded.

1. **What is canActivateChild route guard ?**

This guard checks the access on child routes.CanActivateChild decides whether we can navigate to child routes or not. It is used to decide link access on the basis of authorization. It is possible that those links accessible to ADMIN role, will be not be

allowed to USER role.

Here parent and all childs both gets guarded :

path: 'student', canActivateChild:[childauthGuard],

children: [

{ path: ', component: StudentComponent I,

{ path: 'details', component: StudentdetailsComponent },

{ path: 'registration', component: StudentregistrationComponent }

]

**Here only Child components are guarded.**

path: 'student',

children: [

{ path: '', component: StudentComponent },

path: "', canActivateChild: [childauthGuard],

children: [

{ path: 'details', component: StudentdetailsComponent },I

{ path: 'registration', component: StudentregistrationComponent }

]

]

1. **What commands do you to run test ?**

Ng test ------ for unit test

Ng e2e ------ for end to end test

1. **How do you upgrade angular apps ?**

ng update @angular/cli @angular/core

1. **How to convert TypeScript into JavaScript ?**

It is called Transpiling . Even though Typescript is used for writing code in Angular applications , it gets internally transpiled into equivalent JavaScript.

1. **How to check CLI version ?**

Ng –version

1. **What is ng lint command ?**

It is used for code syntax linting , for checking whether the code is following coding standards. Like missing whitespaces , comment must start with a space .

1. **How can we get the window height and width in components ?**

Height=window.height

Width=window.width

1. **Explain string interpolation and property binding in Angular.**

String interpolation and property binding are parts of data-binding in Angular.  
Data-binding is a feature in angular, which provides a way to communicate between the component (Model) and its view(HTML template).  
Data-binding can be done in two ways, one-way binding and two-way binding.  
In Angular, data from the component can be inserted inside the HTML template. In one-way binding, any changes in the component will directly reflect inside the HTML template but, vice-versa is not possible. Whereas, it is possible in two-way binding.  
String interpolation and property binding allow only one-way data binding.  
String interpolation uses the double curly braces {{ }} to display data from the component. Angular automatically runs the expression written inside the curly braces, for example, {{ 2 + 2 }} will be evaluated by Angular and the output 4, will be displayed inside the HTML template. Using property binding, we can bind the DOM properties of an HTML element to a component's property.

**Property binding uses the square brackets [ ] syntax.**

**<h1>**

**Welcome to {{ title }}**

**</h1>**

**<span [innerHTML]='title'></span>**

**First name: <span [innerHTML]='firstname'></span>**

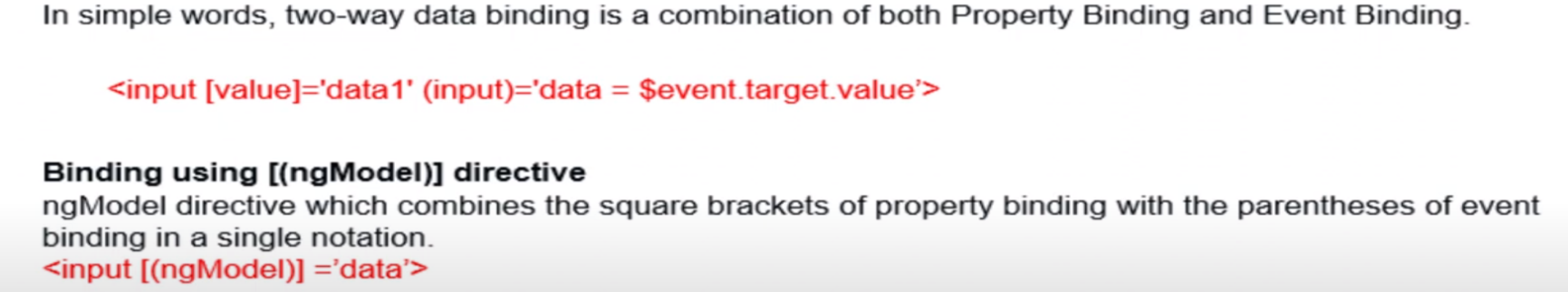
**<img [src]="'https://angular.io/assets/images/logos/angular/angular.svg'"| >**

1. **What is two way data binding or what is ngModel ?**

Two-way binding mainly used in the input type field or any form element where user type or provide any value or change any control value in the one side, and on the other side, the same automatically updated in to the controller variables and vice versa.

import { FormsModule } from '@angular/forms';

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

****

1. **Difference between [(ngModel)] and [ngModel] for binding state to property?**

[(ngModel)]="overRideRate" is the short form of [ngModel]="overRideRate" and (ngModelChange)="overRideRate = $event"

[ngModel]="overRideRate" is to bind overRideRate to the input.value

(ngModelChange)="overRideRate = $event" is to update overRideRate with the value of input.value when the change event was emitted.

Together they are what Angular2 provides for two-way binding.

1. **What are lifecycle hooks in Angular? Explain a few lifecycle hooks.**

Every component in Angular has a lifecycle, different phases it goes through from the time of creation to the time it's destroyed. Angular provides hooks to tap into these phases and trigger changes at specific phases in a lifecycle

* **ngOnChanges( )** This hook/method is called before ngOnInit and whenever one or more input properties of the component changes.  
  This method/hook receives a SimpleChanges object which contains the previous and current values of the property.
* **ngOnInit( )** This hook gets called once, after the ngOnChanges hook.  
  It initializes the component and sets the input properties of the component.
* **ngDoCheck( )** It gets called after ngOnChanges and ngOnInit and is used to detect and act on changes that cannot be detected by Angular.  
  We can implement our change detection algorithm in this hook.
* **ngAfterContentInit( )** It gets called after the first ngDoCheck hook. This hook responds after the content gets projected inside the component.
* **ngAfterContentChecked( )** It gets called after ngAfterContentInit and every subsequent ngDoCheck. It responds after the projected content is checked.
* **ngAfterViewInit( )** It responds after a component's view, or a child component's view is initialized.
* **ngAfterViewChecked( )** It gets called after ngAfterViewInit , and it responds after the component's view, or the child component's view is checked.
* **ngOnDestroy( )** It gets called just before Angular destroys the component. This hook can be used to clean up the code and detach event handlers.

1. **How are Angular expressions different from JavaScript expressions?**

* Angular expressions are evaluated against a local scope object whereas JavaScript expressions against global window object.
* Angular expressions one cannot use loops, conditionals and exceptions.
* Angular expressions quite beneficial is the use of pipes

1. **How to disable the prefix to any component in Angular CLI command ?**

ng generate component <name> --prefix=false

1. **Angular by default, uses client-side rendering for its applications. Can one make an angular application to render on the server-side? Or What is Angular Universal ?**

Yes, angular provides a technology called Angular Universal, which can be used to render applications on the server-side. Angular Universal executes on the *server*, generating *static* application pages that later get bootstrapped on the client. This means that the application generally renders more quickly, giving users a chance to view the application layout before it becomes fully interactive.  
  
**The advantages of using Angular Universal are:**

1) Facilitate web crawlers (SEO)

Google, Bing, Facebook, Twitter, and other social media sites rely on web crawlers to index our application content and make that content searchable on the web. These web crawlers may be unable to navigate and index our highly interactive Angular application. Angular Universal can generate a static version of your application that is easily searchable, linkable, and navigable without JavaScript. Universal also makes a site preview available since each URL returns a fully rendered page.

2) Some devices don't support JavaScript or execute JavaScript so poorly that the user experience is unacceptable. For these cases, you may require a server-rendered, no-JavaScript version of the application.

3) Show the first page quickly, displaying the first page quickly can be critical for user engagement. Pages that load faster perform better. A static version of the landing page to hold the user's attention. At the same time, you'll load the full Angular application behind it. The user perceives near-instant performance from the landing page and gets the full interactive experience after the full application loads.

ng add @nguniversal/express-engine ------- To add Angular Universal into ourProject

npm run dev:ssr ------- To run our application

1. **What is nestJS ?**

NestJS is server side framework used to create api and can be used to add universal into angular application .

1. **How to deploy an Angular Universal App ?**

We can't deploy an Angular Universal app to a static host (i.e. Firebase Hosting, AWS S3 etc will NOT work). The reason for this is, that you're using Node.js to pre-render pages on the server and those Hosts don't support Node.js.Hence you need a host that does - for example AWS ElasticBeanstalk or Heroku.

To these hosts, you need to upload your dist/ folder along with the package.json file. On the web server, you then have to ensure that npm install is executed, followed by npm serve:ssr.

That's it - your app is now up and running on a web server!

1. **Can we use Browser Or DOM based api in SSR ?**

You MUST NOT use any browser-only APIs like document.querySelector() in your Angular code! .Simply because it will execute on the server and there, such APIs are not available.

1. **How to optimize an angular application ?**

1) Consider lazy loading instead of fully bundled app if the app size is more.

2) We should not have unnecessary import in our application.

3) Application should be bundled and uglified.

4) We should consider AOT compilation.

5) Using angular Universal for better user experience.

1. **What is NgRX ?**

NgRx stands for Angular Reactive Extensions. NgRx is a state management system that is based on the Redux pattern.

NgRx is written specifically for Angular application as a state management solution.

1. **What is a state in angular application ?**

Application state is the entire memory of the application. In simple terms, application state is composed of data received by API calls, user inputs, presentation UI state, application preferences, etc.

An Angular application is typically made up of many components. Each of these components has its own state and has no awareness of the state of the other components. In order to share information between parent-child components, we use @Input and@Output decorators. However, this approach is practical only if your application consists of a few components.

1. **What is REdux ?**

Redux is a pattern that’s used to simplify the state management process in JavaScript applications (not just for Angular). Redux is primarily based on three main principles :--

1. Single source of truth — This means the state of your application is stored in an object tree within a single store. The store is responsible for storing the data and providing the data to components whenever requested.
2. Read-only state — In other words, state is immutable. This doesn’t necessarily mean that state is always constant and cannot be changed. It only implies that you are not allowed to change the state directly. In order to make changes in the state, you have to dispatch actions (which we will discuss in detail later) from different parts of your application to the store.
3. State is modified with pure functions — Dispatching actions will trigger a set of pure functions called reducers. Reducers are responsible for modifying the state in different ways based on the action received. A key thing to note here is that a reducer always returns a new state object with the modifications.
4. **What is default module bundler in Angular applications?**

Webpack.

Other packing tools are : Gulp,Babel,Grunt

1. **Can we develop an application without a module?**

No we need atleast one module in every application that will be bootsraped generally it is app module by default.

1. **How to create a module in angular ?**

Using : ng generate module <name>

1. **How to export a component from a module ?**

Include the component name in the list of exports array in module.ts file.

1. **What is Angular Ivy ?**

Ivy is a very important stepping stone in the Angular history. It changes how the framework internally works, without changing how we write Angular applications.

Angular achieves the same with this effort: Ivy is a complete rewrite of the compiler (and runtime) in order to:

* 🚀reach better build times (with a more incremental compilation)
* 🔥reach better build sizes (with a generated code more compatible with tree-shaking)
* 🔓unlock new potential features (metaprogramming or higher order components, lazy loading of component instead of modules, a new change detection system not based on zone.js…).

1. **How are observables different from promises?**

The first difference is that an Observable is lazy whereas a Promise is eager.

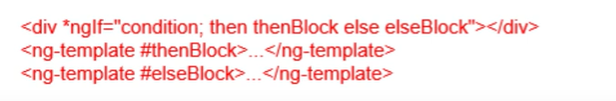
|  |  |
| --- | --- |
| Promise | Observable |
| Emits a single value | Emits multiple values over a period of time |
| Not Lazy | Lazy. An observable is not called until we subscribe to the observable |
| Cannot be cancelled | Can be cancelled by using the unsubscribe() method |
|  | Observable provides operators like map, forEach, filter, reduce, retry, retryWhen etc. |

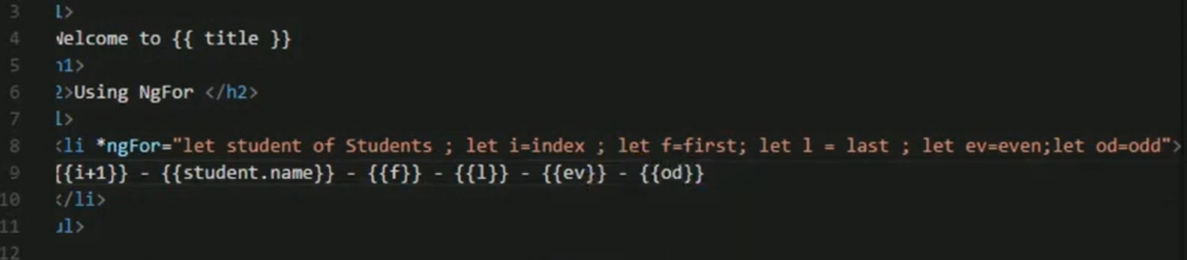
Promises are always asynchronous. Even when the promise is immediately resolved. Whereas an Observable, can be both synchronous and asynchronous.

1. **When to use a directive?**

Consider an application, where multiple components need to have similar functionalities. The norm thing to do is by adding this functionality individually to every component but, this task is tedious to perform. In such a situation, one can create a directive having the required functionality and then, import the directive to components which require this functionality.  
**Types of directives:  
Component directives**These form the main class in directives. Instead of @Directive decorator we use @Component decorator to declare these directives. These directives have a view, a stylesheet and a selector property.

**Structural directives**These directives are generally used to manipulate DOM elements.  
Every structural directive has a ‘ \* ’ sign before them.  
We can apply these directives to any DOM element.

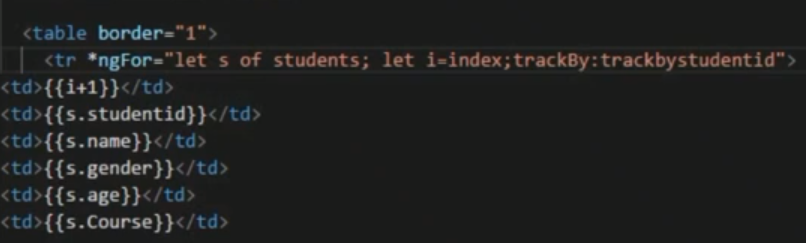


****

**Attribute Directives**  
These directives are used to change the look and behavior of a DOM element. Ex : ngClass ,ngStyle,etc

1. **What is trackBy in ngFor ?**

The use of trackBy it's a performance optimization and is usually not needed by default, it's in principle only needed if running into performance issues.Suppose we have some data coming from some API request into the collection and we need to change the data over the web page using ngFor directive. In this case, Angular 7 will remove all the DOM elements that associated with the data and will create them again in the DOM tree, even the same data is coming. That means a lot of Dom manipulations will happen if a large amount of data coming from the API.

****

**trackbystudentid(index:number, student:any): string{**

**return student.studentid;**

**}**

1. **What are directives in Angular?**

A directive is a class in Angular that is declared with a @Directive decorator.  
Every directive has its own behaviour and can be imported into various components of an application.

1. **What is ngClass ?**

The **Angular ngClass** Directive is an [Angular Attribute Directive](https://www.tektutorialshub.com/angular/angular-directives/#attribute-directives), which allows us to add or remove CSS classes to an HTML element. Using ngClass you can create dynamic styles in angular components by using conditional expressions.

Adding style in .css

.red { color: red; }

.size20 { font-size: 20px; }

And then inside our html :

<div [ngClass]="'red size20'"> Red Text with Size 20px </div>

Or we can use array : <div [ngClass]="['red','size20']">Red Text with Size 20px </div>

We Can Use ngClass in following ways :

1. ngClass with string
2. ngClass with array
3. ngClass with object
4. ngClass with component method
5. **What is ngStyle ?**

The Angular ngStyle directive allows us to set the many inline style of a HTML element using an expression. The expression can be evaluated at run time allowing us to dynamically change the style of our HTML element.

**<input [(ngModel)]="color" />**

**<div [ngStyle]="{'color': color}">Change my color</div>**

**div [ngStyle]="{'background-color':status === 'error' ? 'red' : 'blue' }"></<div>**

1. **What is StyleBinding ?**

The Style Binding uses the [] brackets. Place the CSS Style property (binding target) inside the square bracket. The CSS Style property must begin with ‘Style’ followed by a dot (.) and then style name.

<button [style.border]="'5px solid yellow'">Save</button>

<button [style.color]="getColor()">Button 2</button>

1. **How can be create a custom directives ?**

1)We can create a file with .ts extension like demo\_directives.ts .

2)Now add a selector inside @directive() decorator.

3)Create a class and define functionality of your directives.

Or ng generate directive betterHighlight and skip 1,2,3

4)Now inside app.module.ts we will import custom directive and add it in declarations[] array.

5)Add finally add the directive as an attribute for any html tag.

Example :

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

@Directive({

selector: '[appDropdown]'

})

export class DropdownDirective {

@HostBinding('class.open') isOpen = false;

@HostListener('document:click', ['$event']) toggleOpen(event: Event) {

this.isOpen = this.elRef.nativeElement.contains(event.target) ? !this.isOpen : false;

}

constructor(private elRef: ElementRef) {}

}

1. **What is Renderer2 of Angular Core?**

It is used to create custom directives using it methods like setStyle,setProperty,removeStyle………………. by using ElementRef.nativeElement

1. **What is @HostListener of Angular Core ?**

 Angular will invoke the decorated method when the host element emits the specified event.

@HostListener(mouseenter) mouseover(eventData:Event) {

}

1. **What are types of Angular Decorator ?**

The whole point of a decorator is to store metadata about a class, method or property. When you configure a component for example, you’re providing metadata for that class that tells Angular that we have a component, and that component has a specific configuration.

1. Class decorators, e.g. @Component and @NgModule
2. Property decorators for properties inside classes, e.g. @Input and @Output
3. Method decorators for methods inside classes, e.g. @HostListener
4. Parameter decorators for parameters inside class constructors, e.g**. @Inject**
5. **What is @HostBinding ?**

HostBinding - Declares a host property binding. Angular automatically checks host property bindings during change detection. If a binding changes, it will update the host element of the directive.

@HostBinding - will bind the property to the host element, If a binding changes, HostBinding will update the host element. Ex : @HostBinding (style.backgroundColor) colorname:String=”default name of color”

By using @hostbinding and @HostListener we can eleiminate the use of Renderer2.

We can also dynamically bind value of our style from the component we are using our directive by using

@Input defaultColor:String=”transparent”

We can also set an alias to our @Input like @Input(‘alias1’)

1. **How Can we create a custom Structural Directive ?**

1)We can create a file with .ts extension like demo\_directives.ts .

2)Now add a selector inside @directive() decorator.

3)Create a class and define functionality of your directives.

Or ng generate directive betterHighlight and skip 1,2,3

4)Now inside app.module.ts we will import custom directive and add it in declarations[] array.

5)And finally add the directive as an attribute for any html tag.

1. **What is templateRef and ViewContainerRef ?**

templateRef is used to get access of the template where we want to use our custom Structural Directive.

ViewContainerRef used to get access of the div where we want to use our custom Structural Directive.

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

if (!condition) {

this.vcRef.createEmbeddedView(this.templateRef);

} else {

this.vcRef.clear();

}

}

1. **What is Singleton Object Concept and Multiple Object Concept?**

* **Singleton Object Concept:** When we register service in root level using provideIn or provider[] in module.ts then we may have different object name of the service in two different component but still they will share the same service object.
* **Multiple Object Concept:** When we register service in component level using provider[] in component.ts inside @component decorator of each component , then they will not share the same service object. Each component will may different service object for themselves.

1. **Explain the concept of Dependency Injection?**

Dependency injection is an application design pattern which is implemented by Angular.  
It also forms one of the core concepts of Angular. Dependencies in angular are nothing but services which have a functionality. Functionality of a service, can be needed by various components and directives in an application. Angular provides a smooth mechanism by which we can inject these dependencies in our components and directives.  
So basically, we are just making dependencies which are injectable across all components of an application.

1. **What is providedIn** ?

**With the help of** providedIn we don’t need to manually register our service in any components or module. It gets automatically called whenever the component is loaded.

Earlier in before Angular 7, User have to import all services in module.ts and then had to write service name in provider[].

Problem with this is that,even if service is not used ,as soon as module is loading starts service gets loaded first.

With the help of providedIn , we don’t need to import services in module.ts file hence speeding app start.

1. **How can we use services to communicate between components ?**

While creating services, we can also add ----- statusUpdated = new EventEmitter<string>();

So using eventEmitter we will catch the event where this service will be used………

**this.accountsService.statusUpdated.emit(status);**

And this emitted event string will be subscribe inside other component ………..

**this.accountsService.statusUpdated.subscribe(**

**(status: string) => alert('New Status: ' + status)**

**);**

1. **What are Angular Animations ?**

Animation involves transition from one state of an element to another state. Angular defines three different states for an element:

Void state — represents the state of an element which is not part of the DOM. This state occurs when an element is created but not yet placed in the DOM or the element is removed from the DOM.

The wildcard state — This is also known as the default state of the element. The styles defined for this state are applicable to the element regardless of its current animation state.

Custom state — This is the custom state of the element and it needs to be defined explicitly in the code.

1. **How can we use animation module of Angular ?**

To get started with adding Angular animations to your project,

Step 1: Enabling the animations module

Import BrowserAnimationsModule, which introduces the animation capabilities into your Angular root application module.

Step 2: Importing animation functions into component files

import { trigger, state, style, animate, transition, // ...} from '@angular/animations';

Step 3: Adding the animation metadata property

In the component file, add a metadata property called animations: within the @Component() decorator. You put the trigger that defines an animation within the animations metadata property.

@Component({

// other component properties.

animations: [

trigger('triggerName'), [

state('stateName', style())

transition('stateChangeExpression', [Animation Steps])

]

]

})

1. **How can we create an animations ?**

we have defined a trigger changeDivSize and two state functions inside the trigger.

animations: [

trigger('changeDivSize', [

state('initial', style({

backgroundColor: 'green',

width: '100px',

height: '100px'

})),

state('final', style({

backgroundColor: 'red',

width: '200px',

height: '200px'

})),

transition('initial=>final', animate('1500ms')),

transition('final=>initial', animate('1000ms'))

]),

]

To change the state of our element we will define a function in the class definition of our component. Include the following code in the AnimationdemoComponent class

currentState = 'initial';

changeState() {

this.currentState = this.currentState === 'initial' ? 'final' : 'initial';

}

Open animationdemo.component.html file and add the following code:

<h3>Change the div size</h3>

<button (click)="changeState()">Change Size</button>

<br />

<div [@changeDivSize]=currentState></div>

<br />

1. **How can we apply ballon effect animations ?**

In this section, we will learn how to change the size from all directions. It will be similar to inflating and deflating a balloon,

trigger('balloonEffect', [

state('initial', style({

backgroundColor: 'green',

transform: 'scale(1)'

})),

state('final', style({

backgroundColor: 'red',

transform: 'scale(1.5)'

})),

transition('final=>initial', animate('1000ms')),

transition('initial=>final', animate('1500ms'))

]),

we are using the transform property to change the size from all directions.

<h3>Balloon Effect</h3>

<div (click)="changeState()"

style="width:100px;height:100px; border-radius: 100%; margin: 3rem; background-color: green"

[@balloonEffect]=currentState>

</div>

1. **How to add animation while adding or removing item to list ?**

### Fade In and Fade Out animation

### listItem = [];

### list\_order: number = 1;

### addItem() {

### var listitem = "ListItem " + this.list\_order;

### this.list\_order++;

### this.listItem.push(listitem);

### }

### removeItem() {

### this.listItem.length -= 1;

### }

Add the following trigger definition in the animation property:

trigger('fadeInOut', [

state('void', style({

opacity: 0

})),

transition('void <=> \*', animate(1000)),

]),

### Now adding out html element :

<h3>Fade-In and Fade-Out animation</h3>

<button (click)="addItem()">Add List</button>

<button (click)="removeItem()">Remove List</button>

<div style="width:200px; margin-left: 20px">

<ul>

<li \*ngFor="let list of listItem" [@fadeInOut]>

{{list}}

</li>

</ul>

</div>

1. **What are aliases for Void to \* and \* to void ?**

for void => \* we can use ‘:enter’

for \* => void we can use ‘:leave’

trigger('EnterLeave', [

state('flyIn', style({ transform: 'translateX(0)' })),

transition(':enter', [

style({ transform: 'translateX(-100%)' }),

animate('0.5s 300ms ease-in')

]),

transition(':leave', [

animate('0.3s ease-out', style({ transform: 'translateX(100%)' }))

])

])

Here we have defined the trigger EnterLeave. The ‘:enter’ transition will wait for 300ms and then run for 0.5s with an ease-in effect. Whereas the ‘:leave transition will run for 0.3s with an ease-out effect.

<ul>

<li \*ngFor="let list of listItem" [@EnterLeave]="'flyIn'">

{{list}}

</li>

</ul>

1. **What is keyframes in animation ?**

Defines a set of animation styles, associating each style with an optional offset value. Use with the animate() call. Instead of applying animations from the current state to the destination state, keyframes describe how each style entry is applied and at what point within the animation arc.

**animate("5s", keyframes([**

**style({ backgroundColor: "red", offset: 0 }),**

**style({ backgroundColor: "blue", offset: 0.2 }),**

**style({ backgroundColor: "orange", offset: 0.3 }),**

**style({ backgroundColor: "black", offset: 1 })**

**]))**

If there are no offset values specified in the style entries, the offsets are calculated automatically.

1. **What are animations events or Callbacks?**

**In the HTML template, the animation event is passed back via $event, as @trigger.start and @trigger.done, where trigger is the name of the trigger being used**

<div [@openClose]="isOpen ? 'open' : 'closed'"

(@openClose.start)="onAnimationEvent($event)"

(@openClose.done)="onAnimationEvent($event)"

class="open-close-container">

</div>

An animation can influence an end user to perceive the operation as faster, even when it isn't. Thus, a simple animation can be a cost-effective way to keep users happy, rather than seeking to improve the speed of a server call and having to compensate for circumstances beyond your control, such as an unreliable network connection.

1. **What are few functions for complex animations control ?**

Functions that control complex animation sequences are as follows:

query() finds one or more inner HTML elements.

stagger() applies a cascading delay to animations for multiple elements.

group() runs multiple animation steps in parallel.

sequence() runs animation steps one after another.

1. **How can we create reusable animations ?**

To create a reusable animation, use the animation() method to define an animation in a separate .ts file and declare this animation definition as a const export variable. You can then import and reuse this animation in any of your application components using the useAnimation() API.

import { animation, style, animate, trigger, transition, useAnimation } from '@angular/animations';

export const transitionAnimation = animation([

style({

height: '{{ height }}',

opacity: '{{ opacity }}',

backgroundColor: '{{ backgroundColor }}'

}),

animate('{{ time }}')

]);

**transAnimation is made reusable by declaring it as an export variable..**

**You can also export a part of an animation.** For example, the following snippet exports the animation trigger.

import { animation, style, animate, trigger, transition, useAnimation } from '@angular/animations';

export const triggerAnimation = trigger('openClose', [

transition('open => closed', [

useAnimation(transitionAnimation, {

params: {

height: 0,

opacity: 1,

backgroundColor: 'red',

time: '1s'

}

})

])

]);

Now we will import it in our componets…………………

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

import { transition, trigger, useAnimation } from '@angular/animations';

import { transAnimation } from './animations';

@Component({

selector: 'app-open-close-reusable',

animations: [

trigger('openClose', [

transition('open => closed', [

useAnimation(transAnimation, {

params: {

height: 0,

opacity: 1,

backgroundColor: 'red',

time: '1s'

}

})

])

])

],

1. **How can we animate the Router outlet ?**

Animating this route transition can greatly enhance the user experience.

1. **What is ComponentFactoryResolver ?**

A simple registry that maps Components to generated ComponentFactory classes that can be used to create instances of components. Use to obtain the factory for a given component type, then use the factory's create() method to create a component of that type.

It is used to load a component without using routing or component selector. We take help of ViewContainerRef and templateRef to create component.

Like when we want to load an alert component when an error occurs.

1. **What is ModuleMapLoaderModule ?**

Module Map NgFactory Loader

This is a NgFactory Loader which uses a map of modules instead of resolving modules lazily.This is useful when executing in node because lazy loading serves no purpose..

npm install @nguniversal/module-map-ngfactory-loader –save

Add ModuleMapLoaderModule to your server module.This is important for lazy module loading on server side.